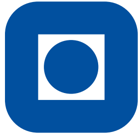
Norwegian University of
Science and Technology

Data-oriented Design approach for processor intensive games

Walid Faryabi

Master of Science in Cybernetics and Robotics
Submission date: September 2018
Supervisor: Geir Mathisen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics



NTNU – Trondheim
Norwegian University of
Science and Technology

MASTER THESIS

Data-oriented design for processor intensive games

Author:
Walid FARYABI

Supervisor:
Geir MATHISEN

September 2, 2018

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

The gap between processor and memory speeds have motivated for an alternative method for software development with focus on data and efficient use of memory, data-oriented design. The primary objective of this design is to utilize the slower memory units in a more efficient way through less cache-misses. The focus is solely on the data in an application and the way they are stored in memory. This design will be compared against the popular programming paradigm, object-oriented programming, to analyze whether a memory-focused application will perform better. This thesis will primarily focus on processor intensive applications where the processor must continuously write and read data from the memory units. This type of application is most commonly found with real-time applications such as video-games, which will be the main focus area for this thesis.

This thesis will present three different applications that will be used to compare the two different programming paradigms, with the goal of comparing which implementation performs better. The first application will involve the implementation of an architectural pattern, the entity-component-system, which can be combined with data-oriented principles to create more processor efficient applications. The second application will involve a simulation implemented in Unity with data-oriented design. The final part will involve conversion of an existing video-game that is object-oriented, into a more data-oriented solution. A pure data-oriented solution was not achieved for the conversion due to limitations imposed by the data-oriented features in the game engine, as a result of the engine being in experimental stage.

The results indicate that the data-oriented design performs better in cases where the processor must perform work on a large set of data. The design is more optimized for applications that require same type of work on large data sets as this allows for better spatial locality. Furthermore, the use of data-oriented principles allows for well established separation between data and logic, making it easier to introduce new type of logic and data into an application. However, the results gathered can not be completely attributed to the underlying design, as there are external factors impacting performance due to the development environments used.

Problem Statement

Object-oriented design is a widely used programming paradigm in the video game industry. The concept of objects allows for a higher degree of modularity and readability. However, this approach is not efficient when it comes to memory access, especially for large memory intensive applications such as video games. A proposition is to use data-oriented design, where the application is designed with focus on data instead of objects. This design approach has some several advantages, such as easier parallelization and better locality of reference. Performance can be improved by reducing the number of cache misses through data-oriented design, which will as a result save number of clock cycles spent for fetching data. This could in theory give better performance in different variety of applications such as video-games.

Based on the statement above, this thesis will go through the following points:

- Relevant technology around data-oriented design will be researched.
- Performance of object-oriented design versus data-oriented design will be investigated.
- Based on the previous points, a new design for an existing video-game currently in development will be suggested.
- As far as time permits, the suggested design will be implemented.

Preface

This thesis is written for the department of engineering cybernethics at the Norwegian University of Science and Technology. All students of the department are required to hand in a master-thesis in their last semester in order to complete their master degree. Part of the work described in this thesis is in cooperation with Pineleaf Studio, a video-game development company.

The idea behind this thesis was given by their technical leader, Fredrik Chrislock, who wanted to analyze alternative methods of optimization for their server due to costs. The company provided me with their codebase for a video-game currently in development. The existing codebase was modified by me to experiment with data-oriented principles. As a part of the cooperation, I was allowed to stay in their offices in Trondheim from April to end of June. Throughout this period I received guidance from Fredrik when it came to general game design, the architecture of their game, use of Unity and potential improvements in efficiency. The work related to data-oriented design principles were solely done by me, as I worked indepedently with the codebase. Unfortunately, the results from this thesis were not discussed with the company due to me staying in Oslo for the last months of this thesis. Furthermore, all other work not related to the video-game were solely done by me with no input from the company.

Few number of tools were required for this thesis. For hardware, only a computer with a graphics processing unit was needed. For software, Unity was used as a game engine and Microsoft Visual Studio as IDE. This thesis was officially started on the 9th of April and concluded on 3rd of September.

Walid Faryabi July 2018 ________________________________

Acknowledgment

I would like to thank Geir Mathisen for accepting this thesis and being my supervisor. His monthly meetings and guidance on writing this thesis made it easier for me to get through.
I would also like to give thanks to the whole Dwarfheim team for assisting me with this thesis and giving me the chance to have a very fun thesis. A special thanks is also given to Fredrik Chrislock for guiding me through the thesis and coming with the problem description.
Finally, I would like to give thanks to my parents, Hassan and Zohrah, for making my life easier by providing with all the basic necessities required, allowing me to focus solely on writing this thesis. It would be a lot more difficult without their support.

W.F

Contents

Summary and Conclusions i
Preface iii
Acknowledgment iv

1 Introduction **3**
1.1 Problem Formulation 4
1.2 Goal Of This Master Thesis 4
1.3 Required work for this thesis 5
1.4 Cooperation with Pineleaf Studio 6

2 Literature Study and Theory **9**
2.1 Memory in modern computers 9
2.2 Data-Oriented Design 13
2.2.1 Data-oriented design principles 13
2.2.2 Entity Component System 18

3 Functional Specification and Evaluation Criteria **23**
3.1 Evaluation criteria 23
3.1.1 Frame Rate 24
3.1.2 Cpu usage time 24
3.2 Entity-component-system in C# 25
3.2.1 Specifications 25
3.2.2 Evaluating against object-oriented programming 25
3.3 Entity-component-system in Unity - Pure data-oriented solution 26
3.3.1 Evaluation of the application 26
3.4 Converting DwarfHeim into a data-oriented solution 28
3.4.1 Specifications for conversion 28
3.4.2 Evaluation of the conversion 29

4 Materials and Methods **31**
4.1 Development Environment 31
4.1.1 Game Engine - Unity 31
4.1.2 Different terminologies and concepts in Unity 32
4.1.3 Scripting in Unity - Adding behaviour to game objects 34
4.1.4 Analyzing performance - Unity Profiler 35
4.1.5 Programming Language - C# 36
4.1.6 Programming Language - Python 36
4.2 Measuring the frame rate 38
4.2.1 Measurement of frame rate for Unity 38
4.2.2 Frame rate counter outside Unity 38
4.2.3 Refresh rate 39
4.3 Entity Component System - Custom Implementation 39
4.3.1 Initial entity-component-system architecture details 39
4.3.2 Improved design 43
4.3.3 Functional testing of the ECS implementation 46
4.3.4 Performance tests for the ECS implementation 47
4.3.5 Integrating OpenGL with the ECS implementation 49
4.3.6 Test Application using the entity-component-system implementation 52
4.3.7 Simulating the sine wave 53
4.3.8 Simulating a sine wave using opengl and object-oriented principles 54
4.3.9 Simulating a sine wave using opengl and the custom entity-component-system implementation 54
4.3.10 The sine wave simulation tests 55
4.4 Entity-Component-System in Unity 57
4.5 Pure data-oriented application in Unity 59
4.5.1 Objected-oriented sine wave 59
4.5.2 Data-oriented Sine wave 60
4.5.3 Testing of the sine-wave simulations 62
4.6 Data-oriented design for Dwarfheim 63
4.6.1 Computer architecture of Hybrid/Pinecone 63
4.6.2 Methodology for converting to data-oriented design 72
4.6.3 Making it more applicable on a server 88
4.6.4 Testing 89

5 Results 93
5.1 Entity-component-system implementation . . . 93
5.1.1 Functional Test . . . 93
5.1.2 Performance tests for the different versions of ECS . . . 93
5.1.3 OpenGL sine wave simulation tests . . . 94
5.2 Sine wave simulation . . . 94
5.2.1 Sine wave simulation results . . . 96
5.3 DwarfHeim Conversion . . . 97
5.3.1 Functional results . . . 98
5.3.2 Performance test . . . 99

6 Discussion 105
6.1 Discussion of Results . . . 105
6.1.1 Custom C# implementation of Entity-Component-System . . . 105
6.1.2 Limitations of the custom ECS implementation . . . 108
6.1.3 Meeting the specifications . . . 108
6.1.4 Potential issues with the current design . . . 108
6.1.5 Sine-wave simulation results in Unity . . . 109
6.2 DwarfHeim conversion to a more data-oriented design . . . 112
6.2.1 Functional features of the data-oriented design . . . 112
6.2.2 Performance results . . . 112
6.2.3 Inspecting time values for the converted parts . . . 114
6.2.4 A hybrid solution vs pure data-oriented . . . 115
6.2.5 The implications of the research . . . 115
6.3 General results . . . 116
6.4 Developing with the entity-component-system . . . 117

7 Conclusion 119

8 Further Work 121
8.1 Recommendations for the custom entity-component-system . . . 121
8.2 Recommendations for the DwarfHeim conversion . . . 122

Bibliography 123

A Acronyms 127

B Additional Information 129
B.1 Concepts in Unity - Some additional concepts . . . 129
B.1.1 Graphics in Unity . . . 130

B.2 Simple example of scripting in Unity . . . 135
B.3 ECS custom implementation . . . 137
B.3.1 Example of system structure . . . 137
B.3.2 Example of inject-attribute with componentDataArray . . . 139
B.3.3 Optimization steps . . . 139
B.3.4 Reducing number of boxing and unboxing . . . 140
B.3.5 Reducing number of function calls through another class . . . 140
B.4 OpenGl program code . . . 141
B.4.1 Shader code . . . 141
B.4.2 Code for drawing a simple triangle. . . . 142
B.5 Prototype-based programming . . . 142
B.6 Programming language C# and its features . . . 143
B.7 Sine-wave simulation in Unity . . . 147
B.7.1 Accessing mesh and material data with entity-component-system . 147
B.7.2 Sine-wave simulation, profiler stats with profiler data exporter . . . 148
B.8 Unity profiler data exporter results for DwarfHeim . . . 148

Bibliography **169**

Chapter 1

Introduction

Performance of computers are rapidly growing. Processors are getting more complex with multiple cores, faster clock speeds and larger internal caches. Memory units such as random-access-memory is getting better access times, larger bandwidth and larger capacity. The improvements in performance allows for more complex applications with better performance. However, the gap between processor performance and memory performance is growing, with processor performance outperforming memory at a faster rate(1). As this gap increases, the more powerful processors can be stalled by the memory units and its access times.

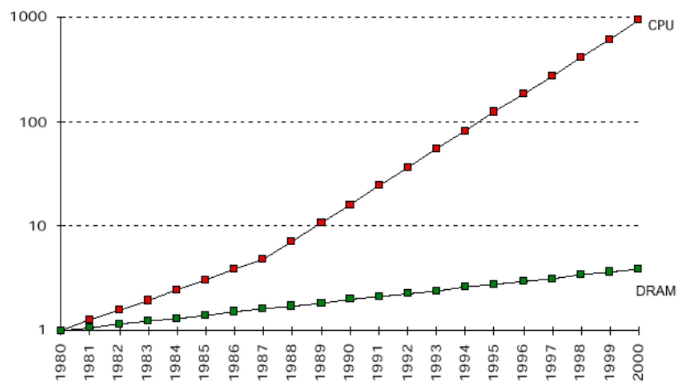


Figure 1.1: Performance gap between processor and memory (1)

A large amount of applications are developed in an object-oriented manner. Data fields and the logic around these are encapsulated into "objects". This type of design paradigm have several benefits such as re-usability, less maintenance and design benefits that al-

lows large teams to work together efficiently. One could also argue in favor of object-oriented design by the claim that working with objects are more intuitive as they can represent real world attributes in a more intuitive way for humans. However, this design paradigm do not care about the performance of the memory. In this kind of design, the objects are normally stored on a data structure called heap, which allocates objects on random addresses on the main memory unit(2). The lack of coherence between placement of objects on the main memory increases cache misses. This can have detrimental effect on the performance for applications with a significant number of dynamic allocations. This is especially the case for video-games.

The performance of complex applications will be more affected by the memory unit as the performance gap increases. This holds true if applications are designed in an object-oriented way. For this reason, another design paradigm will be researched and analyzed in this thesis, data-oriented design. Data-oriented-design is a programming paradigm where the applications are designed around data instead of objects. Data layout and how data flows becomes the fundamental criteria of the design. The focus of this approach is to reduce cache misses by inspecting how data related to each other is used and laid out on the memory. This design focuses more on the limitation of the memory unit and how the processor fetches data.

1.1 Problem Formulation

Given the introduction above, the problem formulation can be stated as the following:

> «How does data-oriented design compare to object-oriented design in computer games, when it comes to performance?»

In this thesis, the performance criteria will mostly be based on frame rate. Furthermore, the usability of data-oriented design will also be discussed. The scope of this thesis is only limited to few cases. These cases involves scenarios where large number of objects are rendered. Due to this limitation, this thesis not applicable for all types of video game, such as games with simpler structure consisting of less objects rendered.

1.2 Goal Of This Master Thesis

The two design paradigms will be compared and analyzed in this thesis. Comparing the two solutions will give insight in how memory-access impacts performance. In reality, what this thesis really tests for, is how much of a better performance an application gains by having linear memory-layout for its data. A structure like this allows for less

cache-misses, and thus gives a better understanding of memory-bottleneck. In order to assess the differences, several test applications will be constructed for this thesis. First, a custom entity-component-system implementation following data-oriented design principles will be constructed in order to directly test the effect of linear memory-layout. Opengl will be integrated into this architecture and used to simulate graphs consisting of large number of objects. The data-oriented entity-component-system will be compared to the object-oriented counter-part. The second part of this thesis will use the game engine Unity to analyze the two programming paradigms. A simulation of sine-wave will be performed in Unity utilizing the two different paradigms. For the final part, the design principles of data-oriented design will be incorporated into the server of a video-game currently in development by Pineleaf Studio, named DwarfHeim. The goal behind the transformation of the server is to reduce the number of clock cycles used by the server.

The data-oriented design approach for DwarfHeim will not be a complete rewrite of the existing codebase, but rather a conversion. Unity will be used as the game engine for this part as it is already used to develop DwarfHeim. To implement data-oriented principles with Unity, their newly released entity-component-system will be used. For this reason, the goal of this thesis will have two objectives

- Compare data-oriented design against object-oriented design in video-games.
- Devise a strategy for converting from object-oriented implementation to data-oriented implementation with Unity's entity-component-system.

In this thesis, video-games does not strictly mean playable systems. Parts of this thesis will create non-playable simulations that use the same development environment and methods as one would find in video-games.

1.3 Required work for this thesis

Based on the goals from previous section, the work required for this thesis can be summarized with the following tasks:

- Research data-oriented-design principles.
- Research how data-oriented principles can be implemented with the architectural pattern entity-component-system.
- Create a custom entity-component-system application in C# where data is stored linearly in memory.

- Create a simple example in Unity to evaluate the performance boost gained by using data-oriented principles.
- Use the data-oriented design principles for a video-game currently in development.

1.4 Cooperation with Pineleaf Studio

Some of the work done in this thesis will be in cooperation with Pineleaf Studio. Pineleaf studio(3) is a video-game development studio working on their first, yet unreleased, video-game titled DwarfHeim. Access to their source code, which is written in a typical object-oriented pattern, has been given to me in order to test data-oriented design principles. Their technical leader, Fredrik Chrislock, has acted as my supervisor for this part of the thesis.

As a part of our cooperation, I was assigned a desk at their offices in Trondheim. I attended the weekly meetings and were treated as a member of the team. Their large codebase allowed me to test data-oriented design principles in a more realistic setting. By providing me with their code base and helping me with the thesis, I was allowed to conduct research for them and figure out if they should move to data-oriented design. The code assets given were valuable to this thesis. The studio is currently developing their first game, DwarfHeim.

DwarfHeim is an online multiplayer real-time strategy game for pc. The game has a fantasy setting where Dwarves are the main race. Each match in the game is divided into two teams of four players each, where each player controls a subset of dwarves fulfilling a specific role in the game. The roles are divided into four classes, warrior, diplomat, builder and miner. Each role plays differently and the players on a team must collaborate in order to win. The goal of the match is to destroy the other teams base.

The main goal behind our cooperation is to research if data-oriented design is more optimized for their servers. A company like PineLeaf Studio will have to rent servers for their game, the cost of servers are based on number of clock cycles executed on the machine running the server. This means that reducing clock cycles will reduce cost. Furthermore, using data-oriented principles means that the server will potentially respond faster due to less cycles spent on memory access, giving better latency as a result. Since this is a multi-player game, a server is required in order to synchronize the different players on different computers. Each player is a client and communicates with a

Figure 1.2: Warrior unit in Dwarfheim, property of Dwarfheim

server that synchronizes the game state for each player. The server has several functions such as responding to player commands, updating game state for all players and making sure each client is synchronized properly. The game is currently still in its early stages, meaning there's no real server used yet. For testing purposes, a client behaves like a server.

Chapter 2

Literature Study and Theory

Several research topics relevant to this thesis will be covered in this chapter. The different topics will cover the theory behind the foundation for the thesis and the background necessary to understand the methodologies applied. First topic of discussion will be about the memory architecture in modern computers. The second topic will discuss the fundamentals behind data-oriented design. The final topic will cover an architectural pattern that is suitable for data-oriented design, the entity-component-system.

2.1 Memory in modern computers

This section will cover the memory architecture in modern computers. The research conducted here is mainly from two different sources, the "What every programmer should know about memory" article written by LWN and the Unite Austin 2017 keynote by Unity. Additional sources will be cited when necessary.

A typical memory hierarchy found on a computer is shown in figure 2.1. The relation between memory speed and memory capacity is inverse, as the speed increases, the memory capacity decreases. The non-volatile mediums are orders of magnitude slower in exchange for significant larger capacity. For this thesis, the non-volatile mediums are not of interest. The volatile storage mediums are storage entities that need constant power in order to store data. For a modern computer, they can be divided into three types of storage mediums, registers, caches and main memory.
Registers are the memory unit closest to the processor, and as a result, the fastest memory unit in a computer. It is a part of the processor itself and is used to store instructions, operands and results of the operations performed by the processor. A processor do not usually use the registers to store application data during run-time. The registers

are mostly used to perform operation on data fetched from other parts of the memory architecture. Cache and main-memory are two different storage mediums that uses

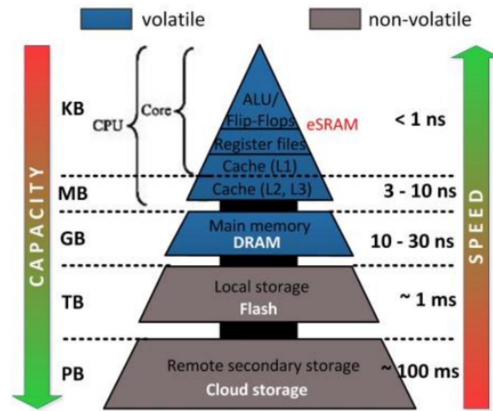


Figure 2.1: Computer Memory Hierarchy (4)

random-access-memory pattern. This means that the access of the data itself can be done in any random order, without the need to be sequential. Essentially, this means that accessing any data is not dependent on the previous memory access. The two mediums differ in the type of ram technology used. Cache's usually uses static random-access-memory(SRAM), while the main-memory uses dynamic random-access memory(DRAM). The DRAM cells have simple structures, only consisting of one capacitor and transistor. The state of each cell is stored in the capacitor, with the transistor guarding access to the state. Whenever a cell is read, the capacitor will discharge, with it eventually being completely discharged. For this reason, each cell must go through a refresh cycle where the capacitor is charged up so it does not loses its state. During the refresh cycle, the access to the state is not available, thus making it slow. The SRAM cells do not suffer this problem as they use six different transistors to store the state. This makes SRAM faster than DRAM, but also more expensive. For this reason, main memory, which is used to store application data and instruction during run-time, is made of DRAM. Some speed is sacrificed for larger capacity and cheaper price. Cache's on the other hand, are smaller in size and made of SRAM-technology in order to have faster access

If a modern computer only used DRAM-technology with the main-memory, the memory access time would be very slow. The processor would spend significant more clock cycles fetching data. On the other hand, if a computer only used SRAM-technology for main-memory, the memory access time would be very fast but significantly more

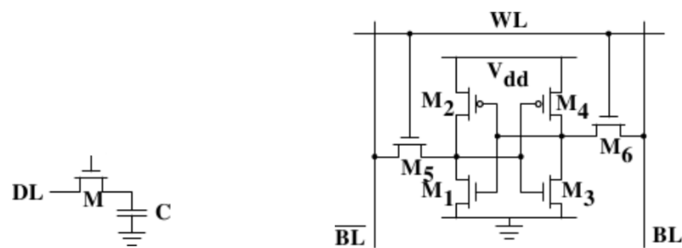


Figure 2.2: DRAM cell to the left, SRAM cell to the right

expensive. For that reason, most modern computers uses a combination of the two technologies to achieve a faster access time without having to increase the total cost too significantly. This is achieved by having caches placed nearer the central processing unit, while the main-memory unit is farther away with larger capacity and slower speed.

Caches are placed near, or is a part of, the central processing unit. More than one cache unit is usually located on the computer. The different caches can be labelled with different levels, depending on how close to the processor they are. A cache of type level 1(L1), is the closest one to the processor. The closer the cache is, the faster access time it has. The higher level caches have slower speed but more capacity. When the processor needs to fetch data through an application, it will first look at the cache units. If the desired data is not available on the first level of cache, it will move onward to the next cache unit on the hierarchy. If none of the cache's have the desired data, it will have a cache miss and move onward to fetch the data from the main-memory unit. As figure 2.1 shows, fetching data from the cache is significant faster. When the processor experiences a cache miss, it will use more clock cycles fetching the data from the main-memory unit.

Whenever the processor needs to fetch data from the main-memory unit, it will fetch a chunk of data including the desired data. The chunk of data, typically 64-bytes on a modern computer, will be stored on the cache units. Data fetched from main-memory will always be done in chunks through the cache-lines. Since the processor fetches a chunk of data from the main-memory that will be stored on the cache, it is desirable that the chunk of data is gonna be referenced soon, as it will already be in cache. Optimizing around the memory access pattern can save clock cycles due to this reason. This brings us to the point of principle of locality, also known as locality of reference. It is a term used to describe the phenomenon in which the same values or related storage

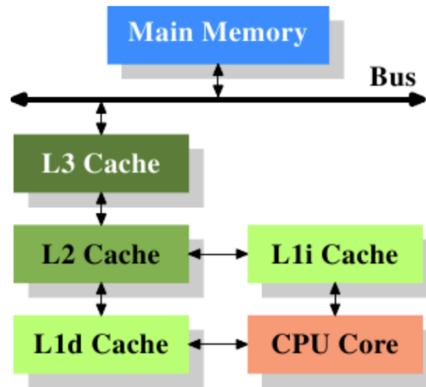


Figure 2.3: The memory hierarchy on the central processing unit

locations, are frequently accessed, depending on a specific memory access pattern (5). There are two types of locality of reference that are of interest to this thesis, temporal and spatial locality. Temporal locality is based on the assumption that if a particular memory location is referenced, then it is likely that it will be referenced again in the near future. For example, if a function uses some specific variable, then it is likely that it will be used again in the near future. This is especially the case for for-loops. In that case, the variable should be saved to reduce cache misses. Spatial locality is based on the assumption that if a particular storage location is referenced at a particular time, then it is likely that the nearby memory locations will be referenced in the near future.

To summarize the research provided above, the following important statements relevant to the thesis can be made:

- The processor will always look at data from the cache first before moving onward to the main-memory.
- Fetching data from main-memory is more expensive.
- The processor will always fetch a chunk of data whenever it reads from the main-memory to the cache.
- Spatial locality can be in our favor if the chunk of data fetched is related to each-other.

In conclusion, the most important aspect of this topic is that the processor is affected by the slower memory units. Performance can be affected depending on how data is stored on the ram. Following the logic learned here, one can conclude that performance can

be potentially improved if the memory-layout is strategically optimized for the way the processor fetches data.

2.2 Data-Oriented Design

In this section, data-oriented principles will be covered. Two topics will be researched for this part, the key elements of data-oriented design and the architectural pattern entity-component-system which utilizes data-oriented principles. In addition, there will be some examples of data-oriented design used for video-games today. Some basic principles behind object-oriented programming will also be covered to give the reader a clear distinction between object-oriented and data-oriented programming. The following resources were used as the basis for this section: Data Locality article by Bob Nystrom (6), Unite Austin 2017 keynote by Unity (7) and "What is Data-Oriented Game Engine Design?" by Davidović(8).

2.2.1 Data-oriented design principles

Data-oriented design is a programming paradigm motivated by the performance gap between the processor and memory. As described in the previous section, memory access time is increasing slowly compared to the processing times in processors. This motivated a programming paradigm that focused around data and cache coherency, with the goal of reducing cache misses. The design paradigm itself became widely used during the PlayStation 3 and Xbox 360 era, where the delays caused by cache misses became to detrimental toward performance. The optimization in this paradigm does not come from advanced algorithms or faster processors, but simply by trying to reduce the number of times the processor must access the slower memory units in a computer. This paradigm is especially efficient for applications where large amount of data must be processed in real-time, as usually found in many types of games. The basic premise behind data-oriented design is simple, program around the data structures. A brief understanding of objected-oriented programming is required in order to further illustrate the motivation behind this paradigm.

2.2.1.1 The pitfalls of object-oriented programming

Objected-oriented design is a programming language model organized around objects. Data in the form of fields and logic in the form of procedures are encapsulated into objects. The programs are designed around these objects. In most programming languages, objects instances are instantiated through classes, which defines the data fields

and procedures. The paradigm has several advantages that have made it popular to use today:

- Inheritance: The concept of data classes, which represents objects, makes it possible to define sub-classes. These are objects that share some or all of the parent class characteristics. This property of oop forces a better analysis of the data models used for the application, reduces development time and ensures more accurate coding by already using well established working models.
- Polymorphism: Objects of different types can be accessed through the same interface, invoking different procedures based on their type.
- Data hiding: A class defines only the data it needs to be concerned with, so when an instance of a class(object) is run, the code will not be able to accidentally access other program data. This characteristic provides greater system security and avoids unintended data corruption.
- Easily distributed: A class definition is reusable not only by the original application, but also by other object-oriented programs as long as they use the same design principles. This makes it easier to distribute among different platforms or use in networks.
- User-defined data types: The concept of data classes allows the programmer to define own data-types that are not already defined by the language itself.
- Higher level of abstraction: Classes can represent real-life components at a higher level, making it easier to model real-life applications. One can solve the problem by modelling around the problem space instead of thinking about low level properties of the hardware.
- Software maintenance: Oop is easier to understand as the objects are modelled around logical entities that are easy to work with. It is therefore easier in theory to test, debug and maintain.

These features makes the paradigm popular and beneficial to use for many applications, especially when working in large teams. It can be used to create different abstraction layers, allowing multiple developer groups to work with their own specific layer. The data encapsulation philosophy allow developers to work with each others modules without having knowledge about the implementation details. All that is required to know is how the data itself is altered, not the method used to achieve it. This is great for maintainability, however not knowing exactly how the data is manipulated can be detrimental for performance.

Unfortunately, there are several issues with objected-oriented programming, that is relevant to this thesis, that affects performance. It is the way dynamic objects are allocated. Whenever the application needs to create new object instances dynamically, it needs to allocate new space for the data. The memory space of an application is mainly divided into two segments, the stack and the heap. The stack will implement data in a last in, first out order. This means that the last item stored on the stack, will be the first item out when data is retrieved from the stack. The ordering of data is done in a linear fashion, where items are added or removed sequentially on the address space. The stack is used to store temporary variables, function arguments and other similar purposes. The heap is a specialized tree-based data structure that is often used for dynamic allocations in applications during run-time. When an application needs to instantiate a new object, it will look for free space in the heap and then allocate it there. The memory allocation in that segment is not sequential. This means that objects instantiated after each other can be placed in complete different addresses on the main-memory. This is against the desire for spatial locality.

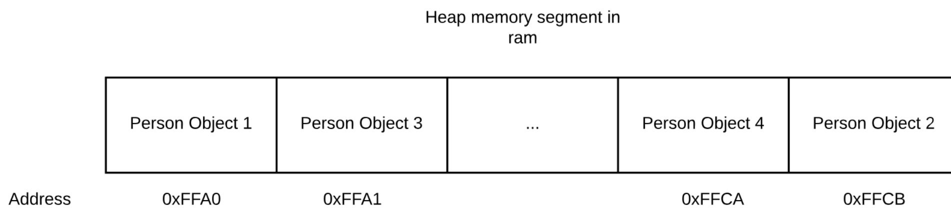


Figure 2.4: Example of 4 objects of arbitrary type person being allocated dynamically on the heap

The second issue arising with the use of objected-oriented programming, is the way object data is stored and the way they are processed in applications. When an object is allocated on the heap, the complete object with all its data will be stored. All the data associated with the object will be fetched from the memory every time the application needs to work with the object data. This will happen regardless if the processor only needs one data field from the object. This will affect performance if a large number of objects with several data fields are needed, with only work being done on some of the fields. This will make the processor fetch the complete object data, meaning that the object will take more space on the cache line and as a result have less space for other data. In addition, if the processor only needs to work on a small subset of the data fields on the object, all the other fetched data will waste space on the cache. This can greatly

affect performance for operations on large number of objects.

The third issue is about parallelization. Synchronization primitives are required for multi-threaded processes for objects, because the state of the data is within the object. Two threads can not operate logic on the same object at the same time without additional overhead, as that could cause race conditions or unsynchronized alteration of data. Each thread must know if some other thread is working on the same object, what type of data it modifies, the side-effects and so on. This makes parallel-programming more difficult for developers, more prone to errors and less efficient use of the processor, all because data is explicitly linked to an "object".

2.2.1.2 How data-oriented programming solves these problems

Given the issues caused by pure object-oriented programming, how does data-oriented programming solve them? As previously stated, the basic premise of data-oriented approach is simple:

> Construct your code around the data structures, and describe functions and methods in terms of what you want to achieve in terms of manipulation of these structures.

Emphasis is put on data, not objects. One does not care about objects that explicitly define a set of data. Instead one cares about the set of data one needs, and allows the "objects" to implicitly be defined by the set of data. Figure 2.5 demonstrates this with an example consisting of data for two persons. In the typical object-oriented manner, the data for two persons would be stored as objects as shown on the figure. The set of data for the person object would be encapsulated into one data type represented through the object. The collection of different data types would be stored together in the memory. For data-oriented design, the set of data associated with a person is not encapsulated together in the same manner. The set of data types could be placed in different sections of the memory and not be referenced through an object reference that collects them into one unit. Some kind of manager would be necessary in that case to link the set of data types that belongs together in the case for the data-oriented design. The benefit from the data-oriented layout is the fact that the collection of data of the different types are separated. This structure allows the same type of data to be efficiently placed linearly in memory. This allows for different processes to work on different type of data sets belonging to the same person, as the data for that particular person is not tightly tied together.

The goal behind this approach is to achieve performance gain by simply making sure

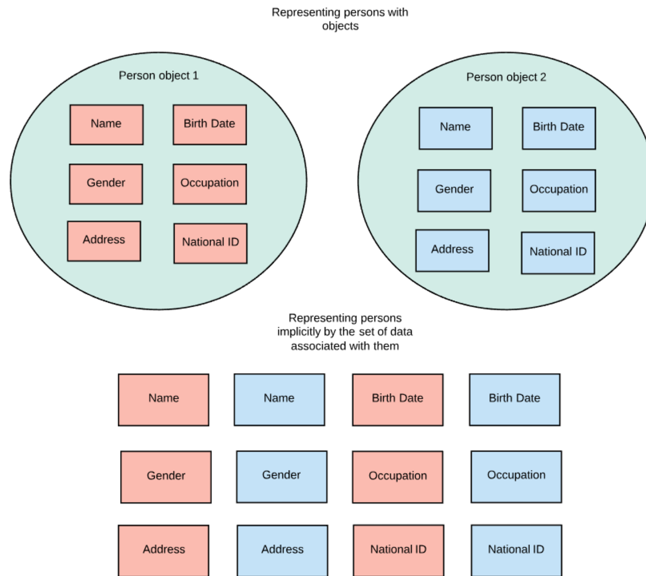


Figure 2.5: Representing data for two persons with objects(top) and implicitly without use of objects(bottom)

the order of data is efficiently used by the processor. It is desirable that the data chunk fetched from main-memory is used sequentially by the processor. For example, imagine a process that prints out the national id of all persons available in a procedure. If the memory layout was object-oriented, then the processor would have to potentially fetch data from non sequential addresses on the memory address. The fetched data through the cache line might not be related to each other, some could be person data and some could be something else. As a result of this ordering, the number of cache misses would increase. In addition, the processor would have to fetch not only the national id of each person object, but also all the additional data associated with it. This would waste space on the cache line and as a result increase cache misses. On the other hand, if the memory followed the pattern as shown in figure 2.5, with the application making sure that data of the same type is tightly packed on ram, the number of cache misses would decrease. All the data on the cache line would be national id's, and after processing the first national id from the cache, the next one could be efficiently fetched from the cache. Not retrieving all type of data would also allow the processor to fetch larger amount of national id's in the chunk each time it fetches through the cache line. This example is visually demonstrated in figure 2.6. Eight national id's are processed for each cache hit.

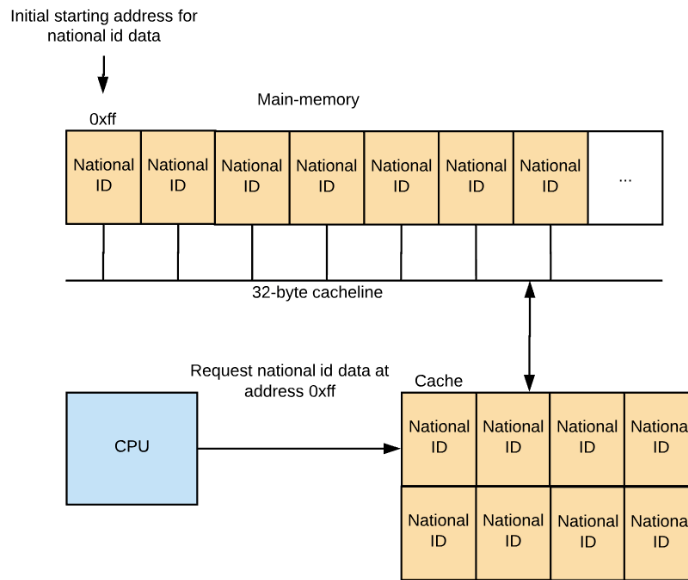


Figure 2.6: The processor requesting national id data

When relevant data is stored contiguously on the memory, the grade of spatial locality is good. The number of cache misses are reduced whenever the processor needs to operate on a set of data of the same type. The processor will fetch a chunk of data containing multiple instances of that specific data type due to its sequential layout. This solution is more efficient for the processor than the heap structure.
Since "objects" do not exist in this scheme, it is easier to separate logic and data. Data is no longer explicitly linked to an object. As a result of this, the application can work on any type of data and make the processor only retrieve the data types required. This gives better principle of locality as the data fetched from main-memory is relevant to each other. This will allow the processor to work on more set of data from the same cache-line. Finally, this structure also makes parallelization easier. Each thread can work on its own specific data type without caring about the others, given that they do not alter each other. This allows for larger scale of parallelization.

2.2.2 Entity Component System

Data-oriented design is all about the ordering of data for efficient memory access. As already stated, by placing related data in a linear fashion, performance can be improved due to less cache misses. There is an architectural pattern that suits well with data-oriented principles, the entity-component-system. Entity-component-systems are mainly

used for games today. The use of it is increasing as it is more efficient for processor intensive games, with Unity recently releasing a new version of their game engine with support of this design pattern (9).

Entity-component-system(ECS) follows the "Composition-over-inheritance" principle, which means that an objects functionality or definition is not given by inheriting the templates of other objects. An object can be defined as having certain methods and functionality by consisting of elements that represent different dataset or functionality. With inheritance, you define your objects with respect to what they are, while with composition, you define your objects with respect to what they can do(10). In practice, this means that you define new types of objects through inclusion of different data types instead of inheriting from a base class. Figure 2.7 tries to illustrate the differences between those two for the same type of object representing a villainous orc in a hypothetical game. With inheritance, the orc class is defined by inheriting from a base class called for "Monster" which represents all kind of objects that can attack and have health. The orc class is extended with the ability to heal and also do more damage through a critical damage attribute. This is what you will usually find in an object-oriented structure. With composition, the orc class is simply defined by including several data types and functions. There is no base class to inherit from, the class itself is defined by its set of attributes and functions.

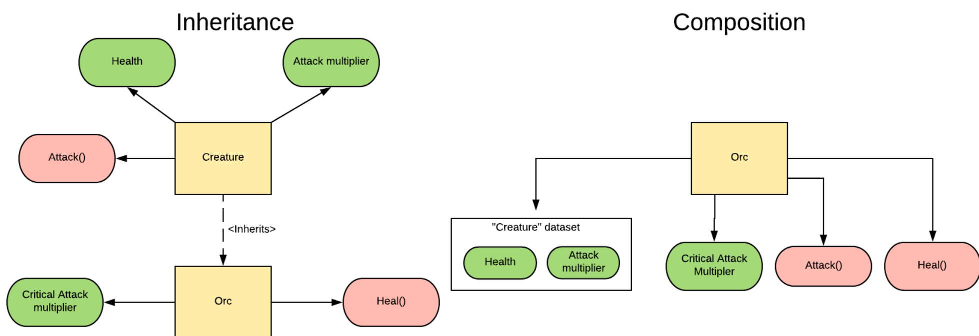


Figure 2.7: Inheritance over Composition vs Composition over Inheritance

There are several benefits of using composition-over-inheritance(c-o-h) such as more flexibility as the objects can implement new features by simply including it in the composition set. The family hierarchy becomes more flat and less complex. This means

that developers don't need to revise the parent classes, child classes and the interoperability between them as much since the objects are no longer dependent on inheriting functionality or data type through each other. The different types in this system will no longer include redundant data or functionality because they only include what they need of features, instead of deriving the whole package from a class(11).

The drawback of using this design pattern is that the types/classes must often implement the methods derived by the functions in the composition set. When using inheritance-over-composition, a child class can inherit several methods from a parent class without the need of implementing their own version of this. For a typical c-o-h architecture, this is not a given behaviour. This means that one must possibly write more code for same functionality as one would do with pure object-oriented way. One could also argue that it is less intuitive to work with composition instead of inheritance, however this is up for discussion.

Entity-component-system uses composition-over-inheritance by separating the application into three different regions, entities, components and systems. Entities are simply an unique identifier that implicitly defines an object. Components are structures that only contains data, they are thus the data-part of this architecture. Systems are the logic-running structure that operates on data from the components. An "object" is implicitly defined in this architecture by having an unique entity id and a collection of different components. These components define the behaviour of an entity by being operated by the systems. The previous orc example in figure 2.7 is again represented in figure 2.8, with the entity-component-system pattern. As the figure demonstrates, the orc object has an unique entity id and a collection of components attached to it. The systems will perform transformation on the data sets it needs, as shown with the dotted lines.

Entity-component-system allows more design flexibility since it follows the composition-over-inheritance principle described earlier. An in-game "object" can easily be extended by adding a new component to its unique entity identifier. New functions and data logic can be implemented by creating a new system that only operates on certain types of components. The data and logic part of this scheme is separated, meaning it is simple in theory to extend each domain without touching the other. ECS can be quite efficient for data-oriented design if the related data is laid out linearly in memory. Some conditions should apply for the ECS design pattern in order to achieve the benefits from data-oriented design.

- All instances of a component type should be laid out sequentially in memory.

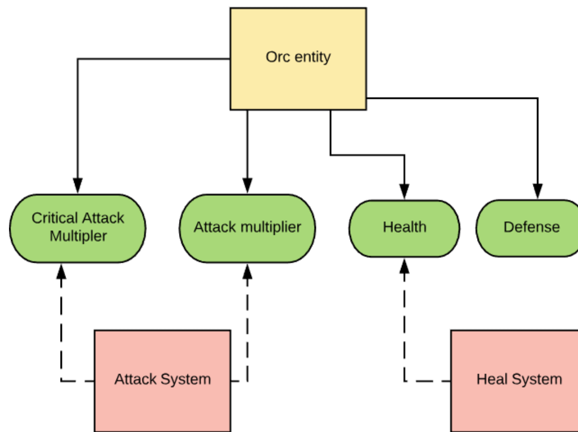


Figure 2.8: An orc "object" represented in the entity-component-system

- Same type of component data used in a system should be laid out linearly in memory so that the system can efficiently iterate through each component instance.
- Accessing component data through entity id should be done through array indexing and not by the use of associative containers .
- Reduced use of branching due to potential branch misprediction.

Chapter 3

Functional Specification and Evaluation Criteria

A foundation for how the two programming paradigms are to be compared will be established in this chapter, based on the research done until now. This chapter will set the basis for the work needed in this thesis. The different topics will cover the type of work required and the specifications along with details on how the solutions can be verified and tested. In total, three different applications will be implemented in this thesis. A custom entity-component-system in C#, a pure data-oriented application in Unity and a conversion of the existing DwarfHeim codebase into a data-oriented solution.

3.1 Evaluation criteria

Each of the solutions proposed will be compared against an object-oriented counterpart. A common evaluation criteria is required for both programming paradigms in order to evaluate the difference in performance. The evaluation will come down to the cpu as this thesis is about cpu efficiency. Some method is required to measure how well the cpu performs in each solution. Given that data-oriented principles are focused around structure of data for better locality of reference, it is important to evaluate the time cpu spends on same type of operation. If data-oriented implementation is indeed better, then it should be able to perform same type of operations with lower amount of cpu clock cycles. For this reason, the frame rate will be chosen as the main evaluation criteria for this thesis. The frame rate is based on time elapsed, which will as a result give an indication of time elapsed for the cpu. Furthermore, frame-rate is an important factor for video-games, as it is also an indicator of visual fidelity.

3.1.1 Frame Rate

Frame rate is used to describe the number of consecutive images appearing in one second. The frame rate is expressed in frames per second(fps), where a frame is an image. Frame rate is an important performance value for video-games. Animations are visualized by having consecutive frames illustrating the same object, but with small modification each frame. This will give the impression of a moving object. A higher frame-rate means a higher number of frames can be outputted to the display each second, giving smoother animation and thus smoother gameplay. The frame rate is also an indication of how fast the processor is. The frame rate can be calculated as:

$$fps = 1/frame_dt$$

Where frame_dt is the time elapsed between two frames. If the processor is fast at processing one single frame, then it will be able to output larger number of frames each second. This formula does not always give an accurate representation of the real frame rate. Sporadic processing events can increase the time elapsed between two frames in different situations, giving different values to the frame rate. For this reason, average frame rate should be used. This can be done by storing several frame rate values and then calculate the average based on the values stored.

The frame rate will be used to compare the different programming paradigms. To get an accurate representation based on frame rate, it is important that the implementations done in each programming paradigm is as similar as possible when it comes to features and the task they are to perform. The only difference in implementation should only be the fact that one is object-oriented and the other data-oriented. Other factors must remain equal to get an accurate representation.

3.1.2 Cpu usage time

The frame rate gives an overall representation for performance. It is not necessary an accurate representation on how the data-oriented design performs versus the object-oriented solution. There might be other factors in the tests that are not related to the programming paradigm that might affect performance negatively. These could be for instance operations related to calculation of physics. For this reason, an extensive inspection into the cpu usage times will be done for a better evaluation of the different solutions. This is only applicable for the solutions done in Unity as it has support for a diagnostic tool, the Unity profiler. This profiler gives a comprehensive look into cpu usage time for the different operations each frame within a process.

3.2 Entity-component-system in C#

To test the usability and efficiency of using the architectural pattern entity-component-system with data-oriented principles, a custom implementation will be written in C#. Using Unity's entity-component-system will not require any knowledge about the internal structure. For this reason, creating a custom implementation in C# will allow for better understanding of the pattern and its advantages and disadvantages. Another purpose behind this is to have full autonomy behind the pattern, allowing the use of data-oriented principles to test the efficiency of data-oriented solution against a traditional object-oriented implementation. This will allow for more control on how the memory layout is for the different data types.

3.2.1 Specifications

The design will have a list of specifications that sets the basis. The requirements must be fulfilled in order to correctly verify intended behavior and have a reference point for comparison against a traditional object-oriented solution. Table 3.1 outlines the specifications for the design along with description of acceptable criteria for each requirement.

The specifications will set the foundation for the implementation described on later sections.

3.2.2 Evaluating against object-oriented programming

The design will have to be compared against a traditional object-oriented application. To evaluate the efficiency of an entity-component-system which follows several of the data-oriented principles, a test application will be devised. The application must be implemented twice, once with the entity-component-system, and once in a traditional object-oriented way. A set of requirements are needed for the test application in order to accurately analyze the differences in the programming paradigms when it comes to memory efficiency. The specifications are listed in table 3.2 and will be the basis for the implementation. The main objective behind the test application is to compare iteration times for the two programming paradigms. The test application should thus perform cpu work that requires iteration through component data.

Specification	Acceptable criteria
The design must follow the architectural pattern of the entity-component-system.	The software follows the entity, component and system definition.
There should be a strict separation between the representation of data(through components) and the behaviour on these data sets(system).	Systems define the intended logic for an application while components store data.
Component data of same type should be addressed linearly in contiguous memory.	Component data of one type is stored in arrays.
The three different domains should be decoupled and only accessed through a manager.	The manager is the only class that directly interacts with all the different domains.
Entities should be unique	Each entity can have an unique id assigned by the manager.
Entities should have a weak reference to its components and not direct.	The manager will access component data through an entity id.
It should be easy to define new behaviour and functionality by creating a new system.	New systems can be created by inheriting a base system class that defines the system interaction in the design.
The solution must be able to render graphics.	The design is able to send draw calls to the graphics processing unit.
The solution must be able to be comparable against an object-oriented solution	An application that could be made object-oriented, must be able to be implemented with the entity-component-system.

Table 3.1: Specification for the custom ECS implementation

3.3 Entity-component-system in Unity - Pure data-oriented solution

The second part of this thesis will consist of an application completely implemented in Unity with their entity-component-system. This solution will have a set of requirements given in table 3.3 that it must fulfill in order to properly assess the efficiency of data-oriented principles.

3.3.1 Evaluation of the application

This application will be written twice with data-oriented and object-oriented principles respectively. The two solutions will be compared against each other using diagnostic

Specification	Acceptable criteria
The test application must perform significant cpu work.	The application will continuously do work on a large number of objects.
The test application must have visual output	Draw calls can be sent to the graphics processing unit for visuals.
The test application must consist of a large number of objects.	Instantiate and create a significant number of objects/entities for the application during the initialization phase.
The test application must continuously retrieve data from the memory units.	References to variables and types storing data must be continuously referenced.
The test application must iterate through large number of entities/objects.	The application must perform same type of work on each individual object.
The test application must perform the same type of work for both implementations.	The function of the application must remain the same for both implementations. The only difference is the underlying architecture.

Table 3.2: Specifications for the test application written for the ECS design and the object-oriented counter-part

Specification	Acceptable criteria
The application must use Unity's data-oriented entity-component-system.	This is given as long as the entity-component-system feature in Unity is used.
The application must perform significant cpu work.	The application will continuously do work on a large number of objects.
The application must be able to measure frame rate.	Time elapsed between each frame must be measured in order to calculate the frame rate.
The application must continuously and often access data from the memory units	References to variables and types storing data must be continuously referenced.

Table 3.3: Specifications for the pure data-oriented solution in Unity

tools provided by Unity, along with the frame rate. The two implementations of the application must be very similar in design with exception to the underlying programming paradigm used. The specifications set for this part will allow for comparison between the two paradigms through significant cpu work for the application.

Frame rate will be an indicator on performance for both solutions. If a higher frame rate is acquired with one of the designs, then it will imply that it is better at using the

cpu than the counter-part. The profiler in Unity will be further used to assess the differences in performance in order to get a more accurate representation of the values collected.

3.4 Converting DwarfHeim into a data-oriented solution

Converting the existing object-oriented DwarfHeim codebase into a data-oriented application will be the final part for this thesis. This part is mainly aimed at the client-server interaction part of the code, and not the complete codebase itself. Data-oriented principles will be achieved for this design by the use of the entity-component-system feature in Unity, which itself follows data-oriented principles. There are two possible alternatives for moving forward with the DwarfHeim game:

- Re-write the complete codebase from start using the entity-component-system in Unity.
- Convert the current codebase into the entity-component-system.

The first option would not be limited by the current object-oriented codebase. A complete refactoring from the beginning would mean that the game could be made into a pure data-oriented game. However, the amount of work required for this is too significant at its current state, especially for this thesis. Not only is the code related to the game itself object-oriented, but also most of the 3rd party assets that it uses. The amount of work required is substantial and not feasible for this thesis. The second option would involve a conversion of the current design, by slowly transitioning the object-oriented parts into the entity-component-system. By converting small parts of the game at a time, one could strategically convert the complete game. However, this alternative is still affected by the fact that parts of the codebase is very object-oriented, and as a result difficult to truly convert into a data-oriented game without investing substantial time. A strategy for this conversion will be devised in section 4.6.2.3.

3.4.1 Specifications for conversion

It is important that the conversion strategy for DwarfHeim does not alter the game too significantly when it comes to features of the game. The conversion aims to make the game more data-oriented and as a result more efficient. The design is going to be restricted due to the objected-oriented structure, which means that the specifications set can not be to restrictive. The basis for this conversion will follow the specifications given in table 3.4.

Specification	Acceptable criteria
The converted parts must conform under data-oriented principles.	This is given as long as the entity-component-system feature in Unity is used for the converted parts.
There should be clear separation between data and logic.	This is achieved by using components and systems in the entity-component-system
The application must be able to measure frame rate.	Time elapsed between each frame must be measured in order to calculate the frame rate.
The conversion must remain true to the games features	The converted areas of the codebase must keep the same type of features as the one found in the original.

Table 3.4: Specifications for the DwarfHeim conversion

3.4.2 Evaluation of the conversion

The conversion will be applied to the existing codebase if time permits. After the implementation is done, the new design must be compared to the old one. The changes must be analyzed and compared to the previous version in order to verify whether the design is more efficient or not. The standard evaluation criteria will be used for this part, the frame rate and cpu usage time. Furthermore, since the main objective behind the conversion is the server-client interaction, it will be the main area of interest for the evaluation.

If the new design has a higher frame rate and better cpu usage times, then it will imply that the design is indeed better when it comes to cpu efficiency. The cpu usage time will give further information about whether the improvements are due to data-oriented principles or other unrelated changes not directly applicable to data-oriented principles. This will have to be analyzed in order to get an accurate representation of the results.

Chapter 4

Materials and Methods

Materials and methods used for this thesis will be described in this chapter.

4.1 Development Environment

A brief description of the development environment will be presented here. This section covers the programming language used for this thesis and the integrated development environments used.

4.1.1 Game Engine - Unity

Several functions are required in order to develop a complex game with graphics. Some of the more important features are the following:

- Implementation of physics in order to handle physical alterations such as transformation and rotation of objects in the game.
- Input from the user who plays the game.
- Graphical rendering of objects in the game.
- Scripting functionality allowing developers to implement new features and game logic in an efficient manner.
- Collision detection.

It would be possible to implement all these features on your own, however that would require substantial amount of work. Fortunately, there are several game engines available on the market which provides with all these features. Some of the more popular

game engines available today are Unreal Engine, Unity and GameMaker. They all provide important features such as a physics system, giving you an easier entry into video-game development. Since Dwarfheim is already written in Unity, I decided to use Unity from the start for this thesis.

Unity is a multi-platform game engine developed by Unity Technologies(12). Rich with features, it makes it easier to jump into video-game development. The engine is customized for both 2d and 3d-game development, allowing a large collection of possible game genres. There are several important features provided by Unity that comes in handy for this thesis such as:

- An extensive physics engine that provides convincing physical behaviour.
- Game-behaviour scripts that can be written in C# or javascript.
- Developer friendly interface for connecting game objects to scripts.
- Mesh renderer for rendering of meshes and other visual entities.

In addition, Unity recently released their newest version, $Unity$2018.1 with support for data-oriented design. The new update introduces entity-component-system to the engine along with easier parallelization through their new job system. These features will allow for easier transition into data-oriented design.

4.1.2 Different terminologies and concepts in Unity

Some of the more basic concepts of Unity will be briefly described, as they are related to the work done for this thesis. Only the concepts relevant in a significant way will be described here. These concepts are necessary to know in order to understand future references in this thesis. Additional information are supplemented in the appendix B.1 for other concepts that are of less importance. The concepts described here are all related to the Unity game engine and its interface.

4.1.2.1 Game objects

Game objects in Unity are all objects with some property within the game engine. All entities that affect the environment, gameplay or physics of the engine are considered game objects. In practice, this means that all objects in Unity are game objects. Entities such as the camera, light sources, visual effects and game characters are all considered game objects, even though they do vastly different things. Understanding the concept behind game objects is important since almost everything done in the engine is done

through game objects. The entities created with the entity-component-system is the exception. All references to game objects in this thesis means any object that affect the game in some way, in an object-oriented way.

4.1.2.2 Components

Components are scripted objects attached to game objects. Components define behaviour that game objects will adhere to as long as they own that component. Components must not necessarily contain behaviour logic, and can instead only contain data. Unity allows you to define your own components by writing the scripts in either C# or javascript. When the behaviour or data type is defined in the component script, it can be attached to all the game objects that it is relevant to. Components will be further described in section 4.1.3.

4.1.2.3 Transforms

Transform is a component that every game object must include as part of their component set. The transform component is used to store a gameobjects position, rotation and scale. It is not possible to create a game object without a transform component attached to it. The position and rotation part of the transform component defines its position and orientation in the game world, while the scale part defines the scale of the size relative to some default values. The transform component is a vital part of the Unity engine as this is the component that is used to update a game objects position or orientation. In addition, the physics engine within Unity uses these values to calculate physical alterations and other behaviour such as physical collision. Unity provides with a scripting api that allows for changes to the transform component, the transform api. For example, the position of a game object can be changed by simply retrieving the current position and updating it with new value through the transform api. The code snippet below shows an example of this.

Listing 4.1: Example of code updating the position of a game object with dx in x direction, dy in y direction and dz in z direction

```
1 public void UpdatePosition(float dx, float dy, float dz)
2     {
3         // Access the current position of the relevant gameobject
4         Vector3 newPosition = this.transform.position;
5         newPosition.x += dx;
6         newPosition.y += dy;
7         newPosition.z += dz;
8         // Update the current position with the new position
9         this.transform.position = oldPosition;
```

```
10        }
```

4.1.3 Scripting in Unity - Adding behaviour to game objects

In order to create a game, one needs objects that perform some sort of behaviour through scripts. The type of behaviour can vary in a game, for example, objects can respond to input from the player or an event that triggers physical collision. An essential part of a video-game is to have objects that are scriptable. Fortunately, Unity offers an easy interface for adding scriptable behaviour through their component system. Components in Unity are scripts that are written in either C# or javascript, deriving from the monobehaviour class. The monobehaviour class is the base class from which every Unity script derives from , giving access to several event functions (13). Once a C# class derives from the monobehaviour class, it becomes available as a component. The component class can then be added to a gameobject in order to give it the behaviour defined in the script. A gameobject can consist of several components, where each component can perform different type of behaviour.

The monobehaviour class gives access to event functions such as Update(), FixedUpdate(), Start() and OnEnable(). The Update() method on a Unity script will be called every frame, executing the logic defined inside. The time between each frame can vary and thus give different times elapsed between each Update() call. If it is desirable to update at fixed intervals, one can use the FixedUpdate() call which will be called at a fixed frame rate. Furthermore, the Start() method will be triggered once the component is enabled and ready to run, before any Update() calls. The OnEnable() method is another event function that is activated once the the script is enabled. All these functions are available in a script through the monobehaviour class, allowing developers to easily add behaviour based on different events. Moving an object every frame can easily be done by adding the logic to a script on the Update() function, which will then update movement each frame. All these event methods have different order of execution and different condition for triggering it. The Update() method will run each frame, while OnEnable() method will only be triggered once a gameobject has the component enabled.

A simple example demonstrating how the scripting system works in Unity is included in the appendix B.2.

4.1.4 Analyzing performance - Unity Profiler

Unity has its own profiling tool for analyzing performance. The tool records performance data for different point of interests. Performance criteria such as frames per second, time spent on rendering and time spent each frame is available through this tool. The profiler will be used to compare performance between the two different programming paradigms. The most important type of data from this tool will be frames per second and cpu usage.

The cpu usage in the profiler displays time information about the different operations during a single frame. It can be used to inspect the time elapsed for one single operation, such as a script update function. The data here can be used to inspect spikes and bottlenecks. For this thesis, the data provided here will be used to find anomalies in the results and to verify that the improvement in performance is actually due to data-oriented principles and not other unrelated changes. The profiler also provides with the frame rate in the cpu usage profiler, however the value is not an actual representation of the overall frame rate for one single frame. The values are given for the different operation groups in the engine, such as the frame rate for rendering, physics and scripts. A script calculating the actual frame rate is required to get an accurate representation.

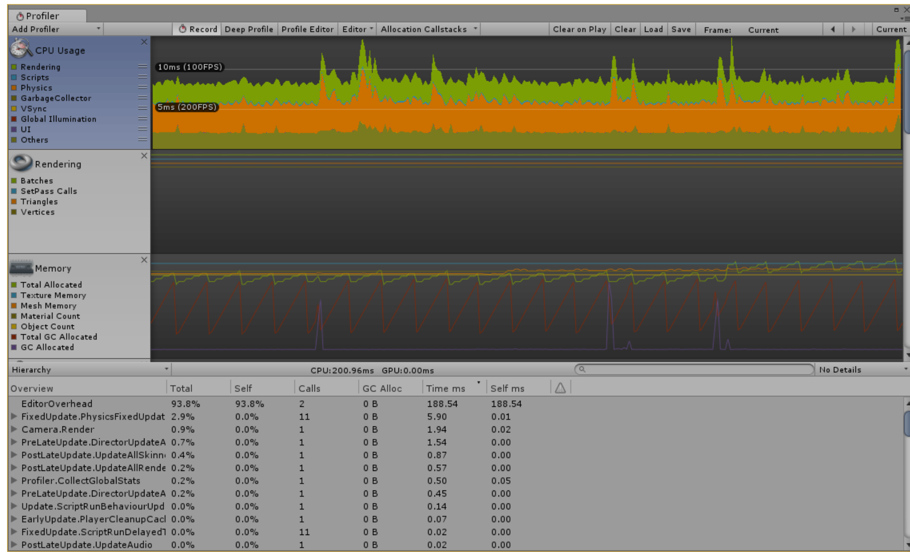


Figure 4.1: Unity Profiler

A third-party script, profiler data exporter(14), will be used to gather useful stats on the data provided in the profiler for cpu usage. The script will be used to calculate min-

imum, average and maximum values for the data recorded on the last 300 frames, as 300 frames are the limit of the profiler data stream.

4.1.5 Programming Language - C#

The C# programming language developed by Microsoft will be used as the main language for this thesis. C# is a part of the software framework developed by Microsoft, the .NET framework, for the windows operating system. The language itself supports several programming paradigms, with emphasis on object-oriented structure. It follows many of the same features as C++, and follows same philosophy as Java with its own just-in-time compiler(15).

There are several reasons for choosing this language instead of a more optimized language such as C++ First, C# is an integrated part of Unity and thus required for this thesis anyway. Second, there are several important features of C# not found in C++ which makes it easier to develop the entity-component-system with. The most important features are reflection and custom attributes, which used in conjunction allows for dynamic creation and processing of data with the simple use of attribute fields. Using these allows for creation of an entity-component-system that can fill data linearly in arrays before a system executes with the use of attribute fields. These things could still be achieved with C++, however it would require more work as the users must spend more time on managing the memory layout correctly. A detailed description of these features and some more are described in the appendix B.6. Finally, even though C++ might be a more optimized language due to it being a compiled language, it is not necessarily more efficient for this thesis as there's little use of direct memory management. Unity takes care of most of the memory management, the user just needs to tell the engine that it will require it. For the custom entity-component-system, the ordering of memory will be done indirectly through the use of arrays and value types. These do not require the autonomy that the C++ language provides anyway, and might thus not really affect the efficiency of the solution.

4.1.6 Programming Language - Python

Python will be used to plot the results from the tests on graphs. Several python scripts will be written for this thesis. The scripts will read in output files in text format, and parse the data points available. Depending on the type of test and format, the scripts will plot graphs of different kind visualizing the results. The python libraries matplotlib and numpy will be used to plot the graphs and figures of interest.

Before the script can plot the graphs, it must read in the data set for x- and y-values on the graph. The results outputted in the text files from the tests must be written in a specific format. For this thesis, all data points and other variables required for the python script is gonna be written with the following format:

$$TypeOfData : Data$$

TypeOfData specifies the kind of data the given data is. This value can be used for specifying type of data on the graph, and the parameters for the graph itself, such as the title of the graph. Figure 4.2 demonstrates an example utilizing this format with the graph plotted. The parsing part and type of data read in will not be the same for the different tests, however they will all follow this format.

testplot - Notepad

File Edit Format View Help

```
Test plot for testing correct behaviour with matplotlib.
title:Drawing curves with matplotlib
x:[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
y:[0.0, 1.0, 4.0, 9.0, 16., 25., 36., 49., 64., 81.]
xlabel:Time spent on matplotlib
ylabel:Learning matplotlibn
```

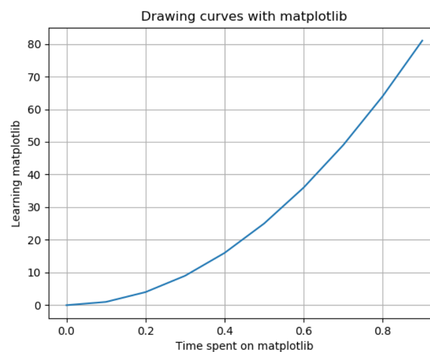


Figure 4.2: Example of output file and the resulting graph plotted

4.2 Measuring the frame rate

The frame rate for the different applications will be measured by recording the time elapsed between the latest frame and the previous one. The implementation for this differs based on whether the solution is written with Unity or Microsoft visual studio.

4.2.1 Measurement of frame rate for Unity

A frame rate measuring script was written in Unity following the tutorial by Catlikecoding(16). Minor modifications were done to the scripts in the tutorial in order to make it more applicable for this thesis, such as writing results to a text file. In total three script files are created, one to display the frame rate on the screen, one for calculating the frame rate and one for setting color and labels to the value shown on the screen. The frame rate data shown on the screen will be based on three different values, the lowest frame rate, the highest frame rate and the average frame rate. These values are based on the 60 last frames measured. The frame rate is calculated by following the formula shown in 3.1.1, where 1 is divided by the time elapsed between the last and current frame.

Figure 4.3: FPS counter for Unity showing max, average and lowest fps based on the 60 last measurements

4.2.2 Frame rate counter outside Unity

There are several tests written in this thesis with Microsoft visual studio. In these cases, the frame rate will be calculated by using the standard stopwatch library in C#, which can be used to measure time between frames. This value can be used to calculate the frame rate. The frame rate values will be written to a text file in scenarios where the frame rate is needed for analysis.

4.2.3 Refresh rate

The frame rate of a display is bounded by its refresh rate. The refresh rate of a monitor is the maximum number of frames it can render in one second. The standard refresh rate for monitors are around 60 hz. The monitor used for this thesis has a refresh rate of 60 hz. For this reason, the max fps will be 60 for some of the tests. It is possible to output a higher fps than what the monitor is capable of displaying. Video cards usually have vertical synchronization, which prevents the video card from changing the display memory until the monitor is finished with its current refresh cycle. In practice, this caps the frame rate to the refresh rate. It is possible to turn this setting off, allowing a fps value higher than the monitor. This will still not increase the actual frame rate displayed by the monitor, but it will allow the processor and gpu to output larger number of frames each second. The vertical synchronization option will be turned off if deemed necessary during testing.

4.3 Entity Component System - Custom Implementation

The design of the custom entity-component-system in C# went through several iterations before being finalized. The initial and final design will be described in this section. The entity-component-system will hereby be abbreviated as ECS.

4.3.1 Initial entity-component-system architecture details

An initial design was constructed with no regards to optimization or speed in order to get a good overview of the overall architecture. This implementation was inspired by a library written in C++ by Sébastien Rombauts(17). The application can be divided into eight different parts, as shown in figure 4.4. A description of each module will be given before proceeding with further optimization into the final design. A more detailed description is given in the appendix B.3

4.3.1.1 Entity Data Type

Entity is an unsigned integer that represents an unique entity. The identifier itself is implemented as a struct with implicit conversion from unsigned integer type. This was necessary as C# does not support the typedef specifier found in C/C++. It is merely a data type representing an aggregation of components through weak reference. This means that every "object" in the game is addressable through an unique entity id, and its components can be found indirectly through this entity. In order for this to work as a weak reference, there must be some kind of data structure that keeps track of which

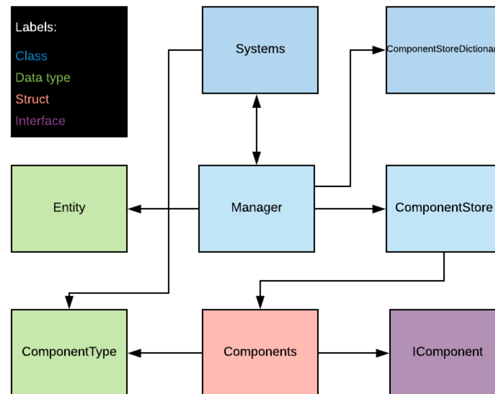


Figure 4.4: The overal architecture of the first implementation for custom ECS

entity belongs to which component instance. This is achieved through the component store module.

4.3.1.2 Component Type

Down at implementation level, the Component type data type is similar to the entity data type. It represents the different component types available in the application. Each component must have an unique component type id assigned to it. This data type uniquely represents the different component structs in the system.

4.3.1.3 Components

Components are user-defined structs that also implements the IComponent interface. These structs are defined by the developer. Component structs are used to represent data for the different entities. They are separate from the entity module and is only weakly referenced through the component store module. Structs were used instead of classes because they are of value type in C#, meaning that the whole struct data-set is stored in the stack and not the heap.

4.3.1.4 IComponent

Every new component created by the developer must be assigned a component type id. IComponent is a C# interface that forces new component structs to include the component type property. It is important for every component to have a component type id in order for the application to work, as reflection is used to access the component data through this interface.

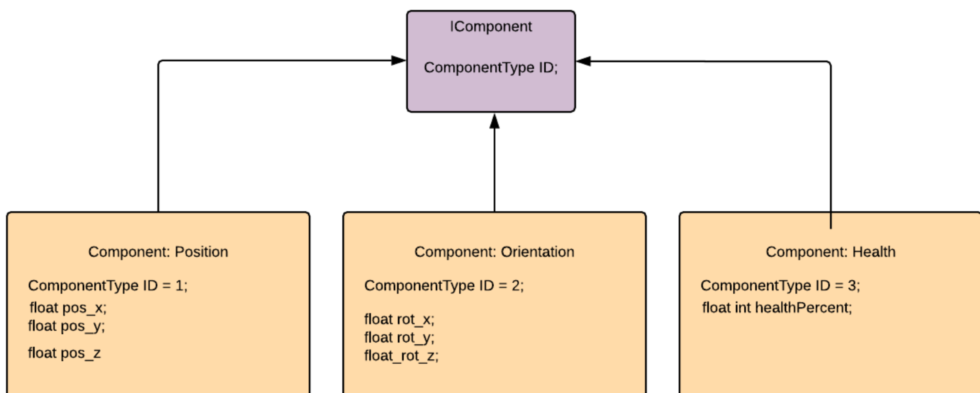


Figure 4.5: Example of components

4.3.1.5 Systems

Systems represent the system part of ECS. All systems in the application derive from an abstract system class LSystem. This base class contains some fields that are necessary for all derived systems. It also includes a virtual function that every derived class must implement, the update function that runs once every loop on the main thread. Behaviour of a system can be defined through this virtual update function. LSystem was chosen as the base class name in order to avoid confusion with the system namespace in .NET. Every derived system class has two containers that are necessary, a list of all entities that are legal and a set of all components that the system operates on. An entity is defined as legal if it has the required components. The system classes access component data of an entity through a manager which invokes calls through reflection. The data is not retrieved in a linear fashion at this state, breaking one of the data-oriented design principles. An example demonstrating the use of this class is shown in the appendix B.3.1.

4.3.1.6 Component Store Class

It is desirable to have multiple entities with different type of components attached to it. Since entities have their own component instance of a component type, a method for mapping a component instance to the correct entity is required. This functionality is achieved through the component store class. This class is defined as a generic class, which means it can operate with different type of components. A new component store object is created for each component available on in the application. Internally, this class has an associative container which maps entities to their respective components. Figure 4.6 shows an example of two different component stores. Entity 1 and entity 2

has only one type of component each, while entity 3 has both components. The manager class will contain a list of these component stores which it then uses for accessing the correct component for an entity.

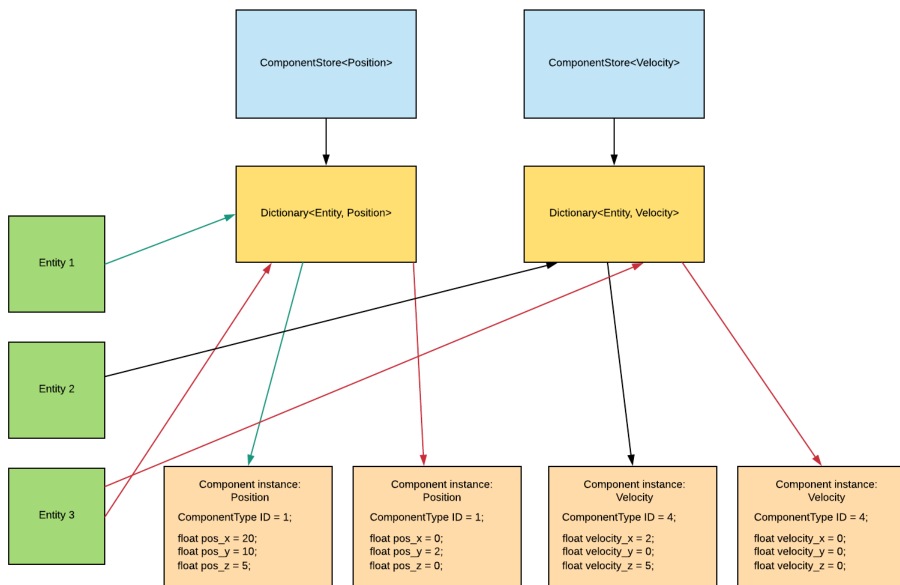


Figure 4.6: Component store for two different components

4.3.1.7 Manager Class

An entity-component-system architecture divides the application into three different separated modules. For an application to run with this decoupled design, a manager is required to connect entities to their components and the systems to the correct components. This is the purpose of the manager class. The manager class has a list of all entities, component types and systems of the application stored internally. The manager creates new entities and assigns them a new id, it couples entities with their respective component instance and runs the virtual system function every frame. A list of component stores are also a part of this manager, and it is the managers sole responsibility to handle these structures. The manager will run a loop that continuously iterates through each system and activates its update function. A complete iteration of the loop is defined as one frame.

4.3.1.8 Component Store Dictionary Class

The manager class needs a way to map component types to their respective component stores. This is achieved through the component store dictionary class which uses C# dictionary. This is an associative container that uses the component type value as the key value. A reference to the component store object is then given by the key value.

4.3.2 Improved design

As mentioned in the previous section, the way a system retrieves data is not efficient, nor does it follow the specifications given in 3.2.1. It doesn't really follow the principle behind data-oriented design, where data layout is important. It is desirable to have component data linearly stored in memory for a system once it iterates through it. For this reason, the top level architecture was modified in order to achieve this functionality. The application structure needs some minor modifications. The improved design is shown in figure 4.7. The goal of this revision was to have memory stored linearly when accessed by the different systems in order to improve performance through spatial locality. The improved design involves extensive use of C# reflections and custom attributes, partly inspired by Unity's entity-component-system.

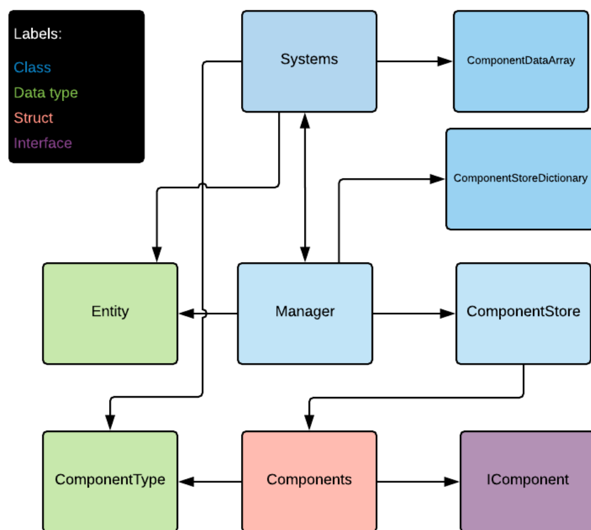


Figure 4.7: Improved ECS top level architecture

Component data array is a new generic class that is very similar to the standard container List in C#. The purpose of this class is to store component data linearly in memory. An internal array is used as a container for the component data. The systems

must declare component data array objects for each type of component type it wants to operate on.
Once the different component data array objects are declared in a system, they will be filled with component data. The component data array object can be modified through reflection in conjunction with custom attributes, allowing the application to fill it with component data for a system before update function is called.

A custom attribute was created for the application, named Inject. The name was inspired by the same type of custom attribute found in Unity. The inject attribute is only associated with component data array objects. Once a component data array of component type T is associated with the Inject attribute, it will fill its arrays with data of type T during run time before the system executes its update function. An example demonstrating the inject attribute in conjunction with the component data array type is shown in the appendix B.3.2.

The improved design will bring some new challenges to the implementation. One of the issues is the fact that the new design has two storage containers for component data, the component store and the component data arrays. The component store contains data for the different types as a part of the manager class. In addition, the system class now holds its own component data array fields for storage of component data relevant to it. This means that the application needs to synchronize between the two containers for each component type. The component stores must update the component values every time a system performs data manipulation. The component data array fields must include new component data every time a new entity has been registered.

These changes makes memory layout more linear, however it also brings more overhead which might reduce performance. Giving each system its own set of component data array fields will also mean that more memory is required by the application. An example of how the manager transfers memory between different storage containers are shown in figure 4.8. The two systems operate on the same component data, with system 1 being the first to execute. After the steps are finished, system 2 will perform the same steps.

By using component data arrays objects with internal array storing data linearly, the system is able to retrieve data from contiguous memory space during its update function. This will in theory give better performance in exchange for larger files. Even though the improved design might improve performance, it is still not close to an object-oriented application when it comes to speed. The naive implementation was written

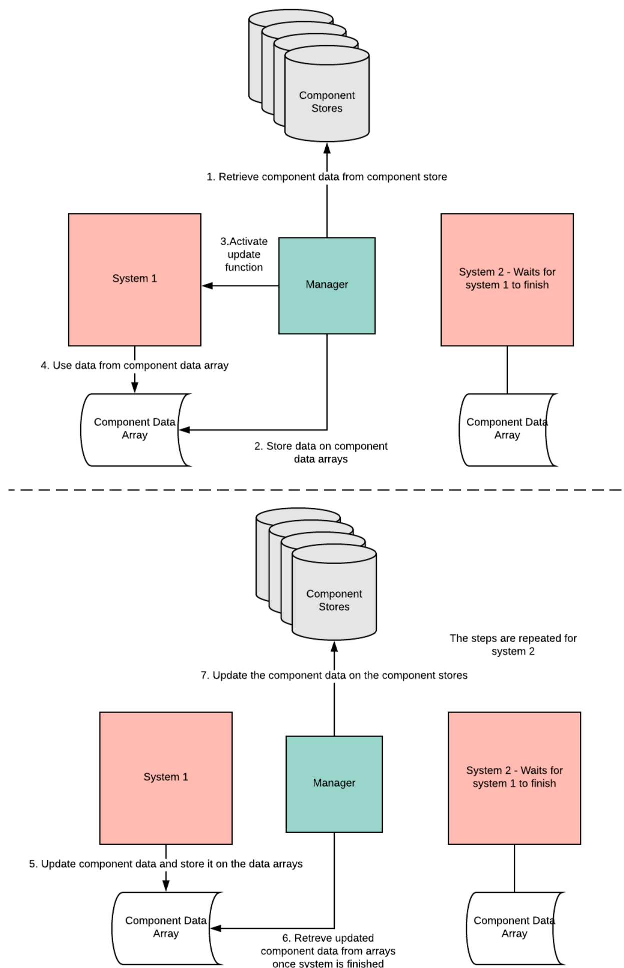


Figure 4.8: Order of operations performed by the manager for a system

first to create a functional ECS design, without putting emphasis on speed. For this reason, the improved version was further optimized in order to make it usable for realistic applications. The changes for this part aren't directly related to the architecture, but rather optimization steps relevant to the C# language. A brief list of the optimization done is given her, with more detailed description in the appendix B.3.3.

- Reduced number of boxing and unboxing
- Reduced number of function calls through another class during reflection
- Component stores will only be updated if changes have been made to the com-

ponent data array in a system

The final design fulfills all the requirements set in section 3.2.1

4.3.3 Functional testing of the ECS implementation

Several functional tests were performed for the implementation in order to verify correct behaviour. The different tests tested the following functions:

1. Creation and deletion of entities
2. Each entity is created with an unique id
3. Entities are assigned component data
4. System registering a component type as part of its list of required components
5. Entity and all of its component data is deleted
6. Systems performing logic on the list of components it has been given
7. Modified component data is updated to the component stores
8. Retrieving component data using the entity manager
9. Systems run in a specific order
10. Allow multiple systems to run
11. Verify synchronization between different systems operating on the same subset of data.

These functions define many of the properties found with the entity-component-system pattern. A verification of the intended behaviour verifies that the design follows the pattern, fulfilling one of the functional specifications given in 3.1. The tests were always performed with the same input values and same expected output values. Assert functions were used in the tests to verify that the functions had correct output when changes were made to the design.

Three tests were created in order to verify all the behaviour listed. The first test verified the behavior for 1-5 in the list above. Twenty entities were created with a dummy component attached to it. Then a system registered that dummy component as a required component for that system. This means that all the entities should be legal for the system in the test. The test will proceed to print all legal entities for the system, delete some entities then re-print with the updated set of legal entities. An assert method is

invoked after the creation and deletion of entities in order to verify that the design have correct behaviour. If the values differ from the expected values, an error exception will be thrown. The output of the results are written to a text file.

The second test confirms the behaviour for 6-8 in the list above. The test has a test component which consists of an integer and float variable, registered to a system. Several entities will then be created with an instance of the test component attached to it. The system will increment the int value of each component with one, while increasing the float value with 20. The values of the test component will be updated afterwards to the component stores. Once again, the assert method is used to verify that the values after system update is equal to the expected values. The output of the results are written to a text file.

The final functional test tested the points 9-11 listed on the list above. The test consists of two systems operating on a set of component types each. The first system will operate on two components with some data fields, while the second one will only operate on one of these components. This test was created to confirm that the systems correctly update values so that the next system can use the updated values. It also tested the possibility of having multiple systems working together. The results were outputted to a text file.
The results of these functional tests are shown in section 5.1.1

4.3.4 Performance tests for the ECS implementation

The ECS design went through several revisions before being finalized. Improving performance and conforming to the specifications given was the aim behind the changes applied. The main performance parameter of interest was the iteration time for component data. A test was written to verify that the iteration time were improved upon the design modifications. The test will be performed on two versions of the implementation, the initial implementation and the final revision. The test is not intended for the object-oriented solution, as it only measures the performance improvements for the entity-component-system design. The reason for choosing these two points are because they differ fundamentally in how they access data. All the other revisions are just intermediate steps in reaching the final version. The first version of the design retrieved component data through calls to the component stores which contained the data. The data was not laid out linearly for fast access. The final version injects the system with component data stored in a linear fashion before update function is called. Allowing the manager to fill the component data arrays before a system update introduces more

overhead, which could have given worse performance. This test will essentially confirm whether the cpu is able to perform better when data is linearly stored in memory through arrays.

The test itself is simple. A large amount of entities are created, with each entity having the same type of component attached to it. The component consists of a three-dimensional vector variable that holds a hypothetical "position". The test will run through all the entities and its component data on a system that registers this position component. The position component of each entity will be updated, by simply being incremented. This operation will happen 100 times for each entity in this test. Figure 4.9 illustrates the structure of the performance test. The time elapsed will be measured after all the entities are created with their initialized component, right after the manager activates the systems. The test was performed for different number of entities created, and each single test was run 10 times in order to reduce discrepancy in hardware during boot-up of application. The test will simply measure time elapsed before the system is done. The exact same type of test is performed for each revision in order to compare results. The results of these tests are written to a text file with the format defined in 4.1.6.

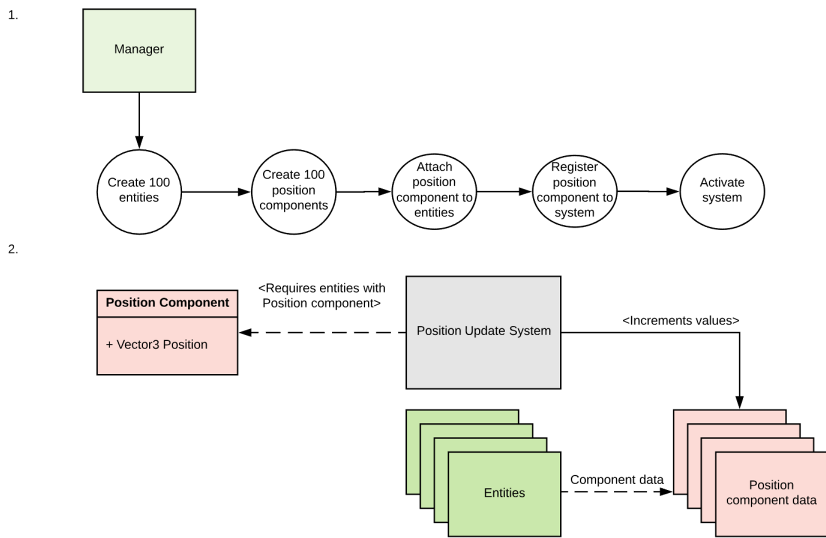


Figure 4.9: The structure of the performance test

The results will be shown in section 5.1.2 and discussed further in section 6.1.1.2

4.3.5 Integrating OpenGL with the ECS implementation

Now that a functional entity-component-system is created in C#, it is time to add graphical support in order to run some visual tests. OpenGL will be used for this task. OpenGL is an industry-standard protocol for high-performance graphics (18). OpenGL itself is not a language, but a standard for common graphics library, written by Khronos. The implementation detail is up to each language. The opengl standard describes api's for accessing the graphics processing unit found in computers. Opengl provides standards for how one should use the graphic processor to render 2d and 3d vector graphics(19).

A C# language binding of OpenGL is required for the ECS implementation. There are currently seven different language bindings for C# listed in the language binding page from Khronos(20). Opengl4csharp(21) was chosen as the language binding after looking through the available options. It was chosen based on availability of tutorials and ease of use. Several tutorials were available for this language, with an easy setup. Tutorials provided by giawa(22) was used in order to get started with opengl using the chosen language binding.

An opengl context needs to be created in order to send commands to the graphics processing unit(gpu). This context represents the internal opengl instance and all its states associated with it. It is not possible to draw meshes or send other commands to the gpu without having a context to reference from. All opengl graphic commands require a context to work from, which should always be provided during initialization from the application. Fortunately, there are several frameworks that makes it easy to create a context without going in depth with the details. One of the more popular ones are FreeGlut (23), which is open-source and free under the MIT license. FreeGlut provides api for creating context and defining parameters such as window size of application. The library itself uses opengl calls to construct the context for the user, however this is abstracted away from the users. Another framework, the Tao framework, is used to access the Freeglut library. This framework provides access to C# libraries most commonly used for game development, such as opengl. Now that the types of libraries required have been established, it is time to import them into the ECS project. The different libraries were downloaded and imported into the custom ECS project.

A simple method for drawing graphics are required for the ECS integration. Opengl render graphics in a similar way as described for Unity in the appendix B.1.1. A buffer of data containing vertices and indices are transfered to the gpu, which will then use the data provided to render the visuals once a draw command has been received. Opengl

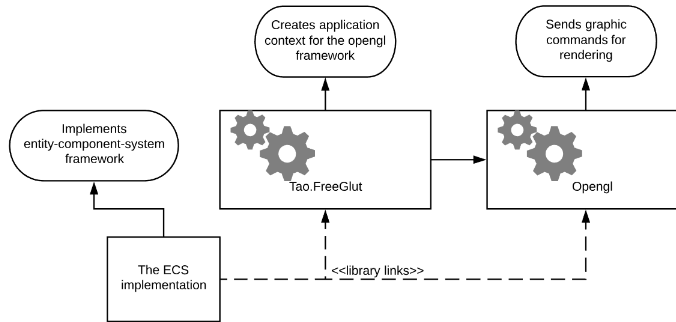


Figure 4.10: The libraries imported to the project and their function

performs a sequence of steps whenever it renders an object, called the rendering pipeline(24). There are several steps in this pipeline, however only some steps are relevant for this part. The application needs to prepare the vertex data for the gpu before it proceeds with the rendering pipeline. The data are loaded into vertex buffer objects stored on the gpu side. Furthermore, a vertex shader must be defined for the vertex stage, which is the step where each individual vertex is processed. The processed vertex data is then moved forward to the primitive assembly. The primitive assembly will take the stream of vertex data and convert it into a sequence of primitives. The primitives determines what the stream of vertices really represents. These primitives can be shapes such as a triangle or a quadrilateral. The type of primitive is decided by the input given to the draw command. The primitives are then constructed based on the values given on the vertex index array, containing the indices for the primitives. After this step, each primitive will go through the rasterization process. At this stage each primitive is broken down into discrete elements called fragments. These fragments have a window space position, values for depth, color and other parameters. Each individual fragment will be processed by the fragment shader, which must be supplied to the gpu. The shader can control parameters such as the color of each fragment. The shaders are the programmable stages of the pipeline, which the users can control. The program in these shaders are written in "OpenGL Shading Language"(GLSL), which is similar to C/C++ in many ways.

To summarize it up, in order to render an object on the gpu, vertex data must be loaded into the vertex buffer and shader programs must defined. The other stages in the pipeline are done automatically without our involvement. Figure 4.11 illustrates the relevant steps involved in the pipeline and the required data.
Following the tutorials available, a simple basic shader program was created. In practice, this is done by writing the whole shader program in a C# string. The string will then

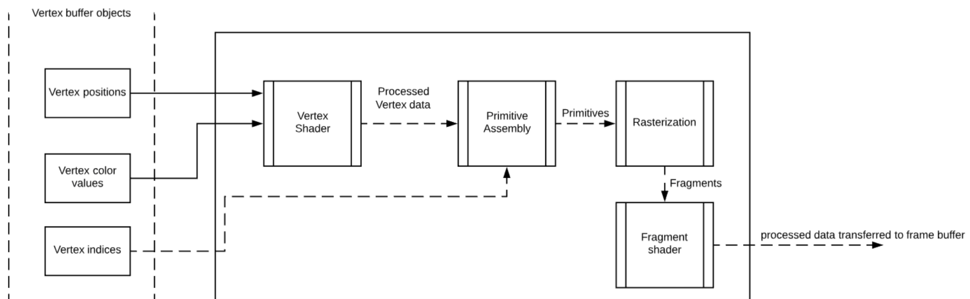


Figure 4.11: The relevant parts of the rendering pipeline in opengl

be compiled into a program which the gpu can execute, with the use of opengl api. The vertex shader will take the position of each vertex and then calculate the position of it on a global axis, based on three transformation matrices.

$$gl_position = projection_matrix * view_matrix * model_matrix * vec4(vertexPosition, 1)$$

The vertex position is defined in its own local coordinate-system, so a model matrix is required in order to transform it into the global world coordinate-system. The view matrix transforms the vertex relative to our view, which would be the camera view. Finally, the projection matrix is responsible for transforming the 3d vertices into our 2d view, making corrections for how close and far an object is to the camera (25).

These matrices can be set by the application before each draw call, allowing the user to change the level of transformation done by each matrix. In addition, the vertex shader will take a buffer of vectors representing color values for each vertex. These values will be passed on to the fragment shader. The fragment shader simply takes in a three-dimensional vector parameter which decides the color of each fragment. The vector specifies the color intensity for the three color types used in the additive red-green-blue color model (26). The complete code for the shader program is shown in the appendix B.4.1

A foundation for graphical rendering is now established after following the steps described. The following steps are required in order to use opengl with the ECS implementation:

- Create a context for the opengl application

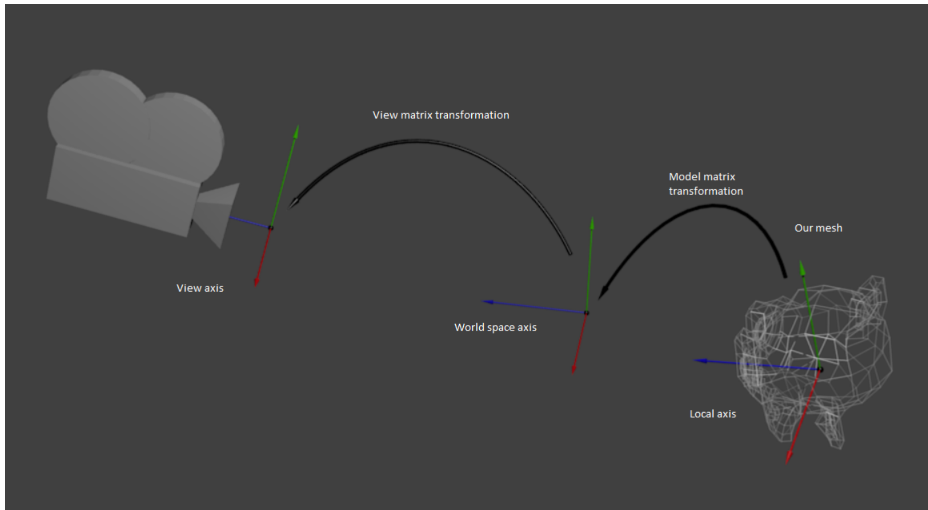


Figure 4.12: How the vertex shader transformation matrices transforms the object - image taken from opengl tutorial website(25)

  - Initialize the window and its size
  - Choose the display mode for the application
  - Create the window itself
  - Choose the idle function that should run each time the processor is ready to render a new frame

- For each frame update, set the viewport and clear the buffer masks
- Fill vertex buffer objects with vertices, the indexes and their color intensity
- Set the transformation matrices found in the vertex shader
- Bind buffer to the gpu
- Send draw command to the gpu to initialize rendering

Figure 4.13 shows an example of a render after following the steps. Complete code following the steps above for drawing a triangle is available in the appendix B.4.2.

4.3.6 Test Application using the entity-component-system implementation

A complete game could have been created with the entity-component-system and opengl integration, however due to limited time, something easier was created. To evaluate

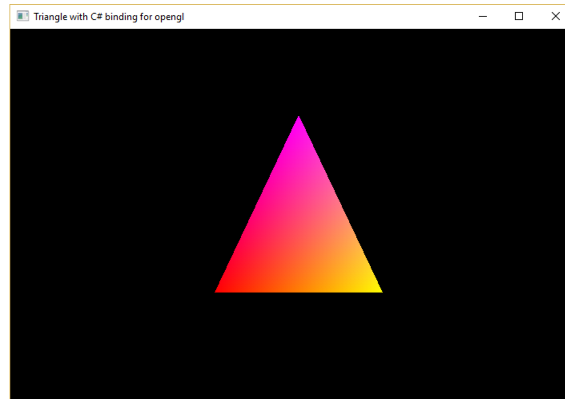


Figure 4.13: Rendering of a triangle done with the opengl binding for C#

the entity-component-system, a test application is required following the requirements given in section 3.2.2. It is important for the tests to analyze both the weak and strong parts of the implementation in order to get a fair analysis. The following key points are in theory improvements for the entity-component-system implementation:

- The use of systems make it easier to divide different tasks in an application
- The use of linear memory layout for the data makes iteration faster due to less cache misses
- No direct references are made to "objects" containing set of data, instead the design works solely on data types.

To test these claims and follow the requirements given, a simple test will be made which simulates a sine-wave. The simulation will consist of several thousand points that together, based on their current position on the x-axis, will simulate a sine wave. The simulation will be created twice using the ECS implementation and traditional object-oriented programming respectively. For the graphical rendering part, opengl will be used. There will be several tests simulating a sine wave, with each test doing it in a different way. The frame rate will be measured multiple times in order to get an average value. Furthermore, the values can fluctuate depending on other external factors, so the same test will run multiple times, reducing the number of potential random discrepancies. Further description of the tests will be given.

4.3.7 Simulating the sine wave

The simulation of the sine wave will be rendered by rendering a large number of quadrilaterals, each representing a single point on the sine curve. Each quad will be evenly

spread across the x-axis, while each individual quad will only move along the y-axis. The y-positions for each quad is calculated using the sinus function with their x-position and elapsed time as the input parameters. These quads will then be placed close to each other, giving the illusion of a connected sine curve even though the quads are decoupled. Four vertices are required in order to render a single instance of a quad. By defining a vertex array consisting of the vertices, along with another array representing the indices, a quadrilateral object to render can be created.

4.3.8 Simulating a sine wave using opengl and object-oriented principles

A class defining each cube point on the graph was created for the object-oriented implementation. The class QuadPoint represents a single quad point object on the graph. This class contains several fields required to define the quad point, as shown in table 4.1. The class has a draw method that will retrieve the data stored on the fields and use them to send draw calls to the gpu in order to draw the quad point. The test will be initialized by creating a large number of objects and adding them to a list. The update function that runs each frame will then iterate through this list, set new position for each quad object and then execute the draw function. Additional classes for color and mesh data was created in order to have more than one layer of classes in the test. This will increase the number of references required and give a more realistic application.

QuadPoint field	Type	Description
Position	Vector3	Position of the point
Square Mesh	Mesh	Contains vertices data for the shape
Color	Color	Contains color data for a single quad
ColorValues	Vector3[]	Contains color data for each vertex of a single quad
Vertices	VBO	Vertex buffer object for the vertex points
VerticeIndexes	VBO	Vertex buffer object for the vertex indices
ColorsVBO	VBO	Vertex buffer object for the color values for each vertex
ShaderProgram		The shader program that will be executed

Table 4.1: The fields in the QuadPoint class

4.3.9 Simulating a sine wave using opengl and the custom entity-component-system implementation

In the ECS implementation, each quad point is represented as an entity consisting of two components. One component representing position value of the point, and one

component representing color value of the point. In addition, a component for describing the mesh is used in some of the tests. Two systems are used, one that updates the position component and another system that reads in the position component and draws the objects. The system that updates position are always executed first. Figure 4.14 shows the structure of the quad point in this scheme. While only one system could have been used for this example, two were used in order to create a more realistic situation for the application. Since the application now uses opengl and its own update function, the manager can no longer be used to activate the systems. This has to be done through the frame function within opengl, which will now instead activate the system update function.

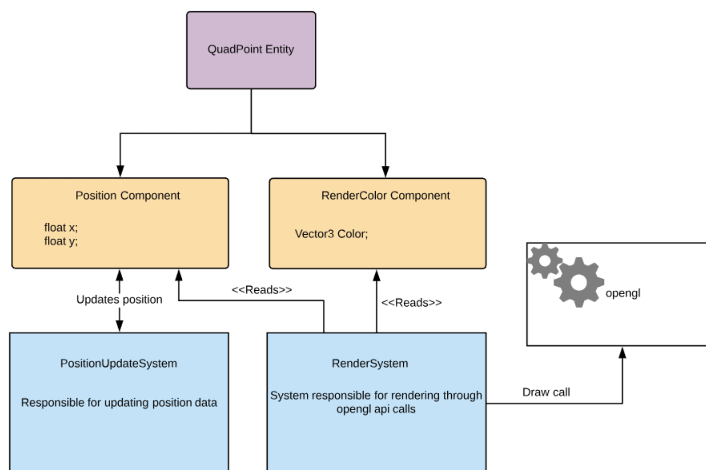


Figure 4.14: Sine-wave simulation structure for ECS

4.3.10 The sine wave simulation tests

The sine-wave will be simulated through a number of different tests, where each test is performed for both programming paradigms. The different tests are listed in table **??**. The tests can be divided into two main groups. One group of tests where the draw command is only called once for the whole graph each frame, this is achieved by having the graph strictly described by the vertices and their local axis relative to the world axis. In essence, the complete graph is drawn by defining each vertex possible on the graph. The other group consists of tests where the draw command is called for each quad point. This only requires the application to define one quadrilateral object and then copy the array for each further draw. We simply modify the transformation matrix before each draw call in order to move the quad to its correct position on the graph.

Test Nr	Description
1	All VBO buffers allocated once, no changes to color values.
2	VBO buffer for vertices and indices are allocated once, with inclusion of VBO buffer for colors
3	Similar to previous test, however color values are now updated each frame.
4	Similar to previous test, but component data for color and position is merged into the same component for the ECS implementation.
5	VBO buffers are allocated each frame for each quad point.
6	The complete graph is drawn by calling the draw api once each frame.
7	Similar to previous test, with only one system used instead of multiple for ECS.

The reason for doing the tests in two different ways are to check how the opengl api affects the performance. We want to verify our results by making sure that the opengl api calls are not too detrimental in regards to performance when it comes to the frame rate compared to the object-oriented and data-oriented design. The goal is to compare the two paradigms, and thus inspecting other external factors will be beneficial in concluding whether the difference in performance is based on the programming paradigm used.

Furthermore, the tests will also have a variation in how vertices are stored for one of the tests. The vertex buffer objects will vary in how they are allocated dynamically. In one variation of the test, the vertex buffer objects will be allocated only once, saving time spent on allocating and deallocating memory for gpu. In the second variation, a new vertex buffer object will be called for each quad. Again, this is done in order to make sure that the results of the two implementations are not significantly affected by opengl api calls.

These tests aim to compare their efficiency against the traditional object-oriented approach. However, the custom implementation is not fully complete and has its advantages and disadvantages. These tests will analyze the different scenarios and verify hypothetical situations. The first four tests investigates how efficient the iteration scheme is with the entity-component-system. The other tests will do the same, but apply to a more realistic setting, such as having points with different meshes to represent ob-

jects of different types. The final two tests will only call the draw command once each frame, instead of calling it for every point. This is one of the most expensive calls in the tests, and thus having only one call will allow us to have a significant larger amount of points on the simulation. This will allow us to test the efficiency of storing and loading component data arrays each frame, which is linearly dependent on number of entities presented.

Each test will run for a different number of objects created in order to include the impact the sizes have. The results from the tests will be written to a text file which will then be parsed by a python script. The results of these tests are shown in section 5.1.3 and further discussed in section 6.1.1.3 All the tests will simulate the same type of graph. Figure 4.15 shows the simulation for one of the tests.

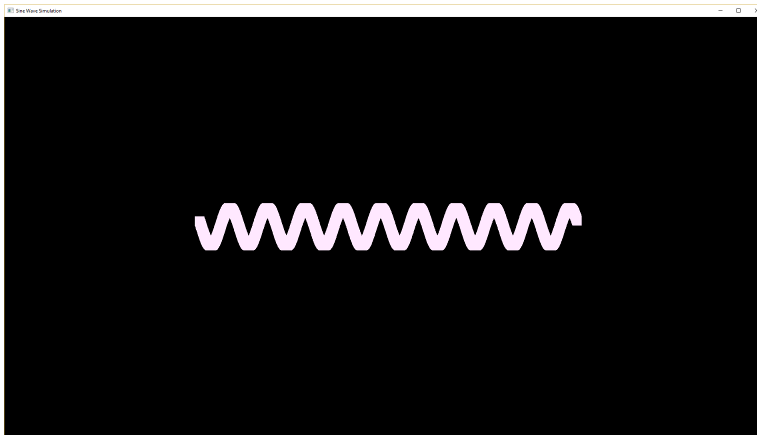


Figure 4.15: Sine wave test with opengl

4.4 Entity-Component-System in Unity

Unity has recently provided support for data-oriented design through their own implementation of the entity-component-system. The structure behind it is quite similar to the custom design implemented in this thesis, however it is significantly more optimized. The memory management of the component data is more complex and linear. It stores data on an entity-to-entity basis instead of just component data of the same type. A brief description of the feature will be given here.

Unity implements the typical entity-component-system architecture. Entities represent an unique object on the game world, with a set of component data that it associates

with. Furthermore, each system performs data transformation on a set of component data that the user defines. The users can define their own component data types and create their own systems. When a system is defined with a set of component data types, it will fill component data arrays with the data that it can operate on. All component data of the same type is stored in what Unity refer to as chunks. The component data is laid out based on their type, where components of the same type are tightly packed linearly in arrays. This structure allows for fast access and iteration through component data. The engine also provides with a new type of components, shared components. Shared components are component data that are shared between many entities, and thus should not be changed often. All entities that use the same instance of a shared component, is grouped together for efficient extraction of data. The inject attribute found in my design is also found in the implementation provided by Unity.

Entity archetypes are arrays of component types that define one type of an entity. It can be seen as a template for an entity and the associated set of components. Entity archetypes can be used to define and instantiate the same type of entities more efficient. The chunks in Unity are all linked to a specific entity archetype, which means that all entities in a chunk follow the exact same memory layout.

Entity manager is a manager that has control over entity data. A collection of api calls related to entities are available in the manager class. The manager can be used to create new entities, check if an entity exists, set or get component data for an entity and other similar entity related methods. Furthermore, a Entity commands buffer object, named PostUpdateCommands, are available in the system classes. The PostUpdateCommand will store entity manager commands to a buffer and then execute it after the system is finished with iterating through the component data arrays. This is necessary in order to avoid corrupting the arrays by inserting or deleting entities that would be legal to that system while the system is not completely done with one whole iteration.

A system can be set with its list of required component types by simply creating a data struct consisting of component data arrays for the different component types. Once the struct has the inject attribute, it will fill the arrays with data and only operate on the entities that have all the required components given in the struct. The PostUpdate-Command should be used here to avoid corruption of the data arrays.

4.5 Pure data-oriented application in Unity

For the pure data-oriented application, a sine-wave simulation will be created using the entity-component-system in Unity. The same application will be created in an object-oriented manner for comparison. The simulation will do the same type of work as the test application written for the custom entity-component-system. A sine-wave will be simulated using a large number of cubes connected together. Each cube will then be given a position along the x-axis and update its y-position by using the sine function with time and x-position as parameters. Iterating through the cubes and updating each movement based on the sine function will then give us a large connected graph that moves like a sine wave. The pseudo code for this behaviour is shown in the code snippet below. To make sure that both solutions execute work in the same type of environment, same type of materials and calculations will be used for the cube points. In addition, both solutions will have gpu instancing activated.

Listing 4.2: Sine wave simulation

```
1  // Get number of ms elapsed since initialization of game.
2  float dt = Time.time;
3  for(int i = 0; i < points; i++){
4      // Get the next cube point
5      Point cubePoint = points[i];
6      // Retrieve the position
7      Position cubePosition = cubePoint.position;
8      // Calculate new position along z-axis
9      Position newPosition = Sin(π * (cubePosition + dt));
10     cubePoint.position = newPosition;
11 }
12 \label{code:2}
```

4.5.1 Objected-oriented sine wave

The object-oriented implementation was inspired by a tutorial written by Jasper Flick (27). Only two scripts are needed for simulating the sine wave. One monobehaviour script is needed to represent a point object on the graph. This class will have variables for x and y position on the xy-graph. Another monobehaviour script is needed for instantiating and updating these points along the x-axis. The points will have the form of a cube and be placed close enough to each other such that they illustrate a connected graph. A couple of game objects were created on the scene with the scripts added as components. The results will be discussed in chapter 5.

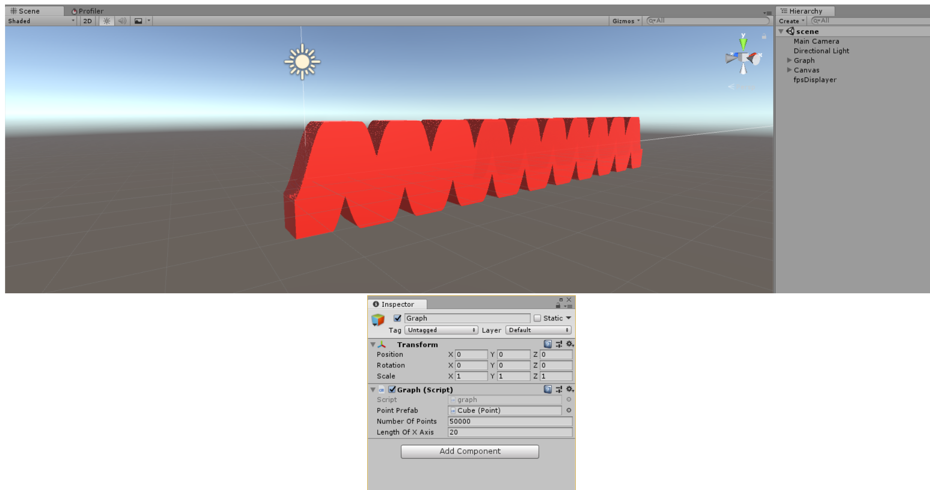


Figure 4.16: Scene view of sine wave in Unity

4.5.2 Data-oriented Sine wave

The newly released entity-component-system framework in unity is used for the data-oriented implementation. The data-oriented implementation performs the same instructions as the one shown in the sine-wave simulation code snippet. A system will be used for updating each point, while another system will render the cubes. When the entity-component-system is used, game objects do not exist on the scene. This means that much of the work and debugging has to be done purely through code, making it more of a challenge. It is possible to attach game objects to the entities through the game object entity interface, however this will not be done for this implementation.

An entity archetype is created for the points. Each point must have a position on the scene and be represented visually as a cube. Unity has several native components available that supports the type of components required. A three-dimensional position component is available in Unity for entities to have. This component has three float values that represent a point on the scene along the cartesian coordinate system. For the rendering part, the mesh instance renderer component can be used. This component contains data for the mesh and type of material used. This is a shared component that multiple entities can use together, since the cube mesh will be the same for each point. Finally, one more component is needed for the entities, the transform matrix component. Each cube entity must update its position along the y-axis, and for this reason, there must be some mathematical definition for the translation. Unity provides with a transform matrix component that can be added to the components. The transform

matrix is used in order to calculate transformations such as translation and rotation. The default values are sufficient to use for the application and further changes are beyond the scope of this thesis. Based on the given information, the cube entities only need three type of components, position, transform matrix and a mesh instance renderer component.

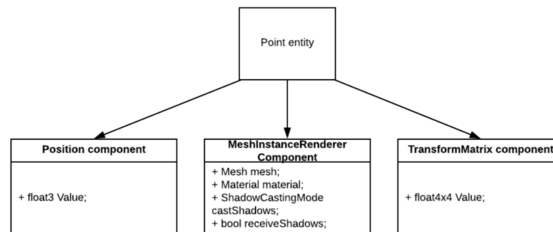


Figure 4.17: The entity that represents a single point

Only one user-defined system is required for this application. A system that iterates through each point on the graph and updates the y-position value. The system only requires the position component, which means that a component data array must be defined within the system. The system will as a result only be legal for entities that have the position component, which all the cube entities have. Unity provides with another native system that will render the points as cubes on the scene. The system is provided by Unity and will automatically render all entities that have a position, mesh instance renderer and transform matrix component attached to it.

There is no simple method for accessing the cube mesh or material type in Unity through scripts with the entity-component-system. A workaround is to create a cube game object on the scene, and use data from this object to define the mesh and material. The game object data was extracted by the use of a Unity api, GameObject.Find(), which finds game objects on the scene and returns object data. The cube object was first created on the scene with the desired size and material before the data was extracted in the script. The extracted data was added to the mesh instance renderer component in order to define the graphical part of the cube entity. The same method was applied to define parameter values for the graph, through a setting game object. Code snippet demonstrating this is shown in the appendix B.7.1

To bootstrap everything and activate the simulation, an initialize function was created with a game object. This game object will simply run once and initialize all the data

required for the systems.

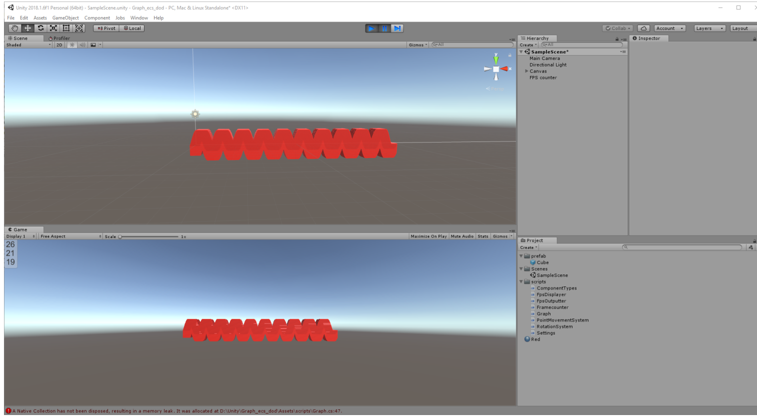


Figure 4.18: Sine-wave simulation in Unity

4.5.3 Testing of the sine-wave simulations

The main performance criteria for the two simulations will be the average frame rate. The test will be performed multiple times with different number of objects spawned in each test case. All the tests will be performed with a window resolution of 1600 x 900. Both solutions will be built in order to remove all kind of overhead related to the unity editor.

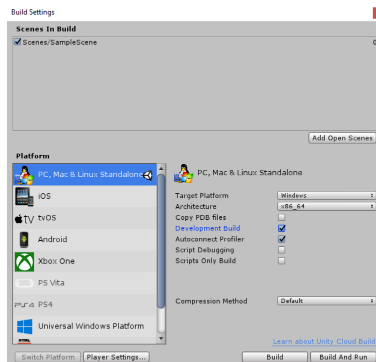


Figure 4.19: Build settings for the tests

The values will be written to a text file that is parsed by a python script as described in section 4.1.6. To make sure that the real effct of data-oriented design is inspected, the Unity profiler with the profiler data exporter script will be used to inspect cpu usage

times. This part of the analysis will only be done for 50000 objects spawned.

4.6 Data-oriented design for Dwarfheim

The architecture behind DwarfHeim, the conversion strategy and implementation will be described in this section.

4.6.1 Computer architecture of Hybrid/Pinecone

DwarfHeim is being developed with their own custom engine on top of the Unity engine. The custom engine, hybrid/Pinecone, is an engine specialized for real-time strategy games. Converting the game will require changes to the specialized engine as well since the game is deeply integrated into the engine. For this reason, it is important to understand how the engine works, as changes to the game will involve changes to the engine. The different sections described here will be relevant for the data-oriented transformation, as these parts will be affected by changes done to the engine and game.

The following sections will go more in-depth for the parts of the hybrid engine prone to changes in order to get a data-oriented solution.

4.6.1.1 Networking model

The game is mainly going to be played as a multiplayer game, meaning it requires networking. The network development of the engine is currently not finished.
It is desirable to have a client-server model for these type of games. Each player in a match acts a client, while the server is responsible for synchronization between the different clients. It is important for the clients to have their game states synchronized, otherwise each player would play their own version of the match. The server is responsible for multiple things, such as making sure each unit in the game has the same position across all clients and calculating non-deterministic actions. In addition, the server is responsible for giving commands to each client. The commands can be of different nature, such as moving towards a position or attacking another unit.

The engine uses Photon Unity Networking framework, which is a 3rd party asset in Unity. The framework takes care of all the back-end work necessary for networking, as well as providing an user-friendly interface. Games that utilize their framework are hosted on their globally distributed cloud service.

The engine is currently not using a client-server model in the sense explained previously. Instead, each client can be seen as a peer, with one of them being the host. The host peer will act like a server, but is actually just a client that does additional work in order to allow communications between the different clients. This is possible through the master client api available with the photon framework. Basically, all the clients in a game will act as a peer, with one of them being defined as the master client. The client that is responsible for starting the match is usually the master client. The other clients will then join the game match provided by the master client. Once a peer is a master client, by enabling a boolean, it will act as the host. It is then possible to execute "server" specific code by checking if the peer is a master client or not. In practice, each client uses the same code, with one of them having the master client boolean set. Each client will then check if it is the master client, and as a result execute server-specific code if it is. The game is still in its early developmental phase, so using such a model makes it easy to quickly test networking part.

For testing purposes, whenever game code is tested, the client will act as a master client and create a new client that will be loaded with the same scene. The testing environment will then have two clients, with one of the clients being the master client. The scenes that are used to play with will act as a normal client, with the master client in the background. This scheme allows for testing of the networking, client code and server code at the same time.

4.6.1.2 World Objects

All objects in the game that can be directly interacted with by a player is derived from the World Object class in the hybrid/Pinecone engine. The class defines some general parameters that all derived objects must have in order to be manipulated by a player. Having a base class for all objects in the engine allows for extensive use of polymorphism. The class provides several virtual methods that are to be invoked after fulfilling certain conditions, such as first time being initialized or when the object is dead. The world object type can be used as a generic type to access data of underlying types without knowing the exact type of that object. All references to derived classes of world object can then be done through the world object base class itself, allowing the program to execute the virtual methods that the derived classes overrides. The derived classes can then invoke their own specific methods within the virtual method. This scheme allows for all references through the world object, making the code more modular and adaptable to changes. All references to the world object derived classes in hybrid/Pinecone is accessed through the base class. Furthermore, the class contain fields for the prototype

id, unique name and the prototype it is created from.

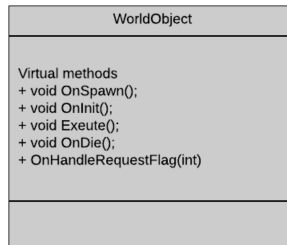


Figure 4.20: WorldObject class and its virtual methods

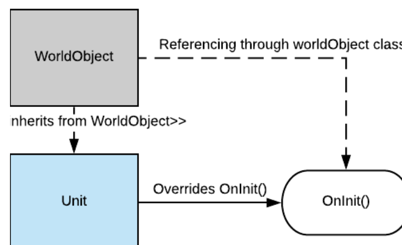


Figure 4.21: Example of a derived class Unit that has its OnInit() function invoked virtually through the base class

4.6.1.3 Basic Commands, Basic Actions and Basic Command Chain

Basic commands are commands that a world object can perform. These commands are sent from the server-side to the client. Each command has a basic action associated with it. Basic actions are the actions that a world object can execute, such as moving towards a position. These actions are deterministic, meaning that the outcome of the action is always the same for the same input, regardless of the type of hardware the client has. Basic command chains are queues of basic commands that are to be executed in order. Each world object has its own basic command chain, which it will iterate through each time a basic command is completed.

The type of action a basic action perform is limited due to it being deterministic. The reason for the deterministic requirement is because every client can have different hardware. Different hardware in the clients can cause calculation error and thus give small variances in the results carried out by the actions. An example of this, is the floating-point unit found in computers, which performs mathematical calculations with floats. The results of these calculations can vary from computer to computer, giving different results(28). The server will then as a result not be correctly synchronized with the clients. In addition, the clients won't be synchronized with each-other. This problem

forces the design to limit basic actions to be deterministic, reducing such errors.

To further illustrate the problem of non-determinism, imagine that an unit is to walk from point A to point B, with an obstacle between the two points, as shown in figure 4.22. Let's assume that the two paths outlined in the figure, has almost the same exact distance with only a small variance in value. Now if we were to calculate a path with two computers that have different hardware, the resulting path distance calculated might have a small variance. This means that both path A or path B is valid, depending on the hardware. The action is thus not deterministic in this case, giving different results for movement. The hybrid engine solves problem like these by having non-deterministic

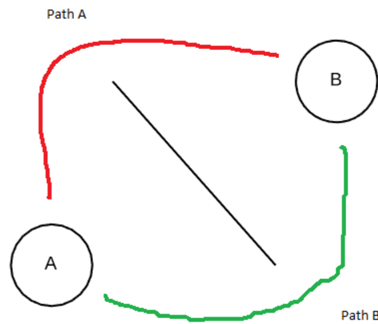


Figure 4.22: Two different possibilities for movement

calculations performed on a server, where the hardware is equal. The resulting path calculated in the server will then be broken down into a chain of small deterministic movements, such as movement in a straight line, and sent to the basic command chains for relevant world objects. The deterministic path is shown in figure 4.23.

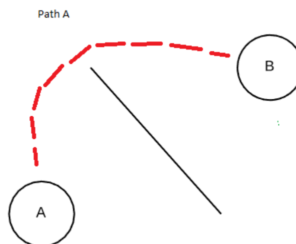


Figure 4.23: A chain of deterministic basic commands where each command is a move action in a straight line

4.6.1.4 Abilities, Utilities and Sentries

Abilities are actions that a world object can execute. They range from movement actions such as following another world object to combat-specific actions such as attacking another unit. Abilities will in essence create new basic commands for the world object to perform. The logic behind abilities are calculated on the server-side only. An ability request will be sent to the server once a world object activates an ability. Abilities will then be executed on the server-side, calculating new set of basic commands for the world object to execute. Abilities can be divided into two subcategories, instant abilities and active abilities. Instant abilities are performed instantly in a single frame. A typical example of such abilities are those who add stat boosts to units immediately.

Active abilities are abilities that executes over several frames. The structure behind these abilities are more complicated than those found in instant abilities. Active abilities consists of a set of utilities. Utilities are a type of class that are responsible for creating new chains of basic commands. Each utility is responsible for its own type of commands to send. Once an utility is finished with creating a new basic command chain consisting of new commands, the chain will be transferred to the relevant world object. The structure of utilities can be depicted as a graph tree, where each node is an utility. An active ability will only have one active utility in all instances. In order to traverse to the next utility on the graph, an utility must meet its exit condition. Exit conditions are conditions that moves the utility out of a valid state to the next one. An utility can have more than one exit condition. Once an exit condition is met, the active ability will look at the next utility to perform in its node list. Utilities can be seen as block functions, they take in input values, calculate new basic commands based on these and transfer it onward to the clients. If an utility's exit condition is met, it will end execution. An utility will perform one specific function, such as calculating a path for movement.

To demonstrate how an utility works, the pathfinding utility in the game will be described. The pathfinder utility calculates a new path for a world object to move towards. It takes the destination as input, calculates a new path and outputs a basic command chain consisting of move actions. The pathfinder utility has two exit conditions, one for when the destination is reached and one for when the destination is suddenly invalid. This can happen if the destination is some target, and that target happens to be removed during calculation. When one of the exit conditions are met, the active ability will either traverse to the next utility or finish the ability given that there are no other utilities to traverse to. Figure4.24 demonstrates the pathfinder utility with its inputs,

exit conditions and internal logic. An ability for movement can then be created by only

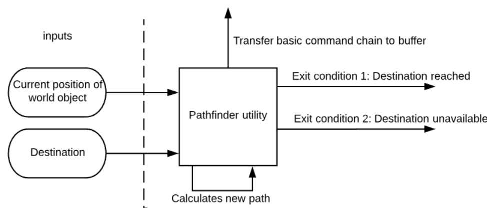


Figure 4.24: The pathfinder utility in Dwarfheim

using the pathfinding utility. Another example is a close-range attack ability, which uses two utilities. One utility for pathfinding and one for doing the attack. The pathfinding utility must be executed first in order to ensure that the unit is close enough to the target. The graph nodes for the abilities are shown in figure 4.25.

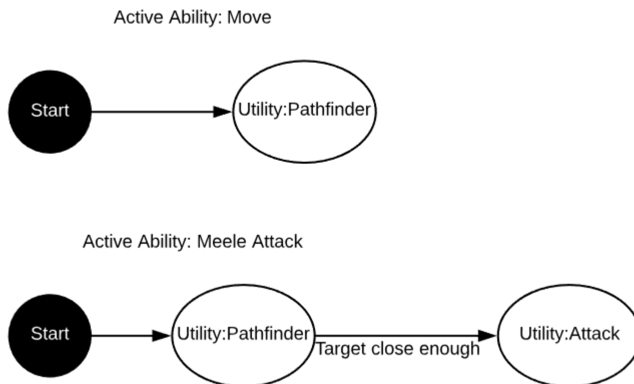


Figure 4.25: Two examples of abilities created by using a set of utilities

There is a reason for why this structure is used. Once a game has defined a varied set of utilities, they can be connected together in multiple ways, creating different abilities. Two abilities with the same set of utilities can be different by simply having different traversal order for the utilities. This allows developers to create new abilities in a fast way, as long as the utilities are defined. In addition, the hybrid engine provides with a graphical user-interface for creation of new abilities. This allows non-developers to create new abilities without having to write code. Only the code for utilities needs to be written.

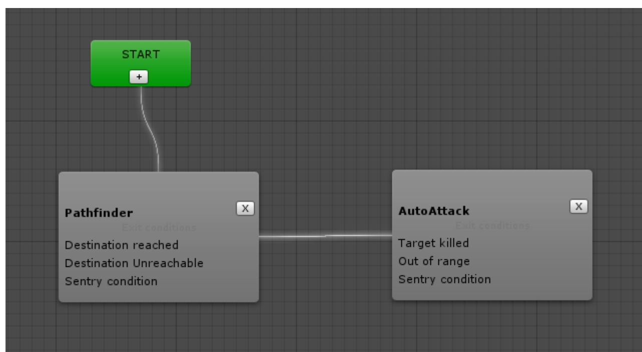


Figure 4.26: Ability editor view, showing the active ability for meele attacks

4.6.1.5 Agents of hybrid engine

Agents of the hybrid engine plays an important role in making the world objects interactable. They are all monobehaviour scripts that give access to certain types of data for manipulation. Each game object using this engine will have to include some of these agents as part of their component list in order to access certain functionality. The agents are one of the first areas that will need to be transformed into a data-oriented structure as they play a vital role in most of the code logic seen in the engine. For instance, each game object in the game can have their own view agent, which is responsible for playing the correct animation during execution. Another agent is the command agent, which is responsible for updating the next command for a world object to execute. To summarize it, these agents are attached to game objects so that they can be responsible for the data manipulation in a specific domain. Currently the engine have 6 different agents and they will be briefly described here. Note that the assisting figures do not show the complete class, only some of the more important attributes.

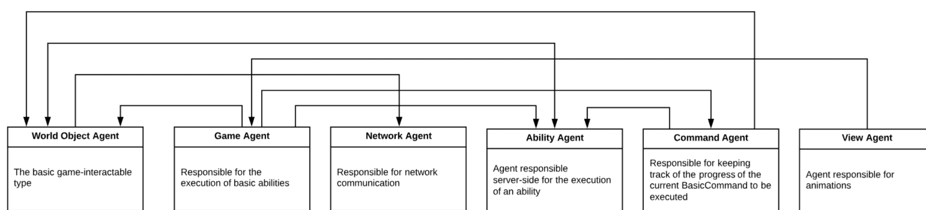


Figure 4.27: Agents of the hybrid engine, the lines show dependencies

World Object Agents Each world object must include a world object agent component. The world object agents act as an interface to the world object data in a game object. The agent is a monobehaviour type which means that a game object can include it as a

component, thus getting access to the data it employs. It is responsible for initializing the world object class by calling the initializing function. Furthermore, it is responsible for synchronizing the transform of the world object on the network, the server, with the transform found in the client version. This synchronization happens through the update function in the world object agent component. The tasks of world object agent can be listed as following:

- Provide an interface to the world object data by having a reference to it.
  - All other scripts can access the world object data by having a reference to the world object agent component linked to a game object.
- Initialize world object data
- Synchronize client transform with server transform
- Keep track of abilities that the world object can perform
- Keep track of effects related to world object stats

Command Agents Command agents are responsible for keeping track of the basic command chains attached to world objects. It tracks the progress of the current basic command that is to be executed. Every game object that has the command agent component will have a basic command chain, as the agent component contains a basic command chain object. The command agent can be seen as the interface to the basic command chain in a world object. Inserting new chains of commands, deleting current chain and iterating to the next basic command in the chain, are all methods available through the command agent.

Game agents Game agents are responsible for execution of basic actions contained in basic commands. Each basic command comes with a basic action, which is a type of action a world object can perform. This agent will check the current action available through the current basic command by referencing through the command agent. This happens every frame through the update function found in a monobehaviour script. Every game object that includes a game agent component will then be able to execute basic actions. The same game object must also have a command agent in order to access the basic command chain.

Ability Agents Ability agents are only found on the servers-side of the engine. These agents are responsible for executing the ability requests that a client sends. The agent keeps track of the current ability and makes sure it correctly iterates to the next utility

found within an ability. The agents have their own reference to a basic command chain, which it will fill with new commands given by the utilities. Furthermore, the agent will transfer the newly constructed chains over the network to the respective clients that it belongs to. One key difference between this agent and the others, are that the ability agents do not belong to a world object. A single instance of an ability agent is independent and can perform work for different world objects.

View Agents View agents have access to animators found in a world object. The agent is responsible for setting and playing the correct type of animation.

4.6.1.6 Prototype-based model in DwarfHeim

Prototype-based programming is briefly described in the appendix B.5. DwarfHeim and the hybrid engine has a well-defined support for prototype-based instancing of new types through their own customized editors in Unity. It is used for creating new type of abilities, units and basic actions. The motivation behind this feature is simple, it allows non-programmers of the team to develop new types without having any software-development knowledge. In DwarfHeim/hybrid, this model is implemented by having base classes with a set of modifiable fields for each base type representing prototypes. In the case of a base class for units, the fields can represent values such as attributes, type of equipment it can equip and list of ability ids for the abilities it can execute. Units with different stats, abilities and equipment can then be created by creating an instance of this base type with the required values to the fields. This makes it easy to create units that are quite different, without needing to write software code for each different unit. The only thing one needs to write code for, is the base class and the associated abilities it can use.

When a prototype is created by a base prototype class, it will be stored in the prototype library as a json file containing data for the prototype. In this library, each prototype will be linked to the base class it is created from, and the set of values it has been given to the base class fields. An instance of the prototype can then be created by retrieving the data stored on this library. The structure behind this model is similar to just creating instances of a class and then assign it the values one desire, however this has to be done with code. The major advantage of the prototype-based modeling is that you do not need to write the instances in code, you can simply do it through an editor and allow it to deal with the rest back-end, given that some developers have already created the interface for it.

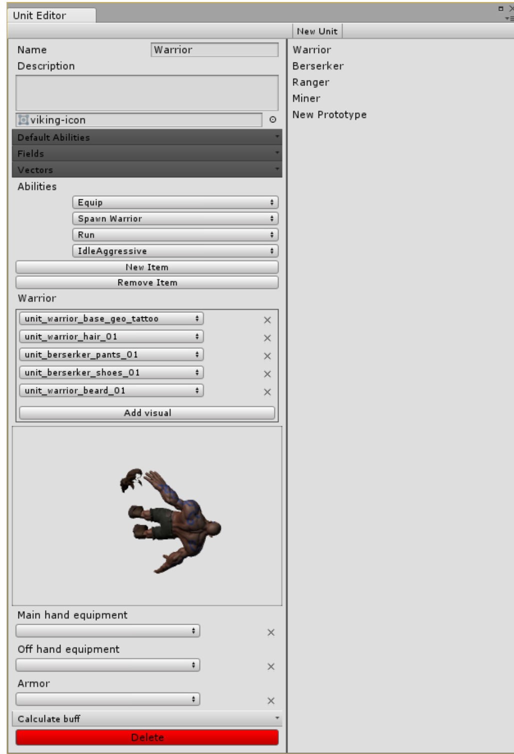


Figure 4.28: Unit editor in Dwarfheim

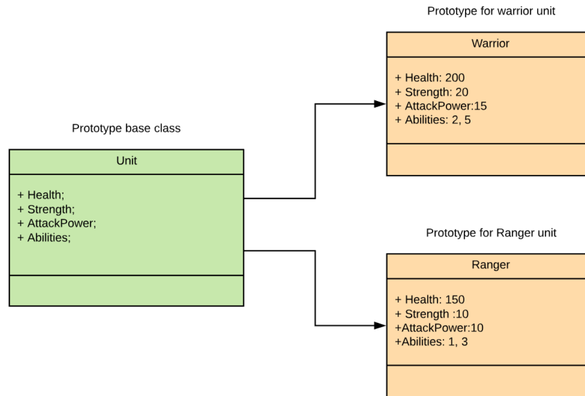


Figure 4.29: Two unit type prototypes created from the unit base class

4.6.2 Methodology for converting to data-oriented design

Now that a general understanding of the architecture behind the engine and game have been established, it is time to consider the methodology for transforming it into a data-

oriented solution. To avoid confusion with the naming scheme behind game objects components and components in entity-component-system, the ECS abbreviation will be used as a prefix to indicate parts that are in the entity-component-system domain.

4.6.2.1 Scope of conversion

Since conversion of the complete game would be infeasible due to time and complexity at its current state, only small sections of the game will be converted. The transformed parts will create the foundation for a potential data-oriented DwarfHeim. Collecting results for the modified parts will indicate whether the conversion has brought improvements.

Since the main task behind the conversion is to improve performance server-side by reducing cpu usage time, the server will be the main part being changed in this conversion. The server is responsible for receiving commands from the player and return a new chain of commands for the game objects to perform. This part will require changes to the current agent system in the engine and the basic command chain structure. To implement changes to the server and client, with the ability to verify correct behaviour for the changes done, the unit model in DwarfHeim will be changed. Units are interactable types in the game representing dwarf characters, which are able to execute different type of abilities depending on type of unit, such as being a warrior unit or a ranger. For this task, the warrior unit will specifically be changed. Figure 4.30 shows an overview of the unit model with the important types of objects it contains in the original object-oriented DwarfHeim version.

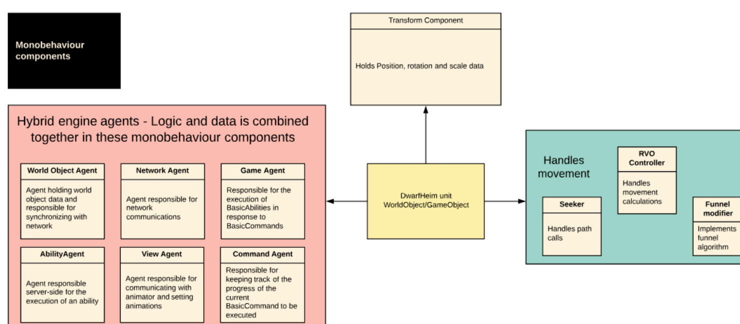


Figure 4.30: Overview of the unit model and the attached monobehaviour components - Not all details are shown

Converting the warrior model will require changes to the current implementation of

abilities and utilities as well, as they are executed server-side and responsible for generating new basic actions. This part of the code needs to become more hard-coded and less polymorphic, meaning that virtual functions are no longer possible. Only a few number of abilities will be converted for this thesis. The current plan is to convert the abilities associated with movement and melee attacks. Changes to some of the utilities must also be done since abilities use them. Finally, basic commands and basic actions will also be converted as a result of this.

To summarize, the parts that will be converted will be listed here. Not all changes are listed up, only the major ones.

- The unit type.
- The agents - command agent, ability agent, game agent, view agent.
- Basic command for standing idle, movement and melee attack.
- Utilities for pathfinder, idle and melee attack.
- Abilities for standing idle, movement and melee attack.

Much of the changes will be done through the agents. Figure 4.31 gives an overview of the important data parts and structure of the agents in the engine.

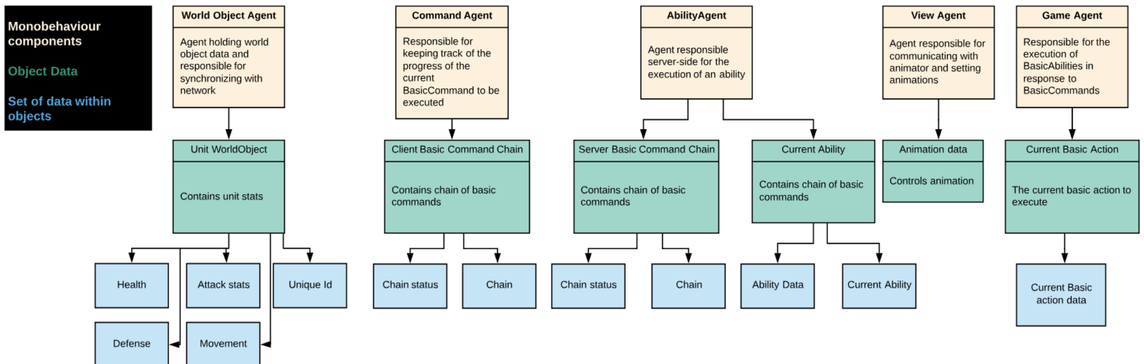


Figure 4.31: Agent data structure prone to changes during the conversion

4.6.2.2 Limitations with the conversion from object-oriented to data-oriented

The ideal conversion would be complete separation between data and logic, as shown in figure 4.32. All data would be contained in ECS-components while systems would

perform logic based on the data found within these components. Furthermore, all sets of data are of value types and thus traversal are faster for the systems, making the processor more efficient.

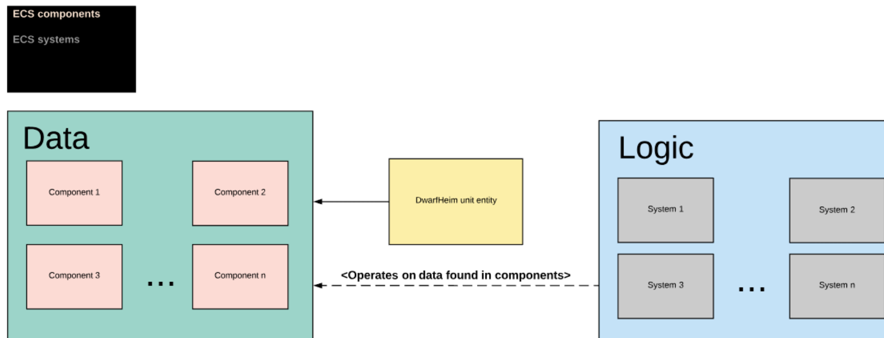


Figure 4.32: Ideal implementation for a data-oriented approach

The entity-component-system in Unity has the goal of providing with linear memory-layout for component data. Achieving this functionality imposes some limitations to the engine. At the time of writing, the development behind the entity-component-system in Unity is still in its early stages. The limitations written here might not be any limitation in the future. Here is a list of some the limitations found with the entity-component-system in Unity that affects the conversion:

- All data types in a component struct must be blittable.
- Arrays, lists and other containers with non-fixed sizes are not allowed in a component struct.
- Arrays of fixed size can only contain basic data types such as integers and doubles.
- Polymorphic behaviour is not possible with the current design.
- The Post update commands api lacks features found in the normal Entity manager.

Blittable types in C# are data types that have the same representation in managed and unmanaged code. These types includes only the basic types such as integers, bytes and integer pointer. No reference types are blittable, neither is a boolean. This means that the engine is quite limited in what it can use in the ECS-component data fields. This makes it challenging to transform the hybrid code as it is very object-oriented and contains several fields with non-blittable types.

The ECS-component data fields can not contain fields that vary in size. The arrays must have a fixed length, and can only consist of basic blittable types such as integers or floats. This makes it a challenge in converting the current object-oriented basic command structure into a data-oriented one. It is possible to declare ECS-components as shared, which can contain all types of fields. However, shared components are shared by multiple entities and is not meant to be changed often, meaning that it can't really be used as a basis for data found within a single entity.

The lack of polymorphic support means that the code will have to become more functional and more "hard-coded". It becomes a challenge to operate with object references as they do not really exist anymore. Furthermore, the prototype structure which is heavily object-oriented and relies on dynamic allocation of parent classes, will potentially no longer be a beneficial solution with the entity-component-system.

Finally, the PostUpdateCommands api lacks some methods that are available in the normal entity manager. The api lacks many of the features that the normal entity manager have. Especially to functions related to dynamic creation and deletion of entity component data based on component type, which is not available in the api. This makes it more difficult to dynamically work with entities and its component data as the application must always know the type beforehand. The current lack of it makes it more difficult to transition into a data-oriented implementation that is more dynamic in nature and less reliant on static information.

The fact that DwarfHeim and the hybrid engine is object-oriented brings another set of challenges. The basic command chain structure is very object-oriented in its nature. The size of the chain is not fixed, making it difficult to convert due to the limitations associated with the entity-component-system. Furthermore, the chain references basic command objects with basic actions in a polymorphic way. Another problem is the calculation of paths for the game objects, as shown in figure **??**. The module uses a 3rd party asset that requires monobehaviour components in order to calculate the path. Making this part data-oriented would mean that the internal structure behind the path finding module must also be changed, which is not a feasible option for this thesis due to complexity.

The limitations by the entity-component-system and the current implementation of the game will limit the conversion. Converting the complete game would be infeasible due to complexity and lack of time. The conversion can be a combination of object-

and data-oriented principles, at the cost of potential performance gained by not going pure data-oriented. Figure 4.33 shows an overview of the model for the unit in this hybrid conversion. As the figure shows, the solution will consist of both entity-component data and object-based data through monobehaviour scripts. In addition, logic is both found in systems and monobehaviour scripts.

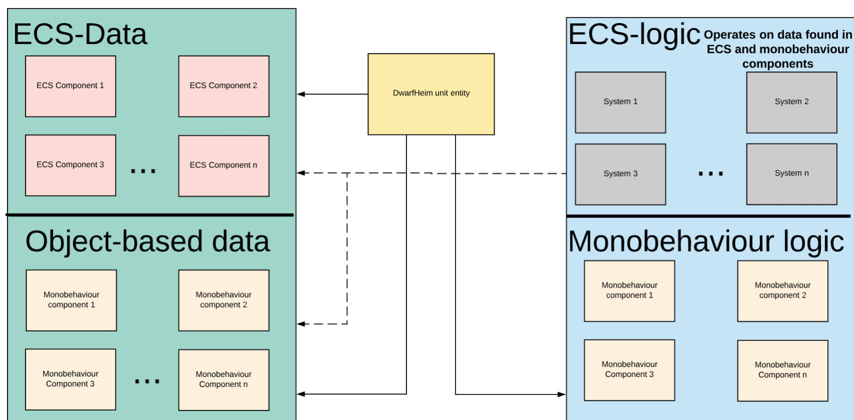


Figure 4.33: Overview of the model for the hybrid solution

4.6.2.3 Methods for conversion

The different methods used for conversion will be covered here, with examples for some of them. Due to the nature of the changes done and the size of the overall architecture, the methods will be described in parts. Figures will illustrate the different methodologies as they are explained. Not all details will be provided in this thesis as there were large changes done to the code. Only a general understanding of the strategy will be given. Both the hybrid engine and the game itself will be referenced as DwarfHeim from this point on. Figure 4.34 gives a brief overview of the final implementation without covering every detail related to the implementation. As the figure shows, the unit model is now defined through a set of monobehaviour components, ECS-components and shared components. The movement module is a 3rd-party asset that handles movement in an object-oriented manner, and was not changed for this thesis.

4.6.2.4 Working with the existing DwarfHeim code

It is desirable to work with the current existing code available.. Use of 3d-models, animations, game-logic and general code architecture are all available to base the work on. In-game objects are accessible through game objects in the traditional object-oriented

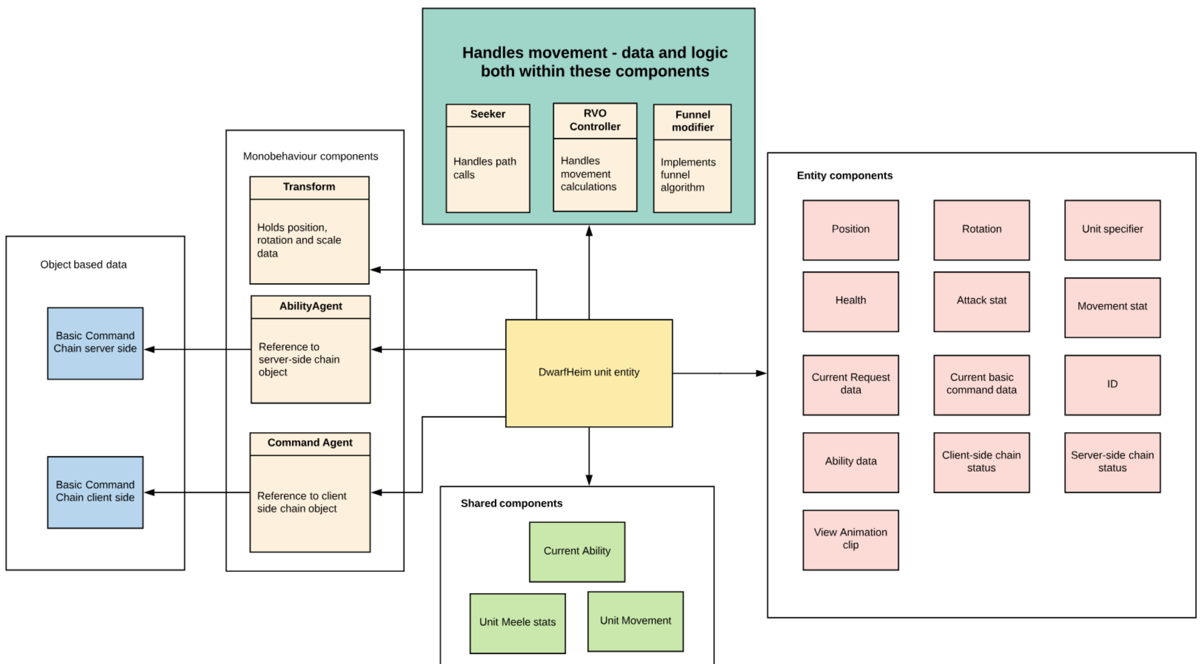


Figure 4.34: Overview of the data structure in the converted version

way. A good starting point is to give the objects entities, making it possible to access them with the new entity-component-system, while at the same type keep much of the same logic available in the regular class. It is fortunately possible to do this with the new entity-component-system. Unity has added a method for slowly transitioning from object-oriented style to the new entity-component-system, by using the game object entity component. This is a normal C# script that gives an unique entity id to an object class. An interface to the entity-component-system is made possible for a class once it has this component. There are some benefits in using the game object entity in the beginning. There are a lot of classes in the codebase that access data through prototype libraries, gets initialized by manager classes and other similar primitives that setups the state of the game. It would be difficult to take the current classes and re-write them from scratch with the new entity-component-system class, as that would require additional changes to the already existing overhead that takes care of the initialization for the game. Using the game object entity interface provided by Unity will remove the need to change the networking instancing and prototype library structure. Instead, entities can be based on existing classes by copying the data found in the newly instanced objects once they are initialized.

An example where this method is used, is the unit class in DwarfHeim. The class contains stats for attributes such as move-speed, health and attack damage. The stats for these attributes vary based on the different type of units available. The units differ in how those values differ, along with the type of abilities that they can execute. The data for these fields are stored in prototype files, and a new unit of a type is instanced by extracting the data found in the prototype file. Instead of changing the current structure of the system, a game object entity component is used. The game object entity component is added for the class during the initialization phase. Once the game object is associated with an entity, ECS-component data is added to the entity through the entity manager. The component data is simply the data found in the unit object. After the data has been copied to a new entity associated with the unit object, the data in the unit object can be ignored. All further processing is now done with the entity-component-system. The original object-oriented object was only used as a template for type of data to copy for a new entity, which represents the old object.

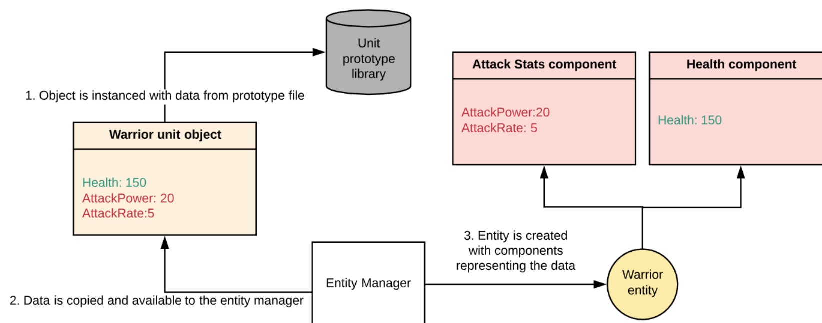


Figure 4.35: Example of an unit object being initialized then used to create the equivalent entity in entity-component-system

4.6.2.5 Animations and dwarf 3d-models

Using the animations and 3d-models available in DwarfHeim is not easily done with the entity-component-system. Currently, at the time of writing, a good animation system is not supported. Nor is it easy to represent 3d-models unless they are simple meshes. The DwarfHeim 3d-models representing units consist of several visual components bundled together. This makes it difficult to render with the entity-component-system. It is still desirable to test the entity-component-system with animations and proper 3d-models, to confirm that those assets would still be usable for the conversion. For this reason, both objects and entities were used for the parts that have 3d-models and ani-

mations.

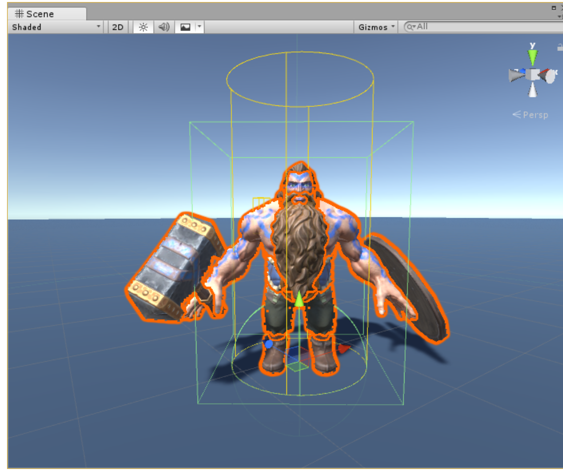


Figure 4.36: Warrior unit 3d-model

Game objects that have graphics and animations are divided into two parts, one part that contains the animations and 3d-model components, and another part that handles logic and other data related to the object. The 3d-models are displayed based on the current transform position, given by the transform component. The transform component is handled as an object, which means that accessing the data found in the transforms are not done in a memory-efficient way. It would be better if the game objects also had a ECS-position component in the entity-component-system, which handled all logic related to position. The problem with only having this position ECS-component is that the 3d-model attached to a game object, is rendered based on the transform component, and not the position ECS-component. The latter would be possible by attaching a mesh to the entity-component-system, however the 3d-models are not simple meshes that can be represented in that way. Both positions are needed in order to make this work. The transform position component in the traditional game object based system will be used to render the position of the actual 3d-model. Another position component in the entity-component-system will also be included as a ECS-component for the game object. All logic requiring position will be done through this position ECS-component. After one frame is finished, a system will be used to synchronize the position component data in the entity-component-system, with the transform position data in the old object-based system. This system will update last and read in all position data found in the ECS-components, and synchronize them with the transform component. This change leads to more overhead as the application now requires

an additional system for synchronization between the position components.

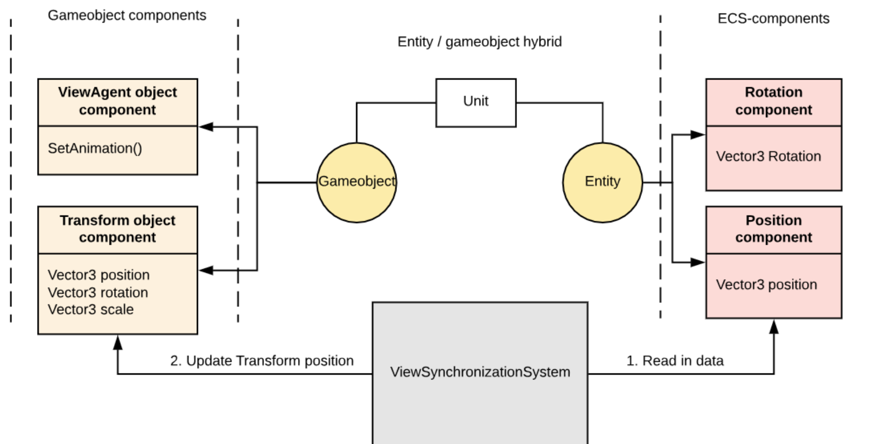


Figure 4.37: Unit consisting of both a traditional game object and entity with components - A system is used to synchronize the data between the two domains

The new change allows for faster memory access for position data, while at the same time making it possible for the engine to render the graphical models with the old transform system. The same will be done for rotation. The current approach should only be temporary, a pure solution should get completely rid of the transform component and instead render graphics and animation with the entity-component-system. Using a mix of both solutions provide with easier implementation, but at the cost of performance. Support for disabling them should also be made possible so that the data-oriented principles can be tested appropriately.

Animations are set through the view agent component. To render different type of animations for the units, a system referencing view agents will be created. Only one system will have access to this view-agent, to reduce the usage of reference types.

4.6.2.6 Representing object data with entity-component-system

Data is defined and accessed as fields within a class in traditional object-oriented programming. With the entity-component-system, data is instead stored in components and accessed through entities. To convert an object containing data fields, one must create components that contain the same type of data as the one found in the template class. Several components can be used to represent data found within a single class, dependent on how the data is accessed by systems. If a class has set of data fields that are

accessed in different ways, then it would be wise to split the fields into multiple components, making it easier to distribute work to several systems. For example, the unit class in DwarfHeim has several data fields related to object state within the class object. The attributes represent values such as the current health of an unit, the move-speed and attack turn. All these stats are related to the unit, but are not used together when engaged with. Whenever an unit is attacked, the health it will lose is not relevant to its move speed or attack turn. The data attributes are used independently of each other based on different circumstances. For this reason, it is wise to split the data fields in that class into different components. Different systems will be responsible for different type of components. A system which updates health status, will only care about the health data, while another system that updates movement will care about move-speed and not the health. For this reason, multiple components should be created for the data fields in the class. Figure 4.38 demonstrates an example of this.

Following the situation given above, the following strategy was devised:

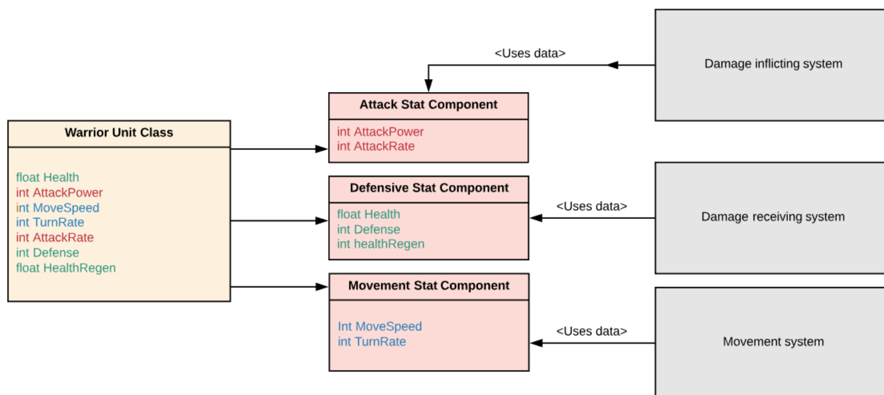


Figure 4.38: Splitting data fields of Warrior object into multiple ECS-components based on their usage

1. Identify how the different data sets in an object is used.
2. Split the data fields of objects into ECS-components.
3. Group data sets that are used together in the same ECS-component

Data sets are grouped together if they are likely to be used together in a system. This scheme allows for clear separation of data. A system can be created for the different ECS-components and perform logic on them easily through this separation. Not all

data-fields can be directly converted due to the limitations of components described in 4.6.2.2. Different tactics must be employed for data-fields that have arrays of non-fixed size or non-blittable types. Changes done to these kind of fields did not follow a general strategy, but were rather changed based on their usage. They will be covered in further sections.

4.6.2.7 Converting the behaviour of monobehaviour scripts

The current codebase of DwarfHeim consists of several C# scripts defining behaviour through components. The monobehaviour scripts usually have an update function that runs every frame for every component on every game object the script has been attached too. The most important scripts for this task are the agent scripts, described in 4.6.1.5. Using the tips from Unity provided in their github repository(29) for converting from object-oriented to data-oriented, the following strategy was devised for this part:

1. Identify the relevant scripts with an update function.
2. Create an ECS-system that implements the same behaviour as the one found in the update function of the original script.
3. Include the required script data through ECS-components

This scheme will provide with some advantages that are beneficial to the task in hand. The logic and data is separated, as the data is now in defined within ECS-components and the logic in systems, resulting in cleaner code. By having the required data given through components and not class objects, the engine is allowed to optimize their placement in memory. All ECS-components made with the iComponent interface in Unity will be tightly packed in chunks of data for effective iteration. However, the last step is not always possible in some cases for the DwarfHeim codebase. Parts of the engine such as the pathfinder utility uses a 3rd party asset for calculation of pathfinding, which is strictly done through objects. It is not possible to convert the required data in components without also creating a system for the calculation part, which is complex and requires more time than available for this thesis. For this reason, some monobehaviour scripts were only partially converted into a proper entity-component-system structure. The engine allows for iteration of normal objects in systems by referencing them through a component array. These objects will keep their data and methods instead of being split into entities and set of components. The data given in those arrays are reference types and thus not guaranteed to be linear in memory. This goes against data-oriented principles, but are a necessary trade-off at the current state. The objective is to reduce the number of such calls in the conversion stage.

With the old structure, the engine would go through each game object and activate their script update functions. Data would be updated as a part of the game object, and large data sets in an object would require more calls to the ram. With the new structure provided through the entity-component-system architecture, the systems will iterate through each set of data for an entity and update the values. The data sets of each system are tightly packed in memory as long as they are of the ECS-component type. This will in theory reduce iteration time as the processor will have higher grade of spatial locality. Small improvement in performance will still be possible for cases where only partial conversion is achieved with references to actual objects. The data and logic part will still be separated and the system will operate on many objects in batches through the component object arrays, allowing for batch-related optimization, such as setting common variables used by all objects before the iteration.

4.6.2.8 Converting the basic command chain

The basic command chain is a data type containing chain of basic commands for a unit to execute in the game. It is defined as a class containing an internal list, some fields related to the chain status along with methods related to the chain. The methods in the class are used for iteration of the chain, either backward or forward depending on the condition. The command agent within the engine is the only script that have direct access to the command chain in an unit. Converting the existing basic command chain into the new entity-component-system is not easily done due to the chain being represented as a list of basic commands. The list is non-blittable, non-fixed and is a list of basic commands objects. The limitations imposed on the ECS system makes it difficult to split the data into components. The internal list can not be represented by a component, which gives the conversion two possible alternatives:

1. Represent chain of commands in a different way that are not through objects.
2. Keep the internal list structure, and instead refer to it through objects in a system.

One possible way of doing the first alternative is creating a new entity for every new basic command sent to an unit. A chain consisting of basic commands can be split into multiple entities, where each entity represents one of the basic commands. Once a basic command has been executed, the entity can be deleted. The entity will consist of components representing the basic command data, the execution order and the unit it belongs to. This will require synchronization between the different commands in the new structure. To have a working chain, the basic command entities must have some way of knowing the previous and next basic command in the chain. The amount of overhead will be significant and possible require to large changes to the code base as

they all refer to a basic command chain. This alternative would truly be data-oriented, however lack of time and complexity makes it too difficult to start with. For this reason, the second alternative will be the first approach, and if given enough time, the first alternative will also be attempted.

Referencing the basic command chains through objects will go against the desire for spatial locality. Having too many systems referencing objects will affect performance negatively, so the number of systems using them should be kept at the minimum. There are in total three systems that have component data arrays consisting of objects that have access to the basic command chain. One system in the client-side will get access to the basic command chain through a command agent object reference, and a second system on the server-side that will get access through the ability agent. The third system is used to add basic commands sent from the server to the basic chains found in the clients. These two agents no longer function the same way, as most data and methods are stripped away in the new data-oriented approach. Instead, they are used as a way to access the basic command chains associated with an entity. This is done as a hybrid solution, where both a game object and entity exists. Other data related to the chain is no longer accessed through the agents, but instead through component data in the entity-component-system. Figure4.39 illustrates the conversion.

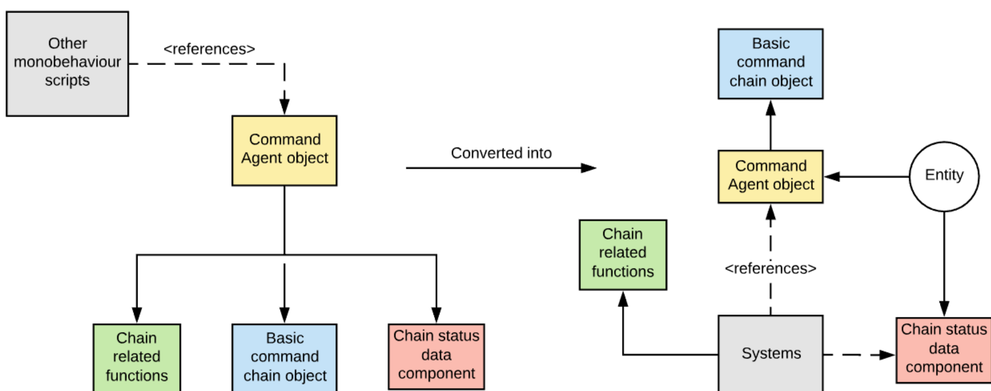


Figure 4.39: The change done to the basic command chain

4.6.2.9 Converting objects with execution logic - Basic actions and utilities

DwarfHeim uses traditional object-oriented design for execution of logic that can vary during run-time. The different type of basic actions and utilities are represented through parent classes with a number of virtual methods. When a game object is ready to execute a basic action or an utility, it will simply call the virtual execution method found in

the parent class. The type of logic found in the execution can vary based on the type of class it holds. The current way is polymorphic and modular as it is easy to just create a new class defining new behaviour for a new action or utility. A game object will retrieve the correct virtual method based on the current basic action or utility that it currently holds. Once something such as an unit requires new basic action, it will use the type id found within the basic command to extract the appropriate object instance from the prototype library. The object-oriented model with the prototype-based instancing gives a modular setting for creation of new functionality in an easy way. With the entity-component-system, it is not possible to use the same prototype based scheme with virtual functions without breaking some of the data-oriented design principles. A new approach is required where polymorphism isn't used extensively. The following strategy was devised for this part:

1. Take the execution logic found in these types of objects and move them into ECS-systems.

2. Create group of component types that are only used within these type of systems.

3. Create systems that manages the actions or utilities an entity should execute next.

The group of component types in step two are component types that are exclusively used by one system. When an entity is assigned one of these component types, it will be able to execute its current basic action or utility. Figure 4.40 shows the systems implemented for the basic actions and utilities. As the figure shows, each basic action and utility have a special type of parameter ECS-component that the system requires before executing its logic on the entity data. The parameter components are dynamically added and removed to an entity when it needs to perform one of the basic actions or utility. Step three is required for managing the type of basic action or utility system an entity should be updated by. It is important that one entity does not have multiple parameter ECS-components at the same time, otherwise several systems will execute their basic action or utility logic on the same entity at the same time, which should not be possible. In the old design, the game agent used to be responsible for switching to the next basic action. The agent has been changed into a system in the new design, with the task of creating and deleting the proper basic action for entities. The ability agent has also been changed into a system which handles the same with utilities. The utility execution will still be done in the server-side, so the desire for determinism is kept. Figure 4.41 shows the two new systems.

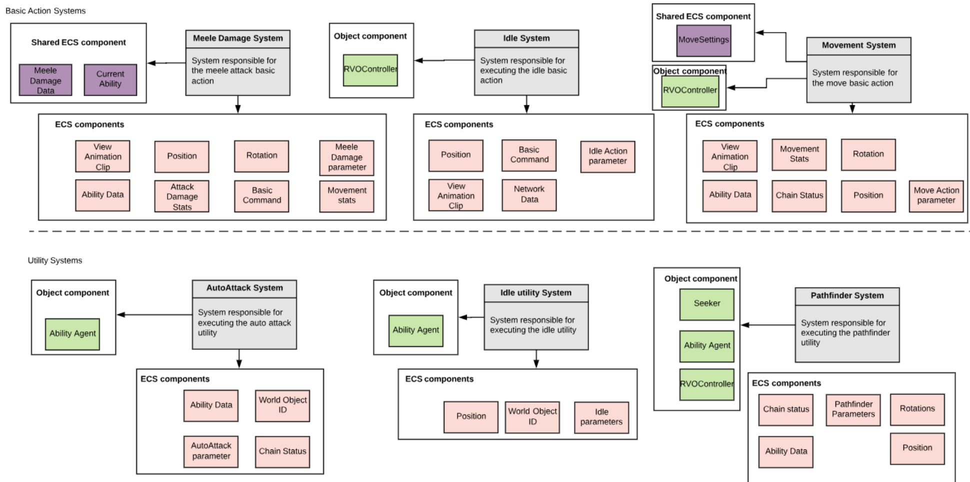


Figure 4.40: Overview of systems for basic actions and utilities

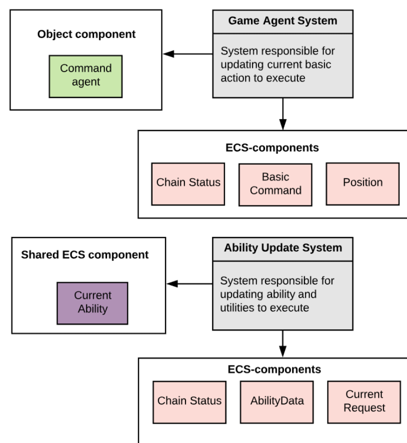


Figure 4.41: Game agent system and ability update system which handles proper execution of basic actions and utilities

4.6.2.10 Converting abilities

Abilities are just a set of utilities with a specific iteration order. Converting abilities requires some sacrifices to the current prototype based scheme. In the old design, ability data is dynamically extracted from prototype files. These files are created when the ability editor is used to create a new ability. The dynamic instancing is no longer done in the new design. Instead, abilities are defined in shared components, and as a result more hard-coded than the previous approach.

Shared components in the entity-component-system do not suffer the same limitations as regular components. They are allowed to have non-fixed arrays of any type and delegates. A specific ability can be defined through an instance of these shared components. A general ability shared component struct is created with two fields, one specifying the id of the ability and another for a delegate. The delegate represents pointer to a switch function that is unique for each instance of an ability. By defining different types of switch functions for the abilities, the shared components can be used to iterate through the correct ability. In addition to the shared ability component, each entity will have a ability data ECS-component that holds information about the current ability that they execute. The shared ability component holds information about the actual ability itself, while the ability ECS-component holds state information about the ability, which is different for each entity.

Using the shared ability component that is shared among many entities, in conjunction with the ability data ECS-component, allows for a functional ability system. Systems can correctly iterate the ability of entities by reading the current ability data and execute the switch function. The switch function takes in the current utility that an entity executes, and outputs the new utility that it should execute once the exit condition has been met. The ability update system is responsible for managing correct ability behaviour shown in figure 4.41.

This scheme moves away from the prototype-based ability structure, making it perhaps less modular in exchange for better iteration times overall for the complete design. However, changing the shared component values for an entity means that it has to be moved in memory, potentially affecting performance negatively if switched too often.

4.6.3 Making it more applicable on a server

The goal of the DwarfHeim conversion was to improve server efficiency. Up until this point, no consideration has been done for the game as a server. A server needs to run an instance of the matches being played client-side in order to synchronize between the players in a game. Additional tasks are required such as performing many of the same calculations needed in the clients in order to keep the same game state. Clients sending ability requests must also be processed. The server does not need to compute any graphic-related operation. This means that the visuals and animations can be removed and in theory improve performance. This is especially useful for the data-oriented solution, as all overhead associated with visuals and animations are object-oriented.

As previously described, the networking model currently uses a master client model, where one of the clients act as a server by being defined as a master client. The client declared as a master client will act as a host and perform server related operations. Whenever new functionality or other behaviour needs to be tested in DwarfHeim, two clients will be loaded with one acting as a master client. One challenge with this design is that there is no easy way to test the efficiency of the server without also having to modify the clients. For this reason, the clients will also have to be modified to act like the server. This means that when visuals and animations are removed, the clients will also need to remove them in order to test the server efficiency. However, the results collected from this scheme will still give an indication in the performance boost gained by the data-oriented solution, as long as both solutions implement the same changes to the server.

Two major changes are done to the clients in order to make them behave more like a server. The first change involves removal of the 3d-models and animations. The graphics could be completely removed for this part, however some sort of visual output are required during testing to verify correct behaviour. For this reason, a cube mesh is used to represent an unit. By removing the visuals and animations, the dependency for transform position component is also removed for the data-oriented solution. The second change involves removal of several game object components, such as the transform position. Removing this dependency means that there will no longer be need for the system that synchronizes between the two position components in an object, and as a result improve performance.

The removal of transform component requires some changes to the RVO controller, which is one of the scripts used for calculation of movement. The RVO controller is dependent on transform position component when calculating collision between units. The RVO controller will be changed to use position ECS-component through the entity manager instead of the transform position.

4.6.4 Testing

The new data-oriented design will be compared to the old object-oriented design written by Pineleaf studio. The same type of test will run multiple times for the two solutions, with minor variations in order to test how different parts affect the performance. The frame rate will be collected and used as a basis for how the two different solutions perform. Furthermore, the profiler will be used to analyze the time spent on the different operations for the solutions, to verify whether the new introduced systems are in

fact faster or not.

Since most of the changes done to the DwarfHeim game is related to the processing of basic commands through utilities and abilities, a test testing performance in these cases are needed. To compare the results between the two different solutions in regards to the efficiency of the client-server model, a test will be made which creates a large number of units moving in random directions. This test will give each unit a new move direction every specific time interval. Once a move command has been given from one unit on the client side, the server must create basic commands as response through the utility systems. The basic commands will then be transferred to the client units, allowing them to move in the direction and position given.

There will be small variations in the test to inspect different parts that can affect performance. The first test will use the 3d-models with animations available in DwarfHeim. The second test will not use the 3d-models or animations. This change will remove the need to have system for animation control in the new design. The game object components associated with an unit, such as position will still be kept so the physics calculations are the same for both solutions. A cube figure will be used to represent each individual unit when the dwarf models are removed. The final variation in the test will only apply to the dod-solution. In this test, several game object components associated with an unit will be removed. Not only will the animations and 3d-models be removed, but also all game object data associated with an unit that is not needed. Game object data for command and ability agents will still be kept in order to reference the basic command chain in an object-oriented manner. Everything else not related to the entity-component-system will be stripped away in the final test for the dod-solution.

The test for the oop-solution requires only one monobehaviour script. The script will instantiate a specific number of dwarf units on the scene. The script will take input parameters to decide whether the 3d-models and animations should be removed, along with other parameters deciding other test factors such as enabling collisions. The script will also consist of an update function that will activate every specific time interval given by one of the input parameters. The update function will iterate through each unit on the scene and give it a new move command. The dod-solution implemenents the same script for instantiating the units. However, the move commands are given through a system and not a monobehaviour script as the case with the oop-solution. The system will iterate through each entity representing an unit and give it a new move command. Figure 4.42 shows the input parameters for the instantiation of units. Figure 4.45 and 4.44 shows the test performed with different number of units and 3d-models.

The 3rd-party script profiler data exporter will be used to calculate cpu usage statistics, such as minimum, maximum and average time values for the different functions. The profiler does not output stream of more than 300 frames at a time, so only data for the last 300 frames will be gathered. Data of interest are those involved with the agents, basic action and utility executions. The results will only be gathered for 400 units spawned.

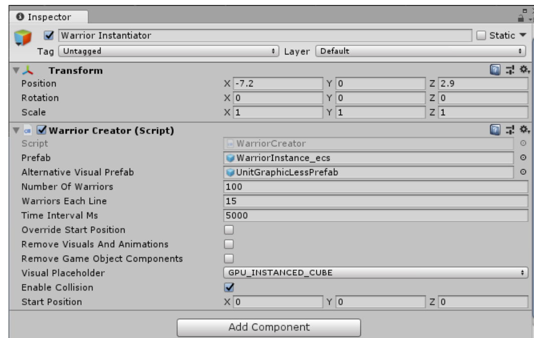


Figure 4.42: Test options for instancing units representing warriors

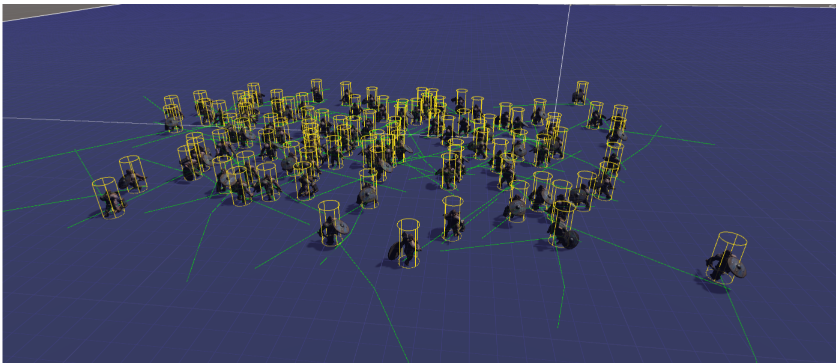


Figure 4.43: The moving units and their planned path

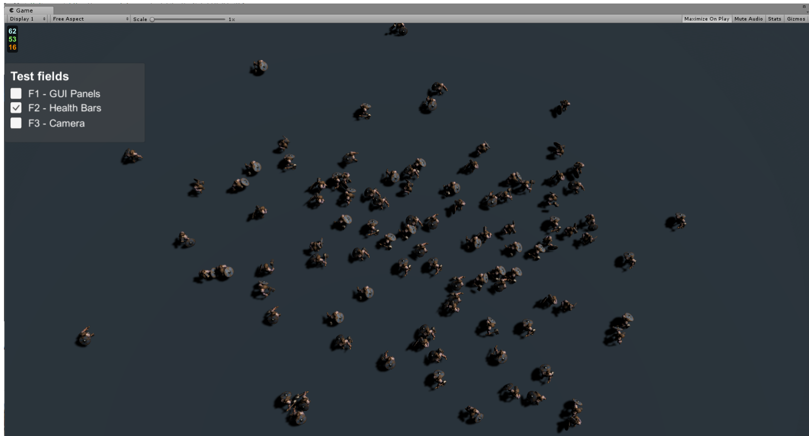


Figure 4.44: Test running with animations and dwarf 3d-models - 100 units

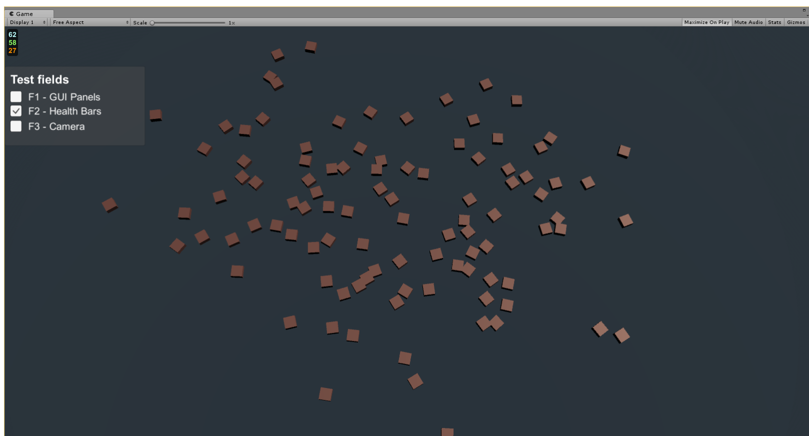


Figure 4.45: Test running without animations and dwarf 3d-models - 100 units

Chapter 5

Results

5.1 Entity-component-system implementation

Several tests were performed for the custom entity-component-system. All the tests were performed on the same computer with the same hardware specifications. Some of the tests compared the different versions of the implementation, verifying if the optimization steps actually worked. The other tests compared the implementation against traditional object-oriented design in order to compare the efficiency of using ECS.
The results of these tests will be presented in this section. Discussion of results will take place in next chapter.

5.1.1 Functional Test

Several tests for the ECS implementation were described in section 4.3.3. In total three tests were created which verified the points listed. The results of each test was written to a text file. If an error occurred, the console command would notify. Figure 5.1, 5.2 and 5.3 shows the results for test 1, 2 and 3 respectively for the final design.

5.1.2 Performance tests for the different versions of ECS

The results of the performance tests, described in section 4.3.4, will be presented here. The tests were completed with different number of entities created, ranging from 10 to 10,000,000. Figure 5.4 and figure 5.5 shows the results of the test for the two different versions. The graphs to the left shows the time elapsed based on the number of entities used. The second graph shows the ratio between the two different implementations, with regards to time elapsed. For the second graphs to the right, logarithmic scale is used.

```
Functional_test1 - Notisblokk
Fil  Rediger  Format  Vis  Hjelp
Creation And Deletion Test

 This test Tests that entities are created properly and that they are destroyed properly
 The test confirms the following requirements:
 * Entities are created with an unique ID
 * Entity are assigned component data
 * A system registers a component as part of its list of required components
 * Entity and its component data is deleted

System will now register a component type as part of its requirements
The name of the test system is: ECS.FunctionalTestSystem
The registered component is: ECS.TestComponent


20 entites are created with TestComponent component attached to it.
Legal entities of the system:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, -----


Following output shows all entities that are part of this componentStore of type ECS.TestComponent
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, -----

Entity with ID 1, 5 and 10 will now be destroyed

Legal entities of the system:
2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, -----
Following output shows all entities that are part of this componentStore of type ECS.TestComponent
2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, -----
---------------------
No errors were found in expected values. The test was a success.
```

Figure 5.1: Functional test 1 output results

```
Functional_test2 - Notisblokk
Fil  Rediger  Format  Vis  Hjelp
Memory transfer test

        The purpose of this test is to verify that a system successfully transfers updated data to the componentStores.
        The following conditions are tested:
         * A system class will successfully run its logic on component data
         * Component data will be updated to the componentStores
         * Retrieving componentdata through entityID with manager

The component type used is ECS.TestComponent which consists of one int and one float variable

Each Entity has the following initial component data:
Entity ID: 1, int_data: 0, float_data: 10
Entity ID: 2, int_data: 1, float_data: 11
Entity ID: 3, int_data: 2, float_data: 12
Entity ID: 4, int_data: 3, float_data: 13
Entity ID: 5, int_data: 4, float_data: 14
Entity ID: 6, int_data: 5, float_data: 15
Entity ID: 7, int_data: 6, float_data: 16
Entity ID: 8, int_data: 7, float_data: 17
Entity ID: 9, int_data: 8, float_data: 18
Entity ID: 10, int_data: 9, float_data: 19


System will now execute one update function on the component data.
Each Entity has following component data after system execution:
Entity ID: 1, int_data: 1, float_data: 30
Entity ID: 2, int_data: 2, float_data: 31
Entity ID: 3, int_data: 3, float_data: 32
Entity ID: 4, int_data: 4, float_data: 33
Entity ID: 5, int_data: 5, float_data: 34
Entity ID: 6, int_data: 6, float_data: 35
Entity ID: 7, int_data: 7, float_data: 36
Entity ID: 8, int_data: 8, float_data: 37
Entity ID: 9, int_data: 9, float_data: 38
Entity ID: 10, int_data: 10, float_data: 39
The test was a successfull with no errors.
```

Figure 5.2: Functional test 2 output results

5.1.3 OpenGL sine wave simulation tests

The results of the test described in **??** will be presented here. The different tests were ran with different number of objects. Not all tests were performed with the same number of objects, due to the refresh rate. Each bar graph figure shows test result for one specific number of objects created. The blue and red bars represent the dod and oop solution respectively. The ratio between the two solutions are also shown for each graph.

5.2 Sine wave simulation

The results for the sine-wave simulation tests from section 4.5 will be presented here. Both implementations ran the simulation with different number of objects spawned.

```
Functional_test3 - Notepad
File  Edit  Format  View  Help
Multiple System Test

        The purpose of this test is to verify that multiple systems can run together in the desired order
        The following conditions are tested:
        * A system class will successfully run its logic on component data
        * System logic will only be operated on data that is legal to that system
        * Multiple systems will run together


Two systems will be used for this test. A system of type ECS.ComponentIncrementerSystem with component ECS.TestIntFloatComponent and a system of type ECS.ComponentDecrementerSystem with
components ECS.TestIntFloatComponent and ECS.TestIntComponent will be used. The first system will increment the integer and float value of the component, while the second system will read
in this value and decrement an integer variable on the second component 12 entities will be created, where 10 of the entities have both components while two of the components only have the
component legal for system 1


The entities have the following data for each component before system update
Entity ID: 1, Component 1: int_data: 0, float_data: 10 - Component 2: int_data: 0
Entity ID: 2, Component 1: int_data: 2, float_data: 11 - Component 2: int_data: 4
Entity ID: 3, Component 1: int_data: 4, float_data: 12 - Component 2: int_data: 8
Entity ID: 4, Component 1: int_data: 6, float_data: 13 - Component 2: int_data: 12
Entity ID: 5, Component 1: int_data: 8, float_data: 14 - Component 2: int_data: 16
Entity ID: 6, Component 1: int_data: 10, float_data: 15 - Component 2: int_data: 20
Entity ID: 7, Component 1: int_data: 12, float_data: 16 - Component 2: int_data: 24
Entity ID: 8, Component 1: int_data: 14, float_data: 17 - Component 2: int_data: 28
Entity ID: 9, Component 1: int_data: 16, float_data: 18 - Component 2: int_data: 32
Entity ID: 10, Component 1: int_data: 18, float_data: 19 - Component 2: int_data: 36
Entity ID: 11, Component 1: int_data: 20, float_data: 20 - Component has no second component
Entity ID: 12, Component 1: int_data: 22, float_data: 21 - Component has no second component

The systems will now run twice each


The entities have the following data for each component after two system updates
Entity ID: 1, Component 1: int_data: 2, float_data: 50 - Component 2: int_data: -3
Entity ID: 2, Component 1: int_data: 4, float_data: 51 - Component 2: int_data: -3
Entity ID: 3, Component 1: int_data: 6, float_data: 52 - Component 2: int_data: -3
Entity ID: 4, Component 1: int_data: 8, float_data: 53 - Component 2: int_data: -3
Entity ID: 5, Component 1: int_data: 10, float_data: 54 - Component 2: int_data: -3
Entity ID: 6, Component 1: int_data: 12, float_data: 55 - Component 2: int_data: -3
Entity ID: 7, Component 1: int_data: 14, float_data: 56 - Component 2: int_data: -3
Entity ID: 8, Component 1: int_data: 16, float_data: 57 - Component 2: int_data: -3
Entity ID: 9, Component 1: int_data: 18, float_data: 58 - Component 2: int_data: -3
Entity ID: 10, Component 1: int_data: 20, float_data: 59 - Component 2: int_data: -3
Entity ID: 11, Component 1: int_data: 22, float_data: 60 - Component has no second component
Entity ID: 12, Component 1: int_data: 24, float_data: 61 - Component has no second component
The test was successfull with no errors. The values were as expected.
```

Figure 5.3: Functional test 3 output results

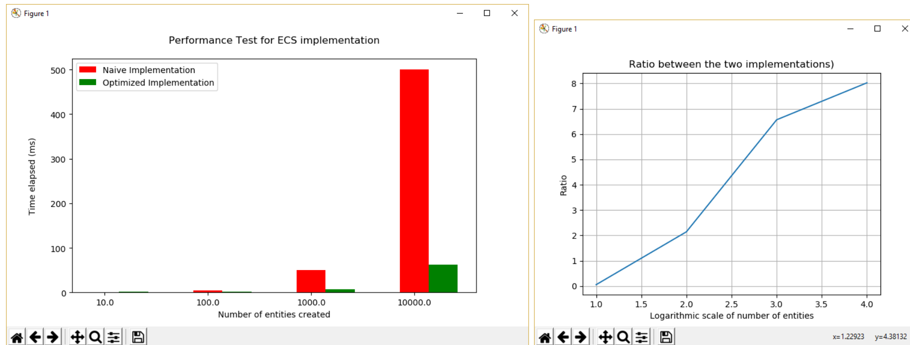


Figure 5.4: Performance test results: 10-10,000 entities

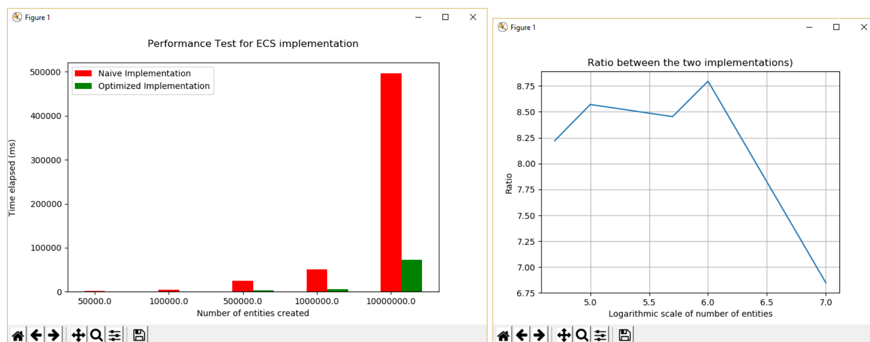


Figure 5.5: Performance test results: 500,000 - 10,000,000 entities

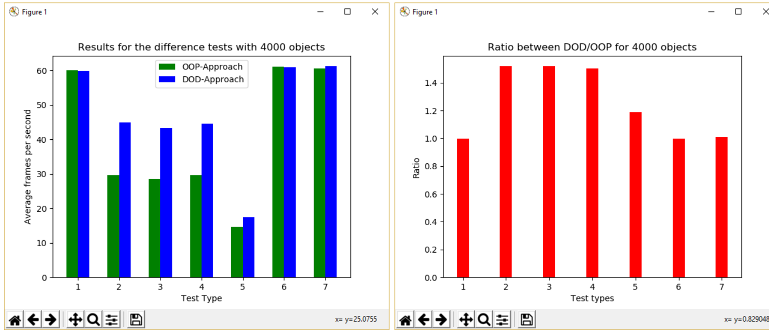


Figure 5.6: Opengl test results with 4000 objects

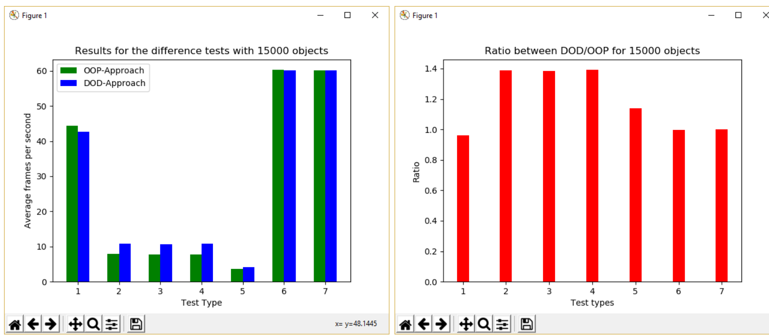


Figure 5.7: Opengl test results with 15,000 objects

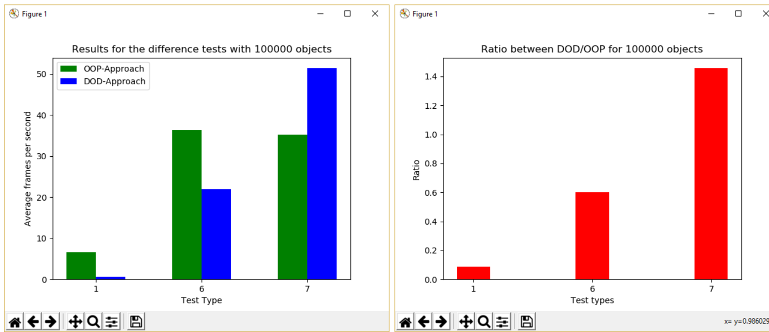


Figure 5.8: Opengl test results with 100,000 objects

5.2.1 Sine wave simulation results

Figure 5.9 shows the average frames per second achieved for both implementations with different number of points spawned. The green bargraph represents the objected-oriented solution, while the blue bar graph represents the data-oriented solution. The ratio between the two solutions are also presented in the figure. Furthermore, some

statistics from the profiler for CPU usage will be presented in the tables, collected with the profiler data exporter asset. The values show minimum, average and maximum values for some functions of interest, based on the last 300 frames. The percentage of total average time for a frame that the functions take will also be shown. Only data which is directly relevant to the design and important for the discussion is shown in the table. The original source for the values are available in the appendix along with other time values for other less important functions in the appendix B.7.2.

In addition, the data-oriented approach was tested with a higher number of objects in order too see how many object it could simulate with before breaking down to the same performance level as the object-oriented solution. The results are shown in figure 5.11.

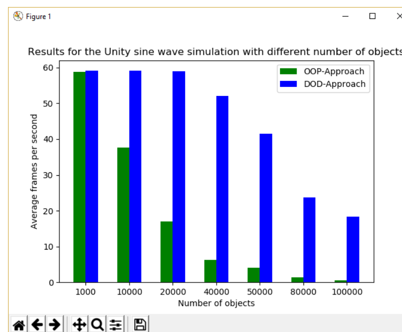


Figure 5.9: Simulation results in Unity

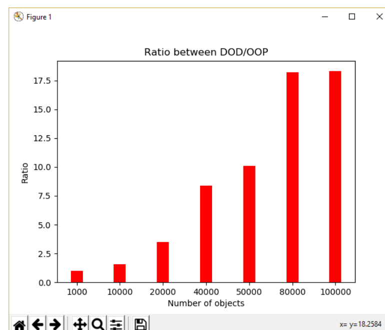


Figure 5.10: Simulation results in Unity - Ratio between DOD and OOP

5.3 DwarfHeim Conversion

This section will cover the results related to the DwarfHeim conversion.

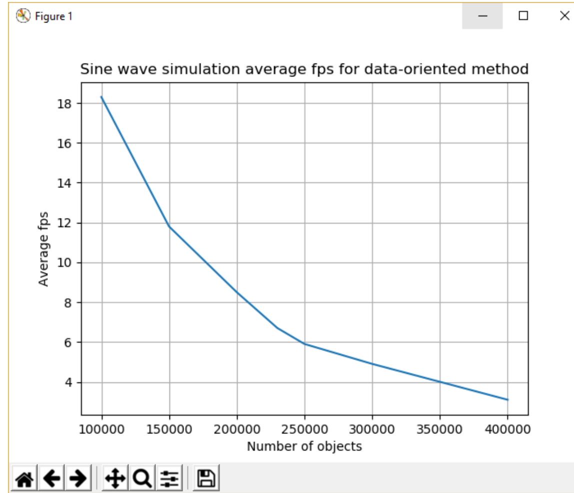


Figure 5.11: Simulation results for higher number of objects for data-oriented solution

Function type	Min time(ms)	Avg time (ms)	Max time(ms)	Total time %
PointMovementSystem	3.59	8.14	11.70	37.87
Gfx.WaitForPresent	0.89	13.14	23.73	47.61
MeshInstanceRenderer System	1.14	1.39	3.05	6.41
Render.OpaqueGeometry	0.12	0.17	0.34	0.76
TransformSystem	3.63	4.90	8.14	22.68

Table 5.1: Minimum, average and maximum time elapsed for several functions based on the last 300 frames for the dod sine-wave - Total time is based on average values

Function type	Min time(ms)	Avg time (ms)	Max time(ms)	Total time %
graph.Update()	15.63	16.22	22.68	12.15
Render.OpaqueGeometry	57.16	60.99	77.86	45.79
Gfx.WaitForPresent	0	0	0	0

Table 5.2: Minimum, average and maximum time elapsed for several functions based on the last 300 frames for the oop sine-wave - Total time is based on average values

5.3.1 Functional results

In total three basic actions were converted for DwarfHeim, the idle, move and meele attack actions. The 3d-models and animations were used for this part to correctly retain the same features in the original conversion. Figure 5.12 shows the successful implementation of movement, where a dwarf unit moves toward another unit. Figure 5.13

shows two units attacking another unit until its health reaches 0, unfortunately there is no animation for death implemented, nor is the health status shown. When the unit dies, the game object representing it will be destroyed and vanish immediately without death animation. The results are from the converted version where the entity-component-system was used in Unity.

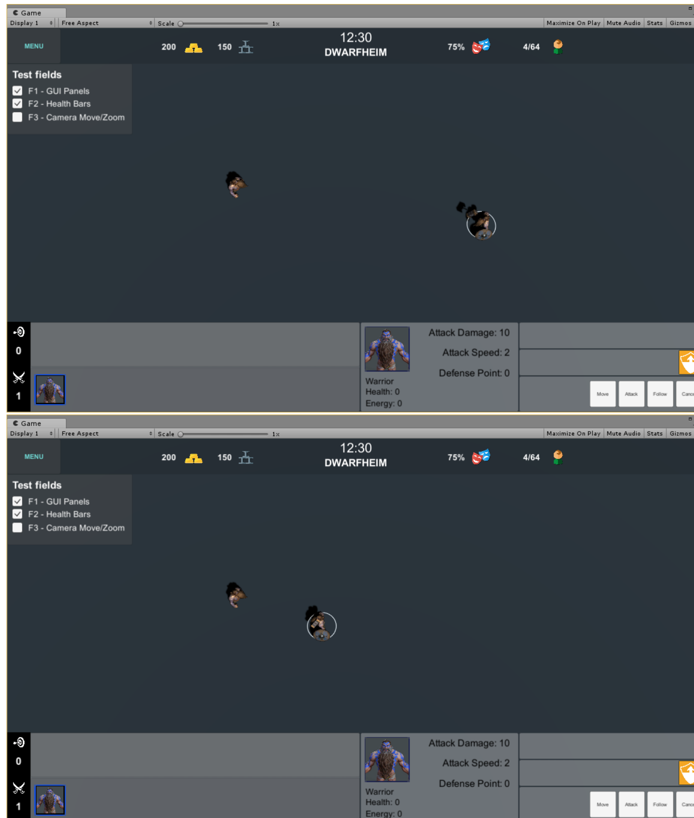


Figure 5.12: Unit walking towards another unit

5.3.2 Performance test

The results from the performance tests, described in section 4.6.4, are shown in figure 5.14, 5.15 and 5.16. The results are shown for both the original DwarfHeim codebase and the converted version. 5.14 shows the results for test where both animation and 3d-models were activated while 5.15 shows results for animation and dwarfheim 3d-models deactivated. Furthermore, 5.16 shows the results for the converted DwarfHeim version once additional object-oriented components were deactivated alongside the original version without 3d-models or animations. The ratio between the results are

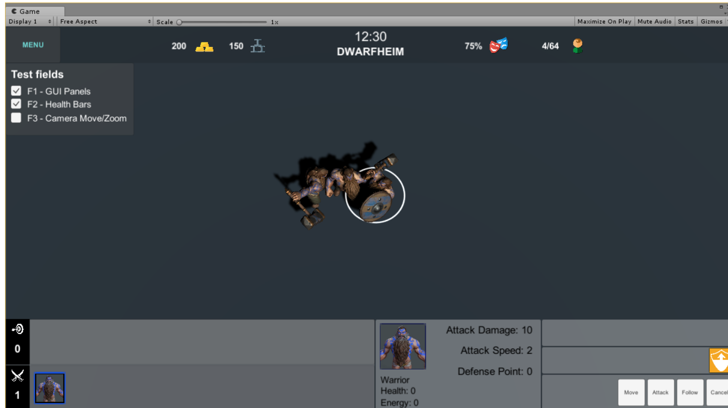


Figure 5.13: Two units attacking enemy unit

also shown for each figure.

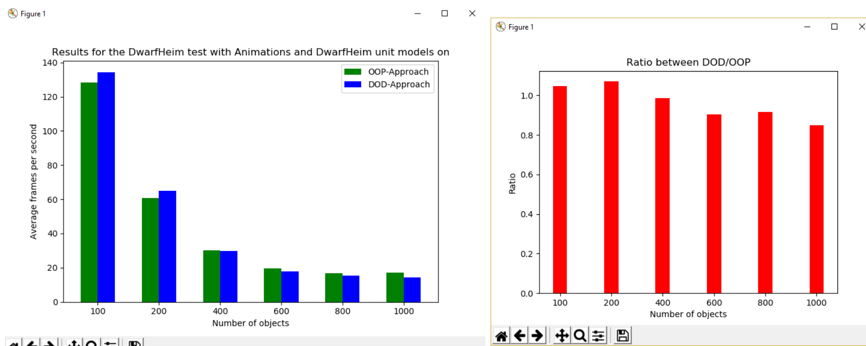


Figure 5.14: Results for DwarfHeim tests with animations and 3d-models on

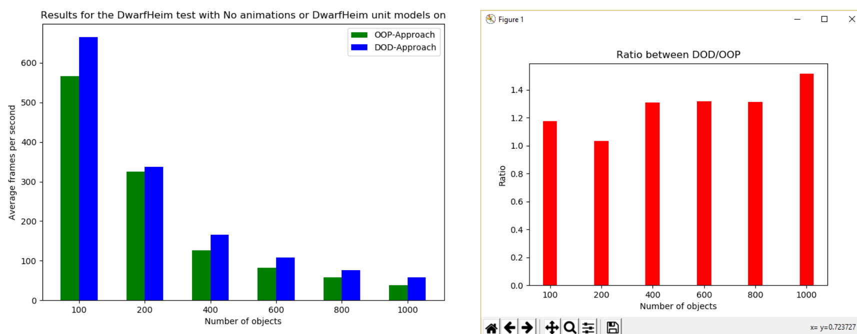


Figure 5.15: Results for DwarfHeim tests with animations and 3d-models off

Cpu usage times were inspected for all the variations in the tests, with 400 objects

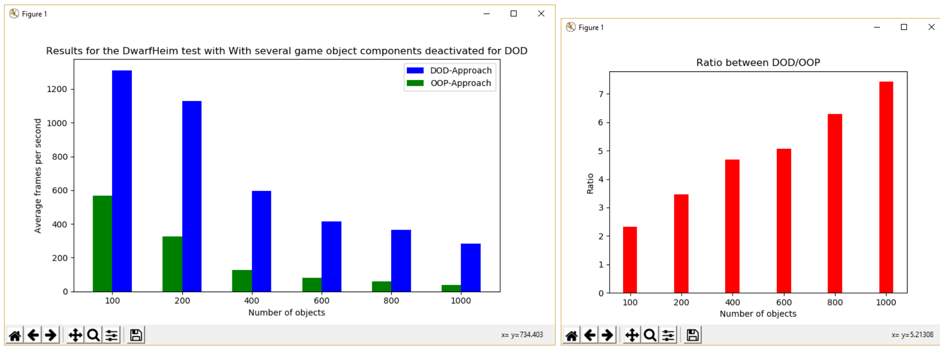


Figure 5.16: Results for the converted DwarfHeim with additional object-oriented components deactivated versus the original version without animations or 3d-models

spawned in each. The tables below shows the data gathered for the three tests with the profiler data exporter script. The only values of interest are the time elapsed for the different functions. The script is only able to extract data for the latest 300 frames. The original data representation with the data exporter is available in the appendix B.8

Function type	Min time (ms)	Avg time (ms)	Max time (ms)
ViewSynchronizationSystem	1.39	1.72	2.99
MoveSystem	0.4	0.51	0.88
GameAgentSystem	0.12	0.41	9.67
AbilityAgentSystem	0.09	0.29	1.05
BasicChainUpdateSystem	<=0.01	0.44	9.13
MovingTestSystem	<=0.01	0.25	12.21
PathfinderSystem	0.02	1.02	1.02
AnimationViewSystem	0.03	0.06	0.6
AbilityUpdateSystem	0.03	0.07	0.75
Sum		4.77	

Table 5.3: Results for type 1 DOD

Function type	Min time (ms)	Avg time (ms)	Max time (ms)
ViewSynchronizationSystem	1.37	1.56	2.52
MoveSystem	0.39	0.48	0.85
GameAgentSystem	0.14	0.22	2.41
AbilityAgentSystem	0.09	0.14	0.77
BasicChainUpdateSystem	0	0.08	1.67
MovingTestSystem	0	0.06	10.51
PathfinderSystem	0.01	0.02	0.23
AnimationViewSystem	0	0	0
AbilityUpdateSystem	0.03	0.04	0.51
Sum		2.6	

Table 5.4: Results for type 2 DOD

Function type	Min time (ms)	Avg time (ms)	Max time (ms)
ViewSynchronizationSystem	0	0	0
MoveSystem	0.35	0.47	0.9
GameAgentSystem	0.09	0.17	1.72
AbilityAgentSystem	0.05	0.11	0.87
BasicChainUpdateSystem	<=0.01	0.02	1.73
MovingTestSystem	<=0.01	0.04	11.33
PathfinderSystem	<=0.01	0.01	0.07
AnimationViewSystem	0	0	0
AbilityUpdateSystem	0.03	0.03	0.47
Sum		0.85	

Table 5.5: Results for type 3 DOD

Function type	Min time (ms)	Avg time (ms)	Max time (ms)
CommandAgent.Update()	0.04	0.07	0.12
GameAgent.Update()	2.03	2.63	7.50
AbilityAgent.Update	0.20	1.19	10.83
WarriorCreator.Update	<=0.01	0.33	13.44
Sum		4.22	

Table 5.6: Results for type 1 oop

Function type	Min time (ms)	Avg time (ms)	Max time (ms)
CommandAgent.Update()	0.03	0.07	0.23
GameAgent.Update()	2.02	2.42	4.41
AbilityAgent.Update	0.18	0.50	3.27
WarriorCreator.Update	<=0.01	0.10	10.37
Sum		3.09	

Table 5.7: Results for type 2 oop

Chapter 6

Discussion

The results will be discussed in this chapter.

6.1 Discussion of Results

Several tests and tools for analyzation was used for this thesis. The results will be discussed in this section.

6.1.1 Custom C# implementation of Entity-Component-System

Several tests were performed for the custom entity-component-system implementation in C#. The tests can be categorized into three objectives, testing for optimization in regards to implementation, testing for confirming correct functionality and behaviour, and testing for performance compared to object-oriented design.

6.1.1.1 Functional tests

The functional tests were created for one reason that happened to be beneficial during production, to verify that the intended behaviour of the design was correct. There were large changes in the code every time improvements were made. This caused all kind of changes in the different modules of the implementation, making it sometimes difficult to verify that the behaviour was correct. Due to this, functional tests were created. These tests were always performed with the same data input with some expected output values. This made it easy to make changes to the module and then verify that the intended behaviour was not corrupted during changes by running the test. In addition, these tests were helpful in verifying the specifications given in 3.2.1. The results shown

in section 5.1.1 showed the results for the final changes, showing that the intended behaviour was correct for the implementation.

6.1.1.2 Performance Tests

The performance test results showed promising gain with the improvements done to the ECS implementation. The final implementation was around 8,5 times faster than the original solution. When a system operates on a significant number of entities, retrieving data through the GetComponent api is expensive as the function will use reflection to retrieve correct id before using that value to retrieve the component store reference. Even though having all the component data stored in arrays for each system, the extra overhead was not significant enough to reduce overall performance, as the results showed. However, the results showed that the original implementation was more efficient for small number of entities. Thus, it would probably be beneficial to use the regular Get Entity data api for systems with small data set. Furthermore, the decrease in the ratio between the two solutions seen for 10,000,000 entities can be due to the memory transfer between the component data arrays in the two different systems. Every time a system is finished with its update function, the updated data will be stored to the component stores. With a large number of entities in the system, this overhead will be more costly, thus not having the same performance gap ratio.

One could also argue in favor of the inject attribute for making it easier for users to prepare data. Even though the results were positive, one must keep in mind that these tests were performed for only a single system. Having multiple systems would mean that more time would be spent on storing and loading values to the component arrays for each system. However, the results showed significant gap between time elapsed, meaning that it would in theory require a huge number of additional systems before the final design experiences worse performance.

Overall, the results of these tests strengthens one of the claims about data-oriented-design principles. Iteration is significant faster when the processed data is stored linearly in memory. Having the data linear in ram will cause less cache-misses and thus less memory fetches from the ram, increasing performance as the results showed.

6.1.1.3 OpenGL integration - Sine wave simulation

Several tests were performed for the ECS implementation once it was integrated with openGL. The test simulated a sine-wave graph consisting of large number of quadrilateral points in order to benchmark the application. Each of these tests were performed

twice, one with the ecs system and one with pure object-oriented method. The goal was to compare the results between the different approaches and see if having data-oriented principles were indeed more efficient or not. The results of the seven different tests were presented in section 6.1.1.3, and they will be discussed here.

For most of the tests, the data-oriented system was more efficient with a higher average fps than the objected-oriented counter-part. This strengthens the assumption that a linear data-layout is more efficient for the processor as the system will experience less cache misses. The data-oriented design performs better than the object-oriented design when the vertex buffer objects are only allocated once during startup, as oppose to allocating it every frame. Measuring the time between the different order of calls showed that allocation of vertex buffer objects were expensive, and thus had a larger impact on performance. This caused smaller performance gap between the data-oriented and object-oriented design for cases where the vertex buffer objects were allocated each frame. Furthermore, test 1 had no internal reference to other objects, which made it very fast for the object-oriented design. This gave similar performance between the two solutions for low number of objects. The other tests had a class structure for the object-oriented design, where the quad point class had references to a color object and mesh object. The increased number of references in the object-oriented design created a larger gap in performance between the two solutions, as test 2-5 shows.

One interesting result from the tests are the one given from test 1 and 6, which shows that the two solutions are approximately equal in performance for low number of objects, however with the object-oriented design slowly performing better as we increase the number of objects and entities. I assumed this was because of the manager always updating the component stores each frame for each system, which becomes quite costly as the number of entities increases. The object-oriented approach does not need to transfer memory between component data arrays in component stores, and as a result would perform better when the number of points increase. To further test this hypothesis, test 7 was created. In this test, the data-oriented approach only had one system for updating the positions, colors and renders of the points. This would remove the need to transfer memory between the different component data arrays in different systems. As the results showed, the data-oriented design was now faster and more efficient than the object-oriented solution. The results from this test shows that the linear memory layout is more efficient in the case where we only have one system with large number of entities. However, there comes a point where a large number of entities will have large enough overhead where the performance boost gained by the linear data-layout is not good enough to give an overall performance boost.

6.1.2 Limitations of the custom ECS implementation

The results from the different tests for the custom implementation strengthened the hypothesis that linear memory-layout is more efficient for a processor. However, this was only tested for a specific case. The ECS implementation is incomplete and not really tested to its fully extent when it comes to the types of applications one could make with it. It is still uncertain whether the custom implementation could create more efficient games, especially for small number of entities required. There is two things we can assume the tests performed for the custom design. First, using entity-component-system makes it easier to divide our application into clear domains. This is mostly based on personal preference and experience, however it is clearly easier to work when you separate data and logic into different domains. Second, the linear memory-layout is better for the processor, given that we do not have too much overhead on making it linear.

6.1.3 Meeting the specifications

The final design of the custom C# implementation satisfies all the specifications determined for the design. The design follows the entity-component-system pattern described in section 2.2.2. The functional tests proved the intended behaviour for the entity-component-system. Data and logic is separated in components and systems respectively, with entities being weak references to the data available. Furthermore, the use of component data arrays forces component data of same types to be stored linearly. The three different domains are decoupled and only accessed through a manager. The base class $LSystem$ gives access to easy interface for defining new logic in an application. The opengl integration provided the possibility of rendering graphics, making it possible to make games. At its current state, the design is advanced enough to be able to create games. An example of such a game demonstrating the capabilities were unfortunately not implemented due to time constraints.

6.1.4 Potential issues with the current design

The current implementation of the entity-component-system in C# is incomplete. Component data of same types are stored linearly together in memory, without any regards to the entities that own them. The memory allocation is not optimal for scenarios where multiple entities have the same set of data. In these cases, it would be more efficient for a system to iterate through the data based on entities instead of strictly component types, as identical data sets would imply that the entities represent the same type of "object".

Furthermore, the tests ran for this part were simple in its design and construction. An actual game was not constructed for the test implementation, but rather a simulation that only required large number of objects. The design might not necessarily be better for cases where the number of objects are low nor for games that requires many systems. These cases are yet to be tested for, but should be in the future if work is to continue for this design.

6.1.5 Sine-wave simulation results in Unity

The results from the sine-wave simulation done in Unity were shown in section 5.2. As the results demonstrate, the data-oriented solution performed better in every way. It managed to outperform the objected-oriented solution with a factor of over 15 once the number of objects spawned became significant. This shows that the entity-component-system in Unity is very optimized and able to waste less time on the same type of work, given that the data is laid out in a different way. Another interesting observation from the result shown in figure 5.11, is how much better the data-oriented design is in spawning larger number of objects. The object-oriented solution struggled with outputting more than a couple of frames per second at 40,000-60,000 objects, while the data-oriented solution did not go down to the same level of frame rate until it reached 400,000 objects.

6.1.5.1 Deviation in gpu rendering

The results shows that the entity-component-system in Unity is efficient, however it is uncertain whether it can all be attributed to memory layout. For this reason, the unity profiler was used for further inspection of cpu usage as explained in section 5.2. As the table for the dod-solution shows in 5.1, one of the most time consuming operations was the Gfx.WaitForPresent operation. This operation consumed as much as 47.61% of the total cpu time in an average frame. According to Unity, this operation is simply a stall in the processor as it waits for the gpu to finish rendering (30). This implies that the decrease in performance is related to the graphical processing unit and not the central processing unit. Based on this, it can be assumed that the processor would be able to perform even better if the performance wasn't bounded by the rendering time on the gpu.

On the other hand, the objected-oriented solution is not affected by a slower gpu. Inspection showed that the Render.OpaqueGeometry operation is the most time consuming operation of the post late update function, consuming a total of 45.79% cpu time as

shown in table 5.2. Interestingly enough, the same operation only requires 0.17 ms for the dod solution. The huge difference in the operations between the two solutions were surprising and further research were conducted in order to figure out the cause, as this operation is responsible for rendering the geometry of opaque elements.

After checking the unity frame debugger available in the profiler, which shows every draw call command sent to the gpu, it was discovered that the oo-solution sent 99 draw calls for this specific operation each frame, as oppose to the dod-solution which only sent 1 call. The profiler also stated that the draw calls were not batched since they used different meshes, which is not true in this case. All the cube objects have the same mesh and materials, as they are clones of each other. Unfortunately, there was no concrete answer to this question, other than the fact that the automatic gpu instancing in Unity is not completely reliable. The results from these numbers implies that the huge performance boost gained is not completely thanks to the data-oriented approach, but also the way unity handles the graphical rendering of objects. It might still be possible that the entity-component-system in Unity processes graphics in a more efficient method through the mesh instance renderer system which is responsible for drawing the meshes. The conclusion around these operations are not clear.

One possible theory for the discrepancy might be due to the way the two solutions represent a cube point. Every cube point is a game object in the oo-solution. This means that they all have transform position components that are updated and further used by the physics engine for calculation. There is no game object representing a cube point in the dod-solution as they are instead directly drawn by sending exact draw commands to the gpu. There is no associated game object with transform position for the physics engine to perform calculations with, as they are now represented in another way. This hypothesis was tested by disabling the physics simulation in Unity, however the results hardly changed.

6.1.5.2 Iteration time for the two solutions

Another interesting outcome of the tests are the time elapsed for the update functions that runs each frame for the two solutions. As the values in table 5.1 shows, the point movement system, responsible for updating position of each cube point, used 8.14 ms to complete its update function. On the other hand, the oo-solution used 16.22 ms to execute its update function as table 5.2 shows. This function performed the same task as the point movement system, with the exception of being object-oriented. The results here shows that the dod-solution were twice as fast at iterating through each cube point

and updating its position compared to the oo-solution. The results here indicates that the linear memory-layout is indeed more efficient for the processor. If the graphical rendering part and other overhead associated with the solutions were the exact same for both, then the implication would be that the dod-solution would be only twice as good at its best in average. This is certainly not the case in this scenario due to other operations playing a larger part than the update functions.

The efficiency of data-oriented design is even more highlighted when the minimum and maximum values are discussed. For the dod-solution, the minimum and maximum update time for the iteration was 3.59 and 11.70 ms respectively for the point movement system. On the other hand, the oo-solution had a minimum and maximum update time of 15.61 and 22.68 ms respectively. At the cases where minimum loop times were achieved, the dod-solution performed 4 times faster than the oo-solution.

6.1.5.3 Implications of the research

The data-oriented solution performed better with the ability to spawn a significant larger number of objects than the object-oriented solution. The results clearly showed that the dod-solution is better. Whether the performance boost gained by the data-oriented solution is only attributed to the data-oriented design, is unclear. The research conducted for the gpu rendering deviation was inconclusive. However, the dod-solution were still twice as fast at iterating through each point than the counter-part for an average frame. It is possible that the use of systems and clear separation between data and logic allows for a more efficient rendering system, handled by the engine. Two assumptions can be made from the research done here. First, the entity-component-system is more efficient than the typical object-oriented solution. Second, linear memory-layout made the cpu more efficient as updating the points on the graph were faster on the dod-solution. Both solutions were able to perform the same task, regardless of how the two solutions differed in how they executed different functions, the dod-solution were able to spawn more objects with better frame rate. In the end, that's the most important thing for playing games. At least for this specific case, it is safe to claim that the dod-solution performed better.

6.2 DwarfHeim conversion to a more data-oriented design

6.2.1 Functional features of the data-oriented design

The mix of data-oriented and object-oriented conversion proved that it would be possible to turn parts of DwarfHeim into a more data-oriented solution while still keeping the same features. This was shown by the results illustrating dwarf units with the ability to move and attack. The prototype-based library structure was not directly converted into a data-oriented solution, however it was used as an intermediate step for the entity-component-system to gather data from objects through that library. This shows the possibility of still using those types of features while still making the game data-oriented.

The main issue with going full data-oriented is the basic command chain,current animation and 3d-model system. The animation is controlled through an animator object which is tied to Unity's engine back-end. Time was not spent on improving this part, however it was later discovered that Unity had a data-oriented example application that showed the use of animations in a data-oriented system. The example was too complicated and involved exhaustive knowledge about animation and was thus ignored for this thesis. The 3d-models of dwarfs consists of a set of visual components that together define the complete model, which made it hard to represent through meshes in the game. The basic command chain could be converted into a pure data-oriented solution if enough time was given. Solving these problems would make it significant more easier to convert it into a data-oriented solution, as the game logic itself can easier be divided into systems and components.

While the results are promising when it comes to features, it must be noted that it was done only for a small part of the game. Large parts of the codebase were not touched nor inspected, making it uncertain whether the game is truly completely convertible. At minimum, the small scope showed promising outlook for the server-part of the game, which is one of the parts where the desire for cpu optimization is biggest due to costs.

6.2.2 Performance results

Three type of performance tests were performed for the data-oriented solution and two type of tests for the original object-oriented version.

6.2.2.1 Test with animation and 3d-models activated

Figure 5.14 showed the results for performance test for the case where both animation and 3d-models were activated. For lower number of objects, the data-oriented solution was performing slightly better until the number of objects spawned increased above 400, at which point the object-oriented solution started to perform better. A potential reason for this might be the increased overhead in the dod-solution where a system is used to synchronize position data between the entity-component position and the transform position. Table 5.3 shows time statistics for several systems, including the ViewSynchronizationSystem which has the task of synchronizing the positions. This system uses an average time, based on the last 300 frames before completion of test, 1.72 ms to complete. At its minimum and maximum it spends 1.39 2.99 ms respectively. This number increases as the number of objects increase. Furthermore, the system uses reference type references to the transform position component, which violates the the spatial locality property, explaining its inefficiency.

The results here shows that the dod-solution is either equal or slightly worse when many of the object-oriented dependencies are injected into the systems.

6.2.2.2 Test with animation and 3d-models deactivated

The dod-solution outperformed the oop-solution once the ViewSynchronizationSystem was deactivated and 3d-models were replaced with simple meshes. The additional overhead brought from the ViewSynchronizationSystem and animation system made the dod-solution perform better than the oop-solution with same features turned off. Both solutions saw a huge jump in frame rate once these parts were removed. This gives promising outlook for the server-side where these parts are not needed. At 1000 objects, the ratio between the dod and oop solution were slightly above 1.4, indicating a moderate boost in performance.

6.2.2.3 Test with dod-solution having additional object-oriented components deactivated

When transform position, animations and 3d-models were completely removed with all path movement calculation done through the 3rd-party path handler, performance improved significantly. The dod-solution had a frame rate seven times better versus the oop-solution for 1000 objects. Much of the gain in performance can be attributed to the reduction in physics due to the non existing transform position, which now stays static. The static transform position values causes less calculation for physics. Collisions and path movement are still taken care by the RVOController, so the extra physics applied

to the game is not really needed.

This version of the test is most applicable to a server, as unnecessary components are removed.

6.2.3 Inspecting time values for the converted parts

Evaluating how the converted parts of the game performs versus the original version is important to establish the efficiency of the solution. The tables in 5.3.2 presented the different time values for the functions associated with the agent, basic action and utility actions. These values will be further examined. In general, lower maximum values were found for the tests that performed with a higher frame rate. This can be attributed to the fact that the statistics calculated were based on the last 300 frames. The range of possible values will be smaller in the case of high frame rates since the 300 frames will cover in total a smaller time span.

6.2.3.1 Game agent conversion

The old game agent structure was split into the Game Agent System, which manages the type of basic action an entity should execute, and the group of systems representing basic actions. The basic action move system is the only relevant part here as the test only dealt with movement. In addition the animation view system overtook responsibility for setting animation through the view agent, which the game agent previously did. The total average time for the new systems representing the old game agent is 0.98, 0.7 and 0.64 ms for the first, second and third type of tests respectively. On the other hand, the total average time for the old game agent is 2.63 and 2.42 ms for first and second type of tests respectively. In average, the new converted version performs better. However, the game agent system has larger spikes in values than the other solution, going as high as 9.67. This spike does not happen often and is in rare cases where the system must change basic action parameter component for every entity in the system. In general, the new game agent system was able to perform better.

6.2.3.2 Command Agent conversion

The command agent did not perform a lot of work in the original version. It was used as a reference to the basic command chain, and thus most of the work was done through other scripts that accessed it. Due to this it is not easy to compare this agent against its respective system. Parts of the command agent was converted into the basic chain update system, which is responsible for updating the chain with new basic commands received from the server. This work is more time consuming in cases where significant

requests are received. This can be seen from the values given in the tables, with the system having spikes as high as 9.13 ms.

6.2.3.3 Ability Agent Conversion

The ability agent was split into the ability agent system, group of utility systems and the ability update system. For the tests performed, the pathfinder utility system were the only one used in addition to the idle utility system, which had negligible values. The total average time for these systems were 1.38 and 0.2 for the first two tests. On the other hand, the total average time for the original version was 1.19 and 0.5 ms. The oop-solution experienced larger spikes due to it having larger workload. The time results gathered are not sufficient to conclude whether the new ability agent system is better or not. There are still too many dependencies on object-oriented objects.

6.2.4 A hybrid solution vs pure data-oriented

As previously stated, the conversion was a combination of data-oriented and object-oriented principles due to complexity and time constraint. Too many systems were dependent on objects as the figures in section 4.6.2.3 illustrated. This dependency goes against spatial locality and increases cache misses. An ideal solution would have none of these object-oriented dependencies. Due to this, the current solution prevents the application from being truly tested in a real data-oriented setting.

The current solution does not truly compare a data-oriented application versus an object-oriented solution. Instead, a combination is tested. It is still possible to see improvements with the hybrid version as the changes can indicate better performance, as the results showed. However, these changes are difficult to interpret when the solution still have many object-oriented dependencies. The current solution prevents the application from being truly tested in a real data-oriented setting. The next step in the design would be to remove all object-oriented dependencies and convert the game into a pure data-oriented solution.

6.2.5 The implications of the research

The hybrid solution proved to be better in cases where it is made more applicable for a server, such as no animations or graphics. The results are promising and indicative of better cpu efficiency when the entity-component-system in Unity is utilized. At best case, the dod-solution managed to outperform the oop-solution with a factor of seven. The server costs could be significantly reduced if same grade of conversion were ap-

plied to the complete server-part of the game. The conversion can potentially be even better if the conversion were pure data-oriented.

The dod-solution didn't perform as well when animations and 3d-models representing dwarfs were used. A pure data-oriented solution with data-oriented animation system and graphic renders are necessary to conclude the efficiency of data-oriented solution for a complete game. The results from the other tests are indicative of better performance when the entity-component-system is properly used. While the results from the first tests were not positive for the dod-solution, it still showed remarkable promise when it comes to efficiency. This remains to be seen with a complete conversion.

Lack of time made it difficult to test other parts of the converted version, such as the meele attack ability. Nor was any other test implemented for the design than the movement one. This limited the scope of the test to the single instance of movement, which can cause results that are not strong enough to imply superiority with one design against the other.

It would also be advisable to test the new job system in Unity with the entity-component-system. Much of the work done in the conversion can be split into multiple threads. It would be beneficial to research the improvements gained by using this feature as data-oriented design is in theory better for parallelization.

6.3 General results

In general, most of the data-oriented tests achieved higher frame rate. The frame-rate was an indirect indication of processor efficiency, as higher frame rate meant the processor was able to perform more work in same time span. The frame-rate gave an overall look at performance without providing in-depth details about how the different processes in the applications performed. Rendering, scripting and physics calculation were performed better in most cases due to linear memory-layout. This was especially the case for the applications written in the Unity engine, as the engine optimized the rendering once the entity-component-system was used. A higher frame-rate meant that the graphics had smoother animation, better response times and able to output more work. The data-oriented solution performed better than the object-oriented counter-part based on the frame rate.

6.4 Developing with the entity-component-system

Most of the software developed in this thesis involved the entity-component-system. The architectural pattern proved to be effective when it came to writing software as the separation between data and logic allowed me to easily implement new functions. Throughout this thesis I experienced many positive things with the usability of the pattern that will be listed here.

- Focusing on data allows for better planning on structure.
- The separation between data and logic allows you to focus on each part separately.
- The pattern allows for easy implementation of systems that do specialized work.
- Adding new behaviour to a set of entities only involves creating a new system.
- Adding additional data to an entity and thus potential new behaviour, is easily done by attaching components.
- Data dependency is negligible as each component data is its own instance.
- Component data is reusable as it does not belong to any specific entity or instance, any "object" can use the type of data if required.

The positives from this pattern is especially beneficial for a video-game in development. With a pattern like this, a developer can easily define new set of component data and systems operating on them without touching other parts. Component data is just data, it is not related to any object or entity that it must adhere to, any new entity can choose to have it as part of its component set. This allows for good re-use of code.

There are however certain issues with the entity-component-system that can be challenging for developers.

- It is less intuitive to work without objects that encapsulate logic and data.
- It is not easy to implement polymorphic behaviour.
- It can sometimes be difficult keeping track of the type of data en entity has and how the data is used.

Overall, using entity-component-system with data-oriented principles can potentially create games that are more optimized. One important factor for video-games is visual fidelity, which is also impacted by the frame-rate. If the goal is to have better frame-rate, then data-oriented principles with entity-component-system is the method future

video-game developers should choose. It performs better than object-oriented design in exchange for less readability and polymorphism.

Chapter 7

Conclusion

Throughout this thesis, three applications have been implemented proving the efficiency of data-oriented design. The results indicate that developing applications with focus on memory-layout of data can improve the efficiency of processors. The use of entity-component-system allows for clear separation between data and logic, making it easier to develop applications that are data-oriented. The usability of the architectural pattern along with the results given from the tests indicates that this is a beneficial alternative for video-game development. The results also strengthened the use of data-oriented principles with the entity-component-system in Unity as the results illustrated performance boost by using a data-oriented approach. Even the incomplete DwarfHeim conversion performed better in certain cases, showing that the processor is indeed better when data layout is optimized for cache efficiency.

Although the results points toward data-oriented design, it is still not absolutely clear whether all performance gains are due to data-oriented principles. The applications that were used to verify the efficiency are dependent on external factors that affect performance as well, making it difficult to clearly assess that data-oriented design is better in every way. Nonetheless, further inspection of results showed that some of the improvements can be attributed to data-oriented principles.

In the end, many of the improvements due to data-oriented design increased the overall frame rate. For video-games, this is one of the most important factors. The improvements in frame rate gained through data-oriented design should motivate future developers to use this paradigm when developing games.

Chapter 8

Further Work

A list of propositions for further work will be presented in this chapter. the propositions are possible improvements based on my experience with the work done in this thesis.

8.1 Recommendations for the custom entity-component-system

The current custom entity-component-system is functional but not optimal when it comes to performance. This is especially the case for the memory-management done in the architecture, which is fairly simple. No game was developed using the custom entity-component-system due to time constraints, which could prove useful in analyzing a real-world scenario. The list of further work is inspired by the discussion given in section 6.1.1. The list of recommendations for further work are as follows:

- Better memory allocation based on entities and its component data instead of arranging strictly through component types.

- Making the API more user-friendly.

- Creating systems that handles graphical renders automatically through components.

- Developing a game using the architecture versus an object-oriented implementation to compare results.

8.2 Recommendations for the DwarfHeim conversion

The current conversion of DwarfHeim is not pure data-oriented, nor is the scope of the conversion significant. The next step in converting the game would involve steps in removing object-oriented dependencies. Additional changes can involve more complex changes such as converting the 3rd-party path movement controller. A full list of recommendations for further work are as follows:

- Convert the basic command chain structure into a pure data-oriented solution.
  - Each basic command in the chain could be turned into entities representing a basic command, as partly described in 4.6.2.8
  - This will involve overhead for synchronizing the correct order of basic commands for an entity.
  - Alternatively create a component with fixed size variables representing the chain.
- Convert the movement calculation part into a pure data-oriented solution.
  - This would involve taking all the monobehaviour components and turning them into systems as described in 4.6.2.7.
  - All data should be moved to components.
- Research methods for representing the 3d-models and animations strictly through systems.
  - Unity has provided source code for achieving this, however the codebase was to complex for this thesis.
- Turn the whole client-server model data-oriented.
  - Allow systems to handle all communication between clients and servers.
- Improve performance by using Unity's job system for multi-threaded programming
  - The job system has support for data-oriented principles

Bibliography

[1] C. Carvalho, "The gap between processor and memory speeds," 2002.

[2] "Finalizing objects, and memory concepts (stack versus heap)," http://archive.oreilly.com/oreillyschool/courses/csharp2/csharp214.html.

[3] "Pineleaf studio," http://pineleafstudio.com.

[4] L. T. Bojan Jovanović, Raphael M. Brum, "Mtj-based hybrid storage cells for normally-off and instant-on computing," 2015.

[5] "Locality of reference," https://en.wikipedia.org/wiki/Locality_of_reference.

[6] B. Nystrom, "Data locality," http://gameprogrammingpatterns.com/data-locality.html.

[7] J. Ante, "Unite austin 2017 - writing high performance c scripts," https://www.youtube.com/watch?v=tGmnZdY5Y-E.

[8] D. Davidović. (2014) What is data-oriented game engine design? Access date: 20-04-2018. [Online]. Available: https://gamedevelopment.tutsplus.com/articles/what-is-data-oriented-game-engine-design--cms-21052

[9] "New 2018 features for unity," https://unity3d.com/unity/features/job-system-ECS.

[10] M. P. Johansson, "Medium - composition over inheritance," https://medium.com/humans-create-software/composition-over-inheritance-cb6f88070205.

[11] "Composition over inheritance," https://en.wikipedia.org/wiki/Composition_over_inheritance.

[12] "Unity technologies," https://unity3d.com/company.

[13] U. Technologies, "Unity - scripting api:monobehaviour," https://docs.unity3d.com/ScriptReference/MonoBehaviour.html.

[14] "Unity profiler data exporter," https://github.com/steve3003/unity-profiler-data-exporter.

[15] "Introduction to the c language and the .net framework," https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework.

[16] J. Flick, "Frames per second - measuring performance," https://catlikecoding.com/unity/tutorials/frames-per-second/.

[17] S. Rombauts, "A small and easy c++ entity-component-system (ecs) library," https://github.com/SRombauts/ecs.

[18] "Opengl overview," https://www.opengl.org/about//.

[19] "Opengl," https://en.wikipedia.org/wiki/OpenGL.

[20] "Opengl language bindings," https://www.khronos.org/opengl/wiki/Language_bindings#C.23.

[21] "Opengl 4 for c/.net," https://github.com/giawa/opengl4csharp.

[22] "Opengl 4 for c/.net tutorials," https://github.com/giawa/opengl4tutorials.

[23] "The freglut project," http://freeglut.sourceforge.net/.

[24] "Rendering pipeline overview," https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview.

[25] "The model, view and projection matrices," http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/#the-model-view-and-projection-matrices.

[26] "Rgb color model," https://en.wikipedia.org/wiki/RGB_color_model.

[27] J. Flick, "Building a graph," https://catlikecoding.com/unity/tutorials/basics/building-a-graph/.

[28] G. Fiedler, "Floating point determinism," https://gafferongames.com/post/floating_point_determinism/.

[29] "Unity ecs - getting started," https://github.com/Unity-Technologies/EntityComponentSystemSamples/blob/master/Documentation/content/getting_started.md#getting-started.

[30] U. Technologies, "Diagnosing performance problems using profiler window," https://unity3d.com/learn/tutorials/temas/performance-optimization/diagnosing-performance-problems-using-profiler-window.

Appendix A

Acronyms

FPS Frames per second

OOP Object-oriented programming

DOD Data-oriented design

RAM Random access memory

ECS Entity-component-system

API Application programming interface

OO-solution Object-oriented solution

DOD-solution Data-oriented solution

Appendix B

Additional Information

Several topics that are relevant to the thesis will be presented here. These topics were not written on the thesis as they were either too detailed, too long or only had a minor role. However, they are supplemented here to give the reader a better understanding.

B.1 Concepts in Unity - Some additional concepts

Several concepts in Unity that were either considered too long or not as important to the thesis will be covered here.

B.1.0.1 Scenes

Scenes in Unity are the domain that contains the environments, menus and objects of a game. A single instance of a scene can be seen as an unique level. A game in unity can consist of several scenes, where each scene can have its own environment and closed domain of objects. Having multiple scenes allows us to design our games in multiple separated pieces. The position of objects and design of the level can be done directly in a scene through the scene window, allowing quick changes.

B.1.0.2 Camera and light sources

Camera and light sources are important parts for every scene found in Unity. The camera is used to display the scene found in the game world to the user. All the visual information is controlled by the use of cameras found in Unity. Each scene must have a minimum of one camera. By deciding how the camera is used, one can create menus, games in first-person or third-person perspective. The user will only see the visuals directed by the camera once a game starts, even though the scene contains a large amount

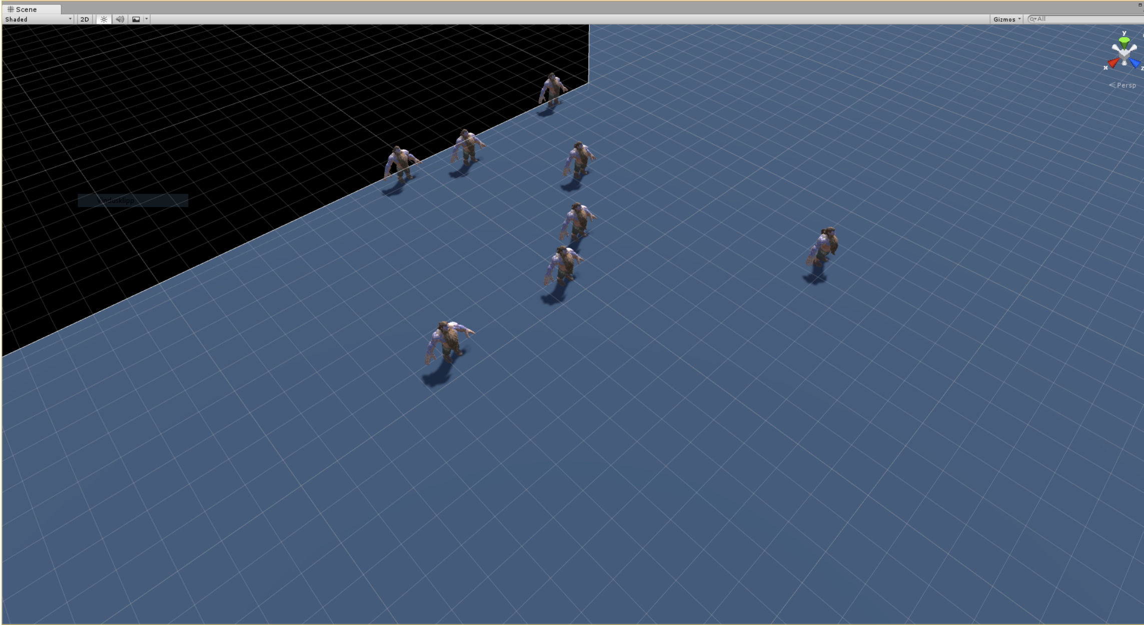


Figure B.1: Example of scene view in Unity

of objects. Light sources are gameobjects that display light on the scene. The camera captures the information from these sources to correctly represent the world. The colors and mood of the environment can be influenced by how the light sources are used.

B.1.1 Graphics in Unity

Unity uses three different components in order to render an objects surface in the game world, materials, textures and shaders. For rendering of the geometric shape of the object itself, meshes are used.

Meshes are a collection of vertices, edges and faces that defines the shapes of polyhedral objects in 3d. Polyhedral shapes are solids in three dimensions with flat polygonal faces, straight edges and sharp vertices. Triangles are usually used as the face in computer graphics. Geometric shapes can be created by combining a collection of polygons together. An example of this is shown in figure B.5 from wikipedia. Meshes can be seen as a collection of triangles that are linked together in 3d space, giving the impression of 3d shapes.
https://docs.unity3d.com/Manual/AnatomyofaMesh.html for mesh part In Unity, a mesh is represented by the mesh class. All the vertices of the mesh is stored in a single array.

Figure B.2: Direction of camera view on the scene(top) and the actual view the player sees(bottom)

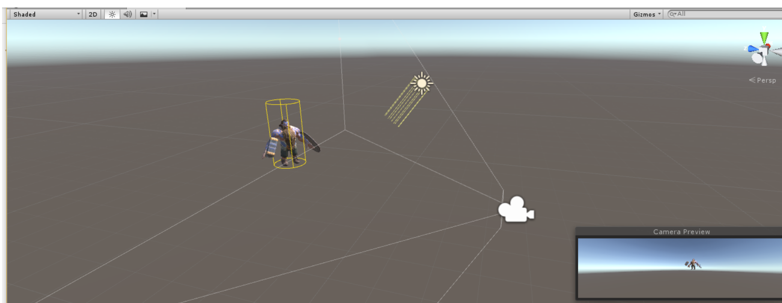


Figure B.3: Example of camera and light source object on the scene.

A single triangle is then defined by three of the vertices found in the vertex array. All the triangles in a mesh is defined in a single integer array. Three integers in a row define one single triangle, and the integers represent the indices found in the vertex array. For example, the three first elements in the triangle array defines the vertices for the first triangle, while the three next one defines the second triangle. Additionally, there are two more parameters for the mesh class that can decide the outcome of the final shape. A normal vector must be applied to each vertex in order to calculate correct lightning for the shape. The normal of the vertices are used to identify the direction of the light

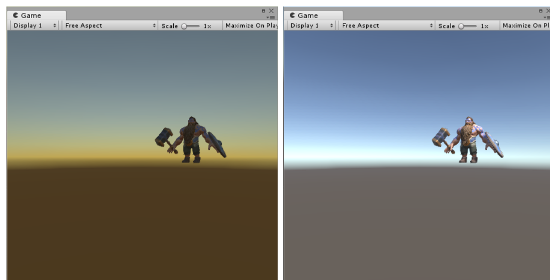


Figure B.4: Same camera view of gameobject with light source off(left) and on(right)

compared to the surface angle. Finally, an array of two-dimensional vectors are used to determine the fractional offset into a texture. This array is called for uv in the mesh class. Having a value of (0,0) means that the lower left corner of the texture is to be used for that specific vertice. This allows us to use different part of the textures for the different vertices. In order to render the meshes in Unity, you must first pass a reference to the mesh asset in a mesh filter component. The mesh filter will only hold a reference to the mesh asset and not render it graphically. The mesh filter must be passed further on to a mesh renderer. The mesh filter will pass on the mesh to the mesh renderer, which will then render the shape defined in the mesh class. An external modelling software

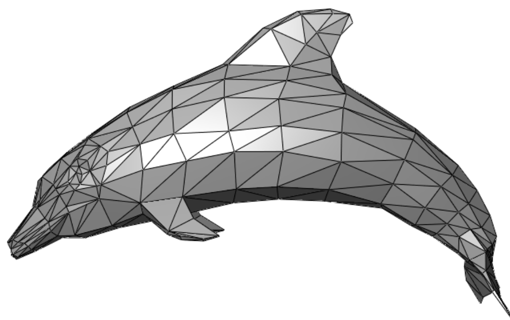


Figure B.5: Mesh of a dolphin, from wikipedia (1)

such as blender is usually used in order to create meshes. Unity does not offer a modelling plugin as part of its engine, however it is possible to create simple meshes in Unity with scripts.

Materials are the component part that defines how a surface should be be rendered in the game world. There are several references in the material data to different properties, such as the textures it uses, tiling information and color tint. Furthermore, the same material can have different options depending on the type of shader it uses.

Shaders are small scripts responsible for calculating the color of each pixel rendered.

The shader script itself contains the mathematical calculation required, following some shader algorithm such as the Lambertian reflectance B.6.0.6 that defines an ideal matte surface. The output from these scripts are dependent on the material configuration attached to it in addition to the lightning provided. Unity provides with a default shader called for standard shader with a comprehensive set of features and customization. A game developer does not necessarily need to know the mathematical intricacies behind the shading algorithms in order to utilise them.
Textures are image files usually stored as bitmaps. Textures are applied on the surface of graphical objects in order to give it finer detail. A shader algorithm can access references to textures through the materials and use this data to calculate the color of surfaces, giving different surfaces based on the textures used.

As described, the shaders are responsible for calculating the color of each pixel rendered. How it is done is dependent on different factors such as the materials. The materials contain metadata about how the surface should be rendered. The material can contain references to textures, which the shader can use to calculate the surface. These together gives us a way to define the surfaces of objects. A material specifies one type of shader to use in Unity. A material will have access to different options depending on the shader chosen. Furthermore, the shader specifies one or more textures to use for its calculations.

Given our knowledge above, we can create a simple 2d rectangular shape in Unity by using four different vertices. The shape consists of two triangles connected together to form a rectangular shape. Figure B.5 shows a schematic of the shape that I want to render in Unity. The mesh class in Unity needs three type of data in order to render this

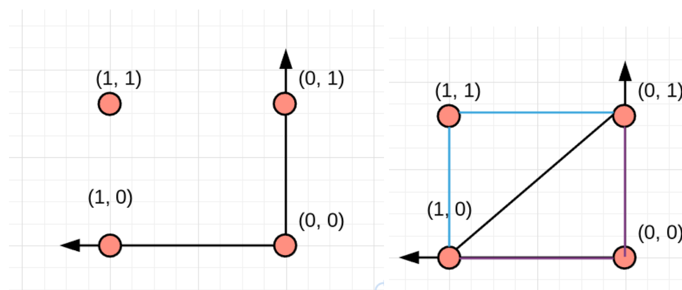


Figure B.6: Vertices of the quad mesh and the two triangles constructed

shape. It needs an array specifying all the vertices, an array that specifies the triangles and an array that specifies the normals to the vertices. The vertex array simply consists of the four points shown in figure B.5. The triangle array needs 6 elements in total,

three for each triangle in the shape. Following the coloring scheme in figure B.5, we can create our first triangle using the following vertices: (0, 0 ,0), (1, 0, 0) and (0, 1, 0). The second triangle can then include the following vertices: (1, 0, 0), (1, 1, 0) and (0, 1, 0). It is important that the indices in this triangle array correctly responds to the vertices found in the vertex array. Finally, the normal vectors for each vertex is simply pointing in the negative z-direction since we are creating a 2d shape. Using this information, we can create the arrays and then pass it to a mesh component in Unity. The code for this is shown below.

Listing B.1: Creating a new mesh representing a rectangular shape

```
1 // The vertex array containing vertices
2     private Vector3[] vertices =
3     {
4         new Vector3(0, 0, 0),
5         new Vector3(1, 0, 0),
6         new Vector3(0, 1, 0),
7         new Vector3(1, 1, 0)
8     };
9
10    // The normal array containing vertex normals
11    private Vector3[] normals =
12    {
13        new Vector3(0, 0, -1),
14        new Vector3(0, 0, -1),
15        new Vector3(0, 0, -1),
16        new Vector3(0, 0, -1),
17    };
18    // Triangle array representing triangles
19    private int[] triangles =
20    {
21        // First triangle
22        0, 1, 2,
23        // second triangle
24        1, 3, 2
25    };
26
27   // Use this for initialization
28   void Start ()
29   {
30       // Create a new mesh component
31       Mesh mesh = new Mesh();
32       // Add the new mesh to the mesh filter
33       GetComponent<MeshFilter>().mesh = mesh;
34       mesh.vertices = vertices;
35       mesh.triangles = triangles;
```

```
36        mesh.normals = normals;
37    }
```

The result of this code is shown in figure B.7. A default material was used for this render. The surface of this object was rather boring, so we can use another type of material. As explained, the material contains data for how the surface should be calculated, along with a algorithm found in a shader component attached to the material. We can add a texture map to our material and let the shader calculate colors based on that. A picture of hovedbygget was chosen as the texture. A new array was included in our script for the uv part, which decides how the image should be mapped on to the surface. By having the values (0, 0), (1, 0), (0, 1) and (1, 1) for the uv array, we simply tell the mesh that it should contain the whole range of pixels found in the texture. The new object is shown in figure B.8 after updating the materials.

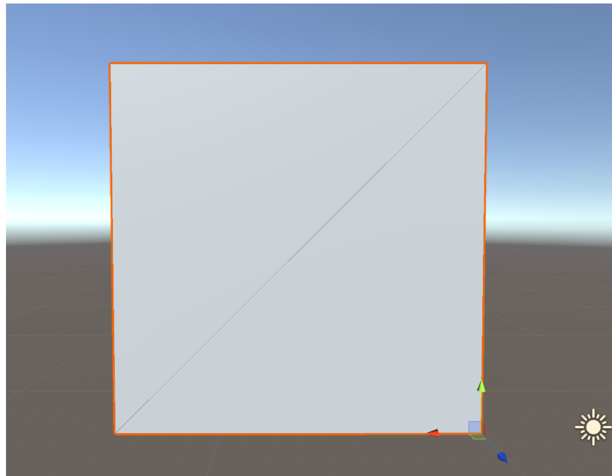


Figure B.7: The rectangular shape created by the script

B.2 Simple example of scripting in Unity

A simple example will be created in order to test the concepts explained in previous section. In this example, a cube will rotate around its axis by using scripts. First, a gameobject representing a cube must be created. Unity already offers a template for a cube. The cube is found by creating a 3d object of type cube already available in the unity editor. Figure B.9 shows the newly created gameobject and its default components. As one can see from the figure on the right, it is possible to add component to this gameobject. These components are the C# scripts previously explained.

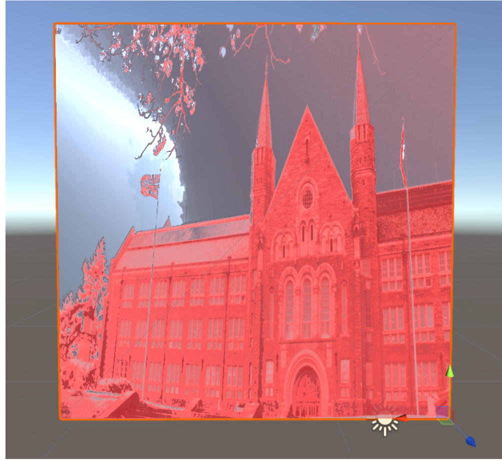


Figure B.8: Same rectangular shape with new material that includes a texture image of NTNU hovedbygget

For this example, the cube will rotate around its axis every frame. This can be achieved by using the monobehaviour.Update() function in a script. A new C# script is created by right-clicking on the project window and then selecting Create>C# script. Figure B.10 shows the newly created script in visual studio. The script is already derived from the monobehaviour class, with two of the event functions already defined. The Update() function will trigger each frame, so the rotation logic must be written there. The cube will only rotate around the y-axis for this example.. Unity has native support for rotation of gameobjects by using the transform api available. The transform of a gameobject is its position, orientation and scale on the world. Several methods for transform manipulation is available through this api. One specific function that is useful for this example is the transform.Rotate(float x, float y, float z) function that will rotate a gameobject along the x,y and z-axis based on the value given in euler degrees. This can be used together with the time api to rotate it with a given degree each frame. The Time api gives access to information such as the time elapsed since last frame. By using this in combination with transform.Rotate, the cube object can be rotated easily. The code is shown in figure B.11. Vector3.Up is simply a vector pointing up, which is the y-axis normalized. In addition, the value is multiplied with 50 to rotate it faster. Finally, in order to make the cube gameobject rotate, the newly created script component must be added to the cube object. It will then perform the behaviour defined in Update() each frame, making it rotate. This is also shown in figure B.11

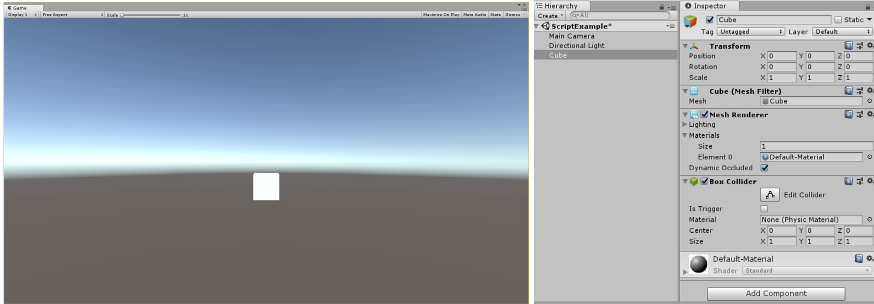


Figure B.9: Cube on the scene and its components

```
Assembly-CSharp                                   RotatingCube
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class RotatingCube : MonoBehaviour {
6
7       // Use this for initialization
8       void Start () {
9
10      }
11
12      // Update is called once per frame
13      void Update () {
14
15      }
16  }
17
```

Figure B.10: A newly created Monobehaviour script

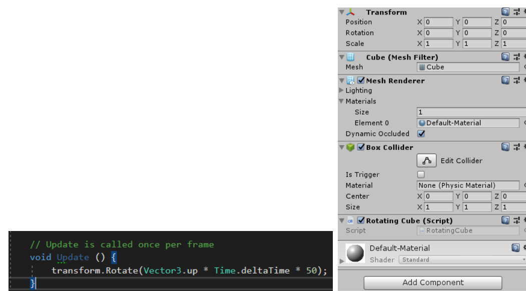


Figure B.11: Script code along with the component attached to a gameobject

B.3 ECS custom implementation

B.3.1 Example of system structure

In the example, a positionMovement system is used. This system has the main task of updating the position of objects. In order to transform the position of an object, the current position is required along with the velocities in each direction. For this reason, two components are required in our example. They are included as a part of the required component set of our system in figure B.12. All entities that have these two components are legal for the system and is to be included in the list of entities for the specific system. The example demonstrates two different entities that do fullfil require-

ment, however their attached components are not shown explicitly. Finally, the derived system class must override the abstract virtual function and define the behaviour. The following code snippet demonstrates how this could be achieved:

Listing B.2: System example

```
public override void OnUpdateEntity(Entity entity, float
    elapsedTime)
    {
        Position position = manager.GetEntityComponent<Position>(
            entity);
        Velocity velocity = manager.GetEntityComponent<Velocity>(
            entity);

       position.x = position.x + velocity.x * elapsedTime;
       position.y = position.y + velocity.y * elapsedTime;
       position.z = position.z + velocity.z * elapsedTime;
       manager.SetEntityComponent<Position>(entity, position);
    }
```

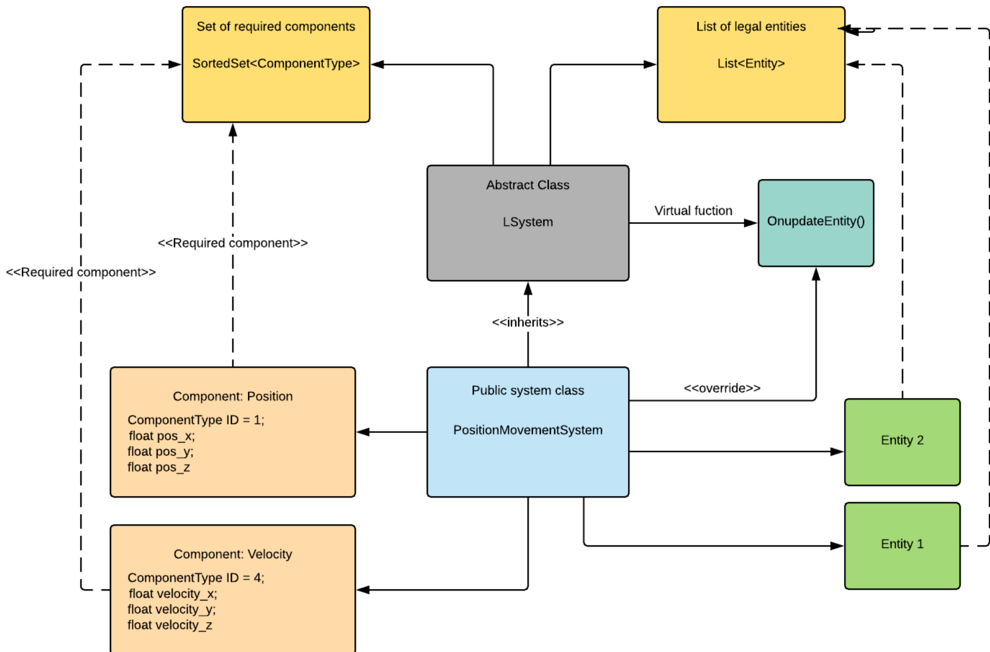


Figure B.12: Example demonstrating the structure of derived system classes

An entity's component value can be accessed and modified through the Manager.GetEntitycomponent<T and Manager.SetEntityComponent<T>() functions.

B.3.2 Example of inject-attribute with componentDataArray

The following code snippet shows an example of a system class using the inject attribute for componentDataArray.

Listing B.3: System example

```
class PositionMovementSystem : LSystem
    {
        public PositionMovementSystem(Manager manager)
        {
            this.manager = manager;
            AddSystemComponent<Position>();
            AddSystemComponent<ExistenceID>();
        }

        [Inject] public ComponentDataArray<Position> myPos;

        public override void OnUpdateEntity(Entity entity, float
            elapsedTime)
        {
            for (int i = 0; i < myPos.Count; i++)
            {
                Position positionius = myPos[i];
                positionius.x++;
                positionius.y++;
                myPos[i] = positionius;
            }
        }
    }
```

The [*Inject*] attribute found in line 10 will tell the application to fill it with data of type Position before first execution of OnUpdateEntity. Theres no need to activate any other function, this will be done internally as part of the manager class with the use of reflection. ComponentDataArray will only be filled with data from entities that are legal.

B.3.3 Optimization steps

Some optimization steps were performed for the custom entity-component-system implementation. Those were briefly referenced in the thesis. A more detailed description is given here for those interested.

B.3.4 Reducing number of boxing and unboxing

In C#, boxing is the process of converting a value type to the ultimate base class for all objects, Object, or any other interface type implemented by that value type. Unboxing is the opposite process, where an object or interface type is converted back to its original value type. When using reflection in C#, you only operate with objects. When a function that normally returns value type is used through reflection, it will be boxed and an object will instead be returned. In the current design, data is moved from ComponentDataArray to the component stores and vica versa through reflection. The data that is moved is of the type struct, meaning it is value type. Each component moved from one location is thus boxed or unboxed. According to the documentation provided by microsoft boxing and unboxing can be computationally expensive(2). If the application is to move large amount of component data, then it will require a large amount of boxing/unboxing, which will have a detrimental impact on performance. All data of value type is stored in the stack, however they are copied to the heap when boxed. When our application later needs to unbox the object, it needs to look after the data on the heap. Doing this for large number of component data means we have large amount of data on the heap, which then needs to be retrieved later to store on the other storage container. The performance loss due to boxing and unboxing large amount of objects are not desired and can easily be fixed. In the Manager::InitializeComponentArraysWithStruct() and Manager::UpdateComponentArraysWithStruct() function, an object reference to the internal data structures are obtained through reflection. Initially, each function used to invoke a method on each individual component data for transfer of data. This meant that every component was boxed and unboxed, impacting performance. Instead of unboxing each individual data component for the component stores, a new function was created that only unboxes the array of data, instead of each individual component. Once the whole array is unboxed once, all of its data are available to be accessed directly on the stack without having to unbox. A quick time test showed that this improved performance with around 30%(maybe show results of this).

B.3.5 Reducing number of function calls through another class

ComponentDataArray must invoke its insert method several times when data is transferred to it. If this process is done outside the class in another class method, the application must continously look up the object function once it is called. Instead of calling ComponentDataArray:insert() multiple times for each component data that is to be inserted, we can call it in the end once all the data is ready. A temporary array is instead used to fill all the data directly, before inserting the whole array into the componentdata array.

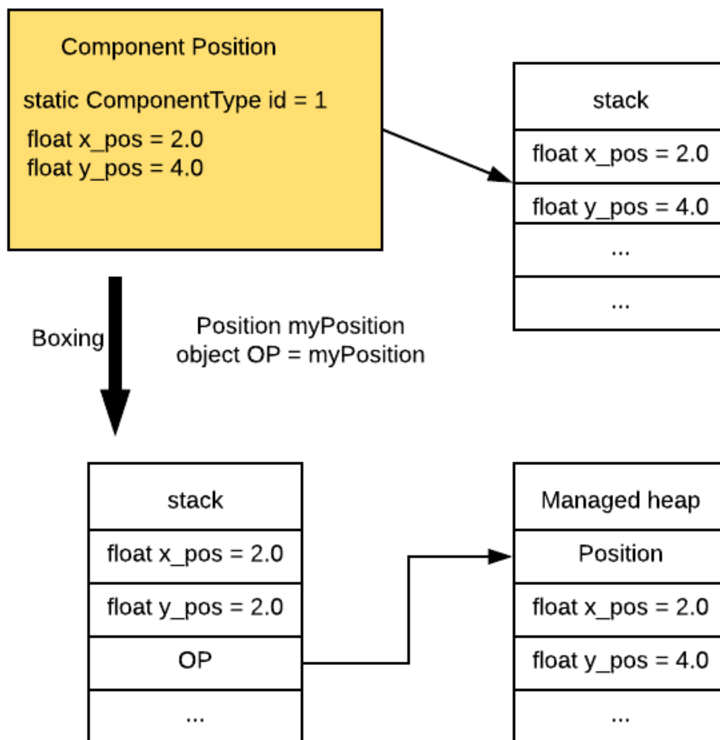


Figure B.13: Example of boxing process of component data - OP is a reference to component data on the heap

ComponentdataArray uses an internal array for storing data. The size of the array must be doubled whenever it is completely filled with data. This is done by creating a new array with double the size, with all the old data copied to it. If we are to move large amount of data to an array of this type, then it must double its size and copy values multiple times. In order to avoid this, we can initialize the component array with a large size to begin with.

B.4 OpenGl program code

B.4.1 Shader code

The code for the two shader program used for the custom entity-component system is presented here.

```
        public static string VertexShader = @"
in vec3 vertexPosition;
in vec3 vertexColor;

varying out vec3 color;

uniform mat4 projection_matrix;
uniform mat4 view_matrix;
uniform mat4 model_matrix;

void main(void)
{
    color = vertexColor;
    gl_Position = projection_matrix * view_matrix * model_matrix * vec4(vertexPosition, 1);
}
";

        public static string FragmentShader = @"
in vec3 color;

void main(void)
{
    gl_FragColor = vec4(color, 1);
}
";
```

Figure B.14: Vertex and fragment shader code

B.4.2 Code for drawing a simple triangle.

The code for drawing a triangle will be presented here.

```
// Initialize the opengl context
Glut.glutInit();
Glut.glutInitDisplayMode(Glut.GLUT_DOUBLE | Glut.GLUT_DEPTH);
Glut.glutInitWindowSize(width, height);
Glut.glutCreateWindow("Triangle with C# binding for opengl");

// Set the function that will run each frame.
Glut.glutIdleFunc(OnRenderFrame);
Glut.glutDisplayFunc(OnDisplay);

// Create new shader program.
program = new ShaderProgram(VertexShader, FragmentShader);

// Bind the program to the gpu.
program.Use();
// Set matrix values.
program["projection_matrix"].SetValue(Matrix4.CreatePerspectiveFieldOfView(0.45f, (float)width/height, 0.1f, 1000f));
program["view_matrix"].SetValue(Matrix4.LookAt(new Vector3(0, 0, 10), Vector3.Zero, Vector3.UnitY));
program["model_matrix"].SetValue(Matrix4.CreateTranslation(new Vector3(-0, 0, 0f)));
```

Figure B.15: Code for setting up the opengl context and shader program

B.5 Prototype-based programming

Prototype-based programming is a style of objected-oriented programming where newly created objects are "cloned" from generic existing objects that act as templates, called for prototypes.(3). The templates define behaviour and data that are common to all inherited objects. These objects can be further modified to have their own specific behaviour. This style is more dynamic in nature, as much of the class and object definitions happens during runtime. One of the key advantages with this style is that the

```
// Define the vertices of the triangle.
Vector3[] triangleVertices = new Vector3[3]
{
    new Vector3(-1, -1, 0), new Vector3(1, -1, 0), new Vector3(0, 1, 1)
};

// Define the indices of the triangle
int[] triangleIndexes = new int[3]
{
    0, 1, 2
};


triangle = new VBO<Vector3>(triangleVertices);

triangleElements = new VBO<int>(triangleIndexes, BufferTarget.ElementArrayBuffer);

triangleColor = new VBO<Vector3>(new Vector3[]
{
    new Vector3(1, 0, 0), new Vector3(1, 1, 0), new Vector3(1, 0, 1)
});

Glut.glutMainLoop();
```

Figure B.16: Code for setting up the triangle data in vertex buffer objects that will be sent to the gpu

```
private static void OnRenderFrame()
{
    // Clear information from last frame.
    Gl.Viewport(0, 0, width, height);
    Gl.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);



    // Set all triangle data to the correct buffers in the gpu.
    uint vertexPositionIndex = (uint)Gl.GetAttribLocation(program.ProgramID, "vertexPosition");
    Gl.EnableVertexAttribArray(vertexPositionIndex);
    Gl.BindBuffer(triangle);
    Gl.VertexAttribPointer(vertexPositionIndex, triangle.Size, triangle.PointerType, true, 12, IntPtr.Zero);
    Gl.BindBufferToShaderAttribute(triangleColor, program, "vertexColor");
    Gl.BindBuffer(triangleElements);

    // Draw the triangle
    Gl.DrawElements(BeginMode.Triangles, triangleElements.Count, DrawElementsType.UnsignedInt, IntPtr.Zero);

    Glut.glutSwapBuffers();
}
```

Figure B.17: Code that runs each frame, where vertex buffer object data is sent to the gpu along with a draw command

number of similar classes are reduced, as they can instead clone from a generic prototype.

B.6 Programming language C# and its features

B.6.0.1 Types and Assemblies in C#

C# is a strongly-typed language, meaning that each variable and constant must have a type. The use of the word type in this context means the entity that contains data about a specific type of variable or constant. The type contains information such as the storage required for the type, the minimum and maximum values it can represent, the base type it inherits from and the member that it contains. This means that a class is also a type which stores these type of information. In essence, the type of something

simply contains metadata about the type itself. The language provides with some basic types such as int, double and float. User-defined types can be created by defining new classes, structs, enums and interfaces.
Assemblies in C# is defined as a collection of types and resources that together forms a logical unit of functionality. They form the building blocks of the .NET framework applications. In essence, an assembly is a chunk of precompiled code that can be executed by the .NET environment. All types found in the .NET framework must exist in assemblies, this is because the common language runtime does not support types outside an assembly. Most applications built in visual studio is for instance stored as a single assembly. Assemblies are either compiled as .exe file or dynamic-link library, dll.

B.6.0.2 Value Types and Reference Types

All variables in C# can be divided into two set of types, value type and reference type. Variables that are of value types will directly contain values of the specified type(4). With other words, the value found on the address represented by the variable is the data of interest itself. Reference types do not directly contain the relevant values itself, but rather a reference to another location on the heap memory space, where the values are found(5). Reference types only store pointers to their respective types on the heap, while value types can be found in the stack or static fields. Figure **??**emonstrates how the different types are stored in memory. Typical standard data types such as integers, floats, booleans and structs are of value type. Examples of reference types are all class objects defined in C#. There are certain aspects one must be aware of when working

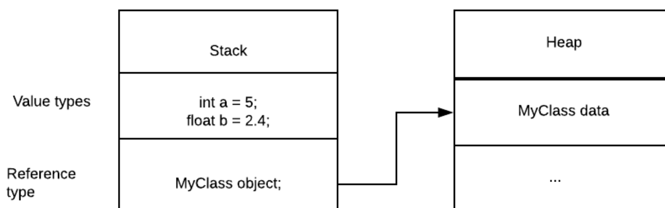


Figure B.18: Value types and reference types in memory

with these types, since they are stored different in memory. When passing value types as function arguments, the values will be copied into new variables for the function to use, the result of this is that the original variable will not be changed. For reference types, the pointer to the object itself will be passed by. This means that every change done within a function will affect the original reference type. Another interesting property is that an array of value types will contain the values in a linear contiguous memory layout.

This is not the case for reference types, instead the array will contain pointers stored in the same linear contiguous way. However the difference here is that the pointers will point to different locations on the memory heap once accessed. This is an important property to know for data-oriented principles, which will be further discussed.

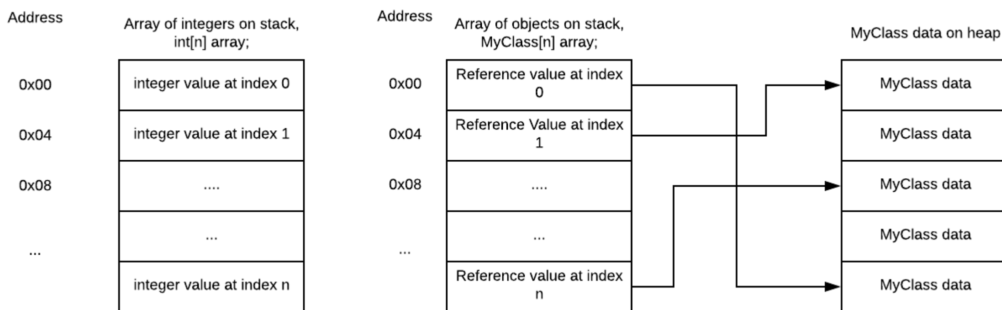


Figure B.19: Array of value type vs array of reference type

B.6.0.3 Interfaces

Interfaces in C# represent a group of functions that classes or structs must implement if they inherit it. Interfaces are important for including behaviour from different sources for a class or struct since C# does not support inheritance from several parent classes. In addition, structs can simulate inheritance-like behaviour by including interfaces since they are not allowed to inherit from other classes or structs.

B.6.0.4 Delegates

Delegates are similar to function pointers found in C and C++. It is a type that holds references to methods with a particular parameter list and return type. Delegates allows different objects to customize their behaviour based on what kind of method a delegate holds. They are also used for events in C#.

B.6.0.5 Events

Events implement the observer pattern, which is a software design pattern. Multiple different objects can "subscribe" to an event with a given method. Once the event has been activated, each subscribing object will call their registered event function. Events can thus be used to signal to other dependent objects that an action has occured.

The type of a variable or constant contains information The type specifier in C# stores information about the type which the

B.6.0.6 Reflection

The last topic of interest four our chosen language is reflection. Reflection gives access to objects of the base type Type, which describes assemblies, modules and types. Having access to this information in runtime allows us to dynamically create instances of a specific type, bind types to an existing object or get type from existing objects and invoke the methods available in that type. Attributes are accessible through reflection, allowing you to read data associated with the attributes or invoke attribute related methods. Using reflection together with custom attributes allows us to modify data with ease by simply adding the attribute to the data field or class. Furthermore, it is possible to create complete new types at runtime by using reflection. Understanding reflection is important for using Unity as the engine has an extensive use of custom attributes.
An example will demonstrate the power of reflection and attributes. Let's say we want to keep track of how often fields of a specific type are used in classes. Say that every time we attach our custom attribute, FieldCounter, to a class field, it counts that type. We can create a simple application that will iterate through the assembly and look at every class that has this attribute. We can then count the number of times the attribute occurs for a type. By using a dictionary, we can keep track of occurrences for a specific type. The custom attribute will only be used as a marker, specifying that this field is of interest. We can then have a function that prints out this information for us. The below code snippet shows an example of this function. In this example, our customattribute is named FieldCounterAttribute.

Listing B.4: Using reflection to access fields with our customattribute

```
1  static void PrintFieldsOfInterest()
2          {
3              // Dictionary containing the type and number of
                   instances found in classes.
4              Dictionary<Type, int> FieldCounts = new Dictionary<
                   Type, int>();
5
6              // Get the current executing assembly
7              Assembly myAssembly = Assembly.GetExecutingAssembly();
8
9              foreach (Type type in myAssembly.GetTypes())
10             {
11                 // Only look for class types
12                 if (type.IsClass)
13                 {
14                     // Check if a field has the Fieldcounter
                           attribute
15                     var fieldInfos =
```

```
16                              type.GetFields().Where(field => field.
                                    IsDefined(typeof(FieldCounterAttribute)
                                    ));
17
18                          foreach (var fieldInfo in fieldInfos)
19                          {
20                              var fieldType = fieldInfo.FieldType;
21                              if (FieldCounts.ContainsKey(fieldType))
22                              {
23                                  FieldCounts[fieldType]++;
24                              }
25                              else
26                              {
27                                  FieldCounts.Add(fieldType, 1);
28                              }
29                          }
30                      }
31                  }
32                  Console.WriteLine("The following types were attached
                        with the custom attribute: ");
33                  foreach (var types in FieldCounts.Keys)
34                  {
35                      Console.WriteLine($"Type: {types}, number of
                            occurrences: {FieldCounts[types]}");
36                  }
37              }
```

The function will print out every type that has the FieldCounterAttribute applied to it, along with the number of times the attribute is applied to the same type. A complete example is demonstranted in appendix .

B.7 Sine-wave simulation in Unity

B.7.1 Accessing mesh and material data with entity-component-system

The folllowing code snippet demonstrates how data for settings and materials were retrieved. The way references are retrieved are shown in line 21-28.

Listing B.5: Getting mesh and setting data

```
1 public sealed class Graph{
2     /// <summary>
3     /// Archetype for the point, specifying kind of components
          attached to it
4     /// </summary>
5     public static EntityArchetype pointArcheType;
```

```
6
7       /// <summary>
8       /// Settings for the graph
9       /// </summary>
10      public static Settings graphSettings;
11
12      public int numberOfPoints;
13
14      public static MeshInstanceRenderer cubeLook;
15
16      // Run this function after scene has been loaded.
17      [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.
            AfterSceneLoad)]
18      public static void InitializeWithScene()
19          {
20              // Name of gameobject representing cube point on the
                    scene is PointRenderPrototype
21              var pointProtoType = GameObject.Find("
                    PointRenderProtype");
22              cubeLook = pointProtoType.GetComponent<
                    MeshInstanceRendererComponent>().Value;
23              Object.Destroy(pointProtoType);
24
25              // Retrieve setting data from settings gameobject on
                    scene
26              var settings = GameObject.Find("Settings");
27              graphSettings = settings.GetComponent<Settings>();
28              Object.Destroy(settings);
29
30              NewGame();
31          }
```

The GameObject.Find(string name) function will search for the game object on the current active scene.

B.7.2 Sine-wave simulation, profiler stats with profiler data exporter

A number of figures showing the cpu usage stats from unity profiler for the sine-wave simulation tests done for 50,000 objects.

B.8 Unity profiler data exporter results for DwarfHeim

Tables displaying cpu usage times were shown in the results section. These values were gathered from the statistics shown with the profile data exporter. The original data representation is shown here, as the result section only displayed values of interest.

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	11.10	0.00	1.00	0 b	3.63	0.00
ExecuteJobFunction.Invoke()	11.10	11.10	1.00	0 b	3.63	3.63
PointMovementSystem	20.20	0.00	1.00	0 b	3.59	0.00
PostLateUpdate.FinishFrameRendering	8.30	0.00	1.00	0 b	1.41	0.01
MeshInstanceRendererSystem	3.30	0.00	1.00	0 b	1.14	0.00
Unaccounted time between: PostLateUpdate.ProfilerEndFrame and EndFrameBarrier	3.20	3.20	1.00	0 b	1.04	1.04
Gfx.WaitForPresent	5.20	5.20	1.00	0 b	0.89	0.89
Camera.Render	0.90	0.00	1.00	0 b	0.32	0.02
Drawing	0.40	0.00	1.00	0 b	0.15	0.00
Render.OpaqueGeometry	0.30	0.00	1.00	0 b	0.12	0.00
PostLateUpdate.ProfilerEndFrame	0.30	0.00	1.00	0 b	0.10	0.00
Profiler.CollectGlobalStats	0.30	0.20	1.00	0 b	0.10	0.07
RenderForwardOpaque.Render	0.20	0.00	1.00	0 b	0.09	0.01
Shadows.RenderShadowMap	0.10	0.00	1.00	0 b	0.05	0.02
Culling	0.10	0.00	1.00	0 b	0.04	0.01
CullResults.CreateSharedRendererScene	0.00	0.00	1.00	0 b	0.03	0.02
TransformSystem	0.10	0.00	1.00	0 b	0.03	0.00
Shadows.PrepareJob	0.00	0.00	1.00	0 b	0.03	0.00
SceneCulling	0.00	0.00	1.00	0 b	0.02	0.00
UpdateDepthTexture	0.00	0.00	1.00	0 b	0.02	0.02
UIEvents.CanvasManagerRenderOverlays	0.00	0.00	1.00	0 b	0.02	0.00
UGUI.Rendering.RenderOverlays	0.00	0.00	1.00	0 b	0.02	0.00
RenderForwardOpaque.CollectShadows	0.00	0.00	1.00	0 b	0.02	0.00
Initialization.PlayerUpdateTime	0.00	0.00	1.00	0 b	0.01	0.01
CullAllVisibleLights	0.00	0.00	1.00	0 b	0.01	0.00
Graphics.Blit	0.00	0.00	1.00	0 b	0.01	0.01
Canvas.RenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
Shadows.ExtractCasters	0.00	0.00	1.00	0 b	0.01	0.01
Profiler.CollectMemoryAllocationStats	0.00	0.00	1.00	0 b	0.01	0.01
Profiler.CollectAudioStats	0.00	0.00	1.00	0 b	0.01	0.00
Update.ScriptRunBehaviourUpdate	0.00	0.00	1.00	0 b	0.01	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.20: Profiler data stats for dod - Minimum values

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	50.00	0.04	1.00	0 b	13.66	0.01
Gfx.WaitForPresent	47.61	47.61	1.00	0 b	13.14	13.14
PointMovementSystem	37.87	0.00	1.00	0 b	8.14	0.00
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	22.68	0.00	1.09	0 b	4.90	0.00
ExecuteJobFunction.Invoke()	22.66	22.66	1.09	0 b	4.89	4.89
Unaccounted time between: PostLateUpdate.ProfilerEndFrame and EndFrameBarrier	7.84	7.84	1.00	0 b	1.64	1.64
MeshInstanceRendererSystem	6.41	0.00	1.00	0 b	1.39	0.00
WaitForJobGroupID	5.59	0.14	1.00	0 b	1.20	0.03
UpdateFunction.Invoke()	3.44	1.68	1.00	0 b	0.74	0.36
Camera.Render	1.89	0.07	1.00	0 b	0.41	0.03
PostLateUpdate.ProfilerEndFrame	1.00	0.00	1.00	0 b	0.22	0.00
Profiler.CollectGlobalStats	0.99	0.71	1.00	0 b	0.21	0.16
Drawing	0.97	0.00	1.00	0 b	0.21	0.01
Render.OpaqueGeometry	0.76	0.00	1.00	0 b	0.17	0.00
RenderForwardOpaque.Render	0.54	0.06	1.00	0 b	0.12	0.02
TransformSystem	0.33	0.00	1.00	0 b	0.07	0.00
Shadows.RenderShadowMap	0.26	0.15	1.00	0 b	0.06	0.04
Culling	0.27	0.05	1.00	0 b	0.06	0.02
EarlyUpdate.PerformanceAnalyticsUpdate	0.19	0.19	1.00	0 b	0.05	0.05
Initialization.PlayerUpdateTime	0.23	0.23	1.00	0 b	0.05	0.05
PreUpdate.SendMouseEvents	0.19	0.00	1.00	0 b	0.04	0.00
Monobehaviour.OnMouse_	0.18	0.00	1.00	0 b	0.04	0.00
SendMouseEvents.DoSendMouseEvents()	0.18	0.15	1.00	0 b	0.04	0.04
Profiler.CollectMemoryAllocationStats	0.12	0.12	1.00	0 b	0.03	0.03
SceneCulling	0.11	0.00	1.00	0 b	0.03	0.00
Update.ScriptRunBehaviourUpdate	0.10	0.00	1.00	0 b	0.03	0.00
UGUI.Rendering.RenderOverlays	0.11	0.00	1.00	0 b	0.03	0.00
UpdateDepthTexture	0.09	0.08	1.00	0 b	0.03	0.02
UIEvents.CanvasManagerRenderOverlays	0.11	0.00	1.00	0 b	0.03	0.00
EndFrameBarrier	0.09	0.00	1.00	0 b	0.03	0.00
CullResults.CreateSharedRendererScene	0.14	0.06	1.00	0 b	0.03	0.02

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.PrepareJob	0.00	0.00	1.00	0 b	0.03	0.00
RenderForwardOpaque.CollectShadows	0.13	0.00	1.00	0 b	0.03	0.00
Canvas.RenderOverlays	0.06	0.01	1.00	0 b	0.02	0.01
BehaviourUpdate	0.09	0.00	1.00	0 b	0.02	0.01
Shadows.CollectShadows	0.07	0.05	1.00	0 b	0.02	0.02
Shadows.PrepareShadowmap	0.09	0.06	1.00	0 b	0.02	0.02
Graphics.Blit	0.04	0.04	1.00	0 b	0.01	0.01
Shadows.CullDirectionalCascades	0.00	0.00	1.00	0 b	0.01	0.01
WatermarkRender	0.04	0.00	1.00	0 b	0.01	0.01
UIEvents.IMGUIRenderOverlays	0.01	0.00	1.00	0 b	0.01	0.00
GUI.Repaint	0.01	0.01	1.00	0 b	0.01	0.00
ProfilerConnection Poll	0.04	0.04	1.00	0 b	0.01	0.01
EarlyUpdate.PollPlayerConnection	0.04	0.00	1.00	0 b	0.01	0.00
Profiler.CollectAudioStats	0.05	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.PlayerUpdateCanvases	0.04	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.UpdateAudio	0.01	0.00	1.00	0 b	0.01	0.00
Shadows.ExtractCasters	0.00	0.00	1.00	0 b	0.01	0.01
UIEvents.WillRenderCanvases	0.04	0.00	1.00	0 b	0.01	0.00
UGUI.Rendering.UpdateBatches	0.04	0.00	1.00	0 b	0.01	0.00
Canvas.SendWillRenderCanvases()	0.03	0.00	1.00	0 b	0.01	0.00
Layout	0.03	0.02	1.00	0 b	0.01	0.01
Canvas.BuildBatch	0.01	0.01	1.00	0 b	0.01	0.01
EarlyUpdate.UpdateCanvasRectTransform	0.01	0.01	1.00	0 b	0.01	0.01
FpsDisplayer.Update()	0.04	0.04	1.00	0 b	0.01	0.01
PostLateUpdate.ClearImmediateRenderers	0.00	0.00	1.00	0 b	0.01	0.01
EarlyUpdate.UpdateInputManager	0.00	0.00	1.00	0 b	0.01	0.01
FixedUpdate.PhysicsFixedUpdate	0.01	0.01	1.49	0 b	0.01	0.00
AudioProfiler.CaptureFrame	0.03	0.00	1.00	0 b	0.01	0.00
AudioManager.Update	0.00	0.00	1.00	0 b	0.01	0.00
GUI.ProcessEvents	0.00	0.00	1.00	0 b	0.01	0.01
Camera.ImageEffects	0.03	0.00	1.00	0 b	0.01	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.21: Profiler data stats for dod - Average values

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	71.20	0.30	1.00	0 b	24.29	0.06
Gfx.WaitForPresent	69.70	69.70	1.00	0 b	23.73	23.73
PointMovementSystem	69.50	0.00	1.00	0 b	11.70	0.00
UpdateFunction.Invoke()	69.50	32.70	1.00	0 b	11.70	5.67
WaitForJobGroupID	48.50	35.00	1.00	0 b	8.17	5.71
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	48.30	0.10	3.00	0 b	8.14	0.01
ExecuteJobFunction.Invoke()	48.20	48.20	3.00	0 b	8.13	8.13
TransformSystem	32.70	0.00	1.00	0 b	5.60	0.00
Unaccounted time between: PostLateUpdate.ProfilerEndFrame and EndFrameBarrier	16.20	16.20	1.00	0 b	3.17	3.17
MeshInstanceRendererSystem	17.30	0.00	1.00	0 b	3.05	0.00
Update.ScriptRunBehaviourUpdate	7.30	0.00	1.00	0 b	1.26	0.00
BehaviourUpdate	7.20	0.40	1.00	0 b	1.25	0.14
FpsDisplayer.Update()	7.10	7.10	1.00	0 b	1.22	1.22
PostLateUpdate.ProfilerEndFrame	4.30	0.10	1.00	0 b	0.73	0.02
Profiler.CollectGlobalStats	4.30	2.60	1.00	0 b	0.73	0.44
Initialization.PlayerUpdateTime	4.10	4.10	1.00	0 b	0.72	0.72
Camera.Render	3.90	0.50	1.00	0 b	0.63	0.12
EarlyUpdate.PerformanceAnalyticsUpdate	3.50	3.50	1.00	0 b	0.58	0.58
ProfilerConnection Poll	3.00	3.00	1.00	0 b	0.53	0.53
EarlyUpdate.PollPlayerConnection	3.00	0.00	1.00	0 b	0.53	0.00
PreUpdate.SendMouseEvents	3.00	0.00	1.00	0 b	0.51	0.00
Monobehaviour.OnMouse_	3.00	0.00	1.00	0 b	0.51	0.00
SendMouseEvents.DoSendMouseEvents()	3.00	2.90	1.00	0 b	0.50	0.50
Profiler.CollectMemoryAllocationStats	2.90	2.90	1.00	0 b	0.50	0.50
FixedUpdate.NewInputEndFixedUpdate	1.40	1.40	3.00	0 b	0.45	0.24
NativeInputSystem.NotifyUpdate()	1.20	1.20	3.00	0 b	0.44	0.44
Drawing	2.50	0.10	1.00	0 b	0.41	0.01
Render.OpaqueGeometry	2.10	0.00	1.00	0 b	0.34	0.00
RenderForwardOpaque.Render	1.90	1.20	1.00	0 b	0.30	0.19
TransformInputBarrier	1.60	0.00	1.00	0 b	0.28	0.00
Initialization.AsyncUploadTimeSlicedUpdate	1.60	1.60	1.00	0 b	0.27	0.27

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
UIEvents.IMGUIRenderOverlays	1.40	0.00	1.00	0 b	0.24	0.00
EarlyUpdate.UpdateCanvasRectTransform	1.50	1.50	1.00	0 b	0.24	0.24
GUI.Repaint	1.40	1.30	1.00	0 b	0.23	0.23
FixedUpdate.PhysicsFixedUpdate	1.30	1.30	3.00	0 b	0.22	0.22
EndFrameBarrier	1.00	0.70	1.00	0 b	0.17	0.13
EarlyUpdate.UpdateInputManager	0.50	0.50	1.00	0 b	0.16	0.16
Shadows.RenderShadowMap	0.70	0.30	1.00	0 b	0.15	0.06
CopyInitialTransformFromGameObjectSystem	0.40	0.00	1.00	0 b	0.15	0.00
CopyTransformToGameObjectSystem	0.40	0.00	1.00	0 b	0.14	0.00
PostLateUpdate.PlayerUpdateCanvases	0.60	0.10	1.00	0 b	0.14	0.01
UIEvents.WillRenderCanvases	0.60	0.00	1.00	0 b	0.14	0.00
UGUI.Rendering.UpdateBatches	0.60	0.00	1.00	0 b	0.14	0.00
Transform2DSystem	0.40	0.40	1.00	0 b	0.14	0.13
Camera.ImageEffects	0.80	0.00	1.00	0 b	0.14	0.02
Culling	0.70	0.20	1.00	0 b	0.13	0.06
Graphics.Blit	0.80	0.80	1.00	0 b	0.13	0.13
HeadingSystem	0.30	0.30	1.00	0 b	0.12	0.11
Canvas.SendWillRenderCanvases()	0.40	0.00	1.00	0 b	0.12	0.00
Profiler.CollectDrawStats	0.60	0.60	1.00	0 b	0.11	0.11
Layout	0.40	0.40	1.00	0 b	0.11	0.11
Profiler.CollectAudioStats	0.60	0.20	1.00	0 b	0.10	0.03
PlayerEndOfFrame	0.60	0.60	1.00	0 b	0.10	0.10
EarlyUpdate.ScriptRunDelayedStartupFrame	0.50	0.10	1.00	0 b	0.10	0.01
AudioManager.FixedUpdate	0.60	0.60	3.00	0 b	0.10	0.10
FixedUpdate.AudioFixedUpdate	0.60	0.00	3.00	0 b	0.10	0.00
PreUpdate.WindUpdate	0.50	0.50	1.00	0 b	0.10	0.10
PostLateUpdate.PlayerSendFrameComplete	0.60	0.00	1.00	0 b	0.10	0.00
AudioProfiler.CaptureFrame	0.60	0.00	1.00	0 b	0.09	0.01
FixedUpdate.DirectorFixedSampleTime	0.60	0.60	3.00	0 b	0.09	0.09
UpdateDepthTexture	0.50	0.50	1.00	0 b	0.09	0.09
Camera.RenderSkybox	0.60	0.30	1.00	0 b	0.09	0.05

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
UpdatePreloading	0.10	0.00	1.00	0 b	0.04	0.00
EarlyUpdate.NewInputBeginFrame	0.10	0.10	1.00	0 b	0.04	0.02
PlayerCleanupCachedData	0.10	0.10	1.00	0 b	0.04	0.04
FixedUpdate.Physics2DFixedUpdate	0.20	0.00	3.00	0 b	0.04	0.00
EarlyUpdate.UpdatePreloading	0.10	0.00	1.00	0 b	0.04	0.00
EarlyUpdate.ClearLines	0.20	0.20	1.00	0 b	0.04	0.04
Loading.UpdatePreloading	0.10	0.00	1.00	0 b	0.04	0.01
Physics2D.Simulate	0.20	0.20	3.00	0 b	0.03	0.03
Shadows.PrepareJob	0.00	0.00	1.00	0 b	0.03	0.00
PostLateUpdate.UpdateVideoTextures	0.20	0.20	1.00	0 b	0.03	0.03
ReflectionProbeAnchorManagerUpdate	0.10	0.10	1.00	0 b	0.03	0.03
CullAllVisibleLights	0.20	0.00	1.00	0 b	0.03	0.01
PostLateUpdate.UpdateAudio	0.10	0.00	1.00	0 b	0.02	0.00
EarlyUpdate.GpuTimestamp	0.10	0.10	1.00	0 b	0.02	0.02
EarlyUpdate.XRUpdate	0.00	0.00	1.00	0 b	0.02	0.02
PreUpdate.IMGUISendQueuedEvents	0.10	0.00	1.00	0 b	0.02	0.00
AudioManager.Update	0.10	0.10	1.00	0 b	0.02	0.02
WatermarkRender	0.10	0.10	1.00	0 b	0.02	0.01
PostLateUpdate.PlayerSendFrameStarted	0.10	0.00	1.00	0 b	0.02	0.00
Shadows.CullDirectionalShadowCasters	0.10	0.10	1.00	0 b	0.02	0.02
PreUpdate.NewInputUpdate	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.MemoryFrameMaintenance	0.00	0.00	1.00	0 b	0.01	0.01
PostLateUpdate.UpdateCustomRenderTextures	0.00	0.00	1.00	0 b	0.01	0.00
EntityManager	0.10	0.00	1.00	0 b	0.01	0.01
UGUI.Rendering.EmitWorldScreenspaceCameraGeometry	0.00	0.00	1.00	0 b	0.01	0.01
Shadows.ExtractCasters	0.00	0.00	1.00	0 b	0.01	0.01
UIEvents.CanvasManagerEmitOffScreenGeometry	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.PlayerEmitCanvasGeometry	0.00	0.00	1.00	0 b	0.01	0.00
FrameEvents.NewInputSystemBeforeRenderSendEvents	0.00	0.00	1.00	0 b	0.01	0.00
WaitForRenderJobs	0.00	0.00	1.00	0 b	0.01	0.01

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.22: Profiler data stats for oop - Max values

Profiler Data E:
Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	63.13	0.00	1.00	0 b	84.07	0.01
Camera.Render	63.06	0.20	1.00	0 b	83.97	0.30
Drawing	49.61	0.00	1.00	0 b	66.08	0.02
Render.OpaqueGeometry	45.79	0.52	1.00	0 b	60.99	0.77
RenderForwardOpaque.Render	34.59	0.21	1.00	0 b	46.10	0.33
Shadows.RenderShadowMap	21.21	3.01	1.00	0 b	28.30	4.07
PreUpdate.SendMouseEvents	15.87	0.00	1.00	0 b	21.16	0.00
Monobehaviour.OnMouse_	15.87	0.00	1.00	0 b	21.16	0.00
SendMouseEvents.DoSendMouseEvents()	15.87	0.00	1.00	0 b	21.16	0.02
Physics.Raycast	15.85	3.95	1.00	0 b	21.14	5.32
Update.ScriptRunBehaviourUpdate	12.16	0.00	1.00	0 b	16.23	0.00
BehaviourUpdate	12.16	0.00	1.00	0 b	16.23	0.00
graph.Update()	12.15	12.15	1.00	0 b	16.22	16.22
Physics.SyncColliderTransform	11.84	11.45	1.00	0 b	15.81	15.29
Shadows.RenderJob	9.51	0.00	4.00	0 b	12.72	0.00
Shadows.RenderJobDir	9.51	7.31	4.00	0 b	12.72	9.79
PostLateUpdate.UpdateAllRenderers	8.35	0.00	1.00	0 b	11.16	0.00
RenderForwardOpaque.Prepare	5.82	5.82	1.00	0 b	7.80	7.80
UpdateDepthTexture	5.42	2.35	1.00	0 b	7.27	3.19
RenderForward.RenderLoopJob	5.12	2.61	1.00	0 b	6.84	3.50
Shadows.PrepareShadowmap	4.66	0.06	1.00	0 b	6.27	0.17
CullResults.CreateSharedRendererScene	4.16	0.02	1.00	0 b	5.60	0.05
WaitForJobGroupID	3.74	1.76	1.13	0 b	5.04	2.40
ExtractRenderNodeQueue	3.66	3.66	1.98	0 b	4.92	4.92
Render.Prepare	3.09	3.06	1.00	0 b	4.17	4.13
DestroyCullResults	2.90	0.00	1.00	0 b	3.92	0.02
DepthPass.Job	2.27	1.62	1.00	0 b	3.08	2.22
UpdateRendererBoundingVolumes	2.15	0.91	8.42	0 b	2.87	1.22
Batch.DrawInstanced	1.46	1.46	1.33	0 b	2.01	2.00
BatchRenderer.Flush	1.10	0.00	3.50	0 b	1.51	0.00
RenderLoop.CleanupNodeQueue	0.94	0.94	1.67	0 b	1.27	1.27

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
TempAlloc.Overflow	0.81	0.79	3.00	0 b	1.11	1.09
Shadows.Sort	0.74	0.74	4.00	0 b	1.06	1.06
Render.TransparentGeometry	0.61	0.60	1.00	0 b	0.85	0.84
Shadows.CullingCallbacks	0.52	0.52	1.00	0 b	0.75	0.75
Culling	0.52	0.00	1.00	0 b	0.75	0.01
SceneCulling	0.50	0.01	1.00	0 b	0.72	0.02
CullSendEvents	0.45	0.04	1.00	0 b	0.67	0.13
CullSceneDynamicObjects	0.19	0.10	1.98	0 b	0.28	0.18
TransformChangedDispatch	0.17	0.01	1.00	0 b	0.27	0.02
PrepareSceneNodesJob	0.02	0.02	1.14	0 b	0.12	0.12
Shadows.CullShadowCastersDirectional	0.03	0.00	1.37	0 b	0.11	0.00
CullObjectsWithoutUmbra	0.01	0.01	1.98	0 b	0.10	0.10
PostLateUpdate.ProfilerEndFrame	0.00	0.00	1.00	0 b	0.07	0.00
Shadows.CullShadowCastersWithoutUmbra	0.01	0.01	1.37	0 b	0.07	0.07
Profiler.CollectGlobalStats	0.00	0.00	1.00	0 b	0.07	0.05
RenderForwardOpaque.CollectShadows	0.00	0.00	1.00	0 b	0.05	0.01
Shadows.CombineDirectionalShadowCasterCullignIndexListsAndDestroy	0.01	0.00	1.00	0 b	0.05	0.00
EarlyUpdate.RendererNotifyInvisible	0.00	0.00	1.00	0 b	0.05	0.05
Shadows.CullShadowCastersDirectionalDetail	0.01	0.01	1.37	0 b	0.04	0.04
JobAlloc.Overflow	0.00	0.00	2.00	0 b	0.04	0.04
Shadows.CollectShadows	0.00	0.00	1.00	0 b	0.04	0.02
CullSceneDynamicObjectsCombineJob	0.00	0.00	1.00	0 b	0.04	0.00
UIEvents.CanvasManagerRenderOverlays	0.00	0.00	1.00	0 b	0.02	0.00
Canvas.RenderOverlays	0.00	0.00	1.00	0 b	0.02	0.00
UIEvents.IMGUIRenderOverlays	0.01	0.00	1.00	0 b	0.02	0.00
CullAllVisibleLights	0.00	0.00	1.00	0 b	0.02	0.00
GUI.Repaint	0.01	0.01	1.00	0 b	0.02	0.01
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.02	0.00
EarlyUpdate.PollPlayerConnection	0.01	0.00	1.00	0 b	0.02	0.00
ProfilerConnection Poll	0.01	0.01	1.00	0 b	0.02	0.02
UGUI.Rendering.RenderOverlays	0.00	0.00	1.00	0 b	0.02	0.00

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
CullPerObjectLights	0.00	0.00	1.00	0 b	0.02	0.02
RenderForwardAlpha.Render	0.00	0.00	1.00	0 b	0.01	0.00
Camera.ImageEffects	0.00	0.00	1.00	0 b	0.01	0.00
Graphics.Blit	0.00	0.00	1.00	0 b	0.01	0.01
Profiler.CollectMemoryAllocationStats	0.00	0.00	1.00	0 b	0.01	0.01
CombineJobResults	0.00	0.00	1.00	0 b	0.01	0.01
WatermarkRender	0.00	0.00	1.00	0 b	0.01	0.00
Material.SetPassUncached	0.00	0.00	1.00	0 b	0.01	0.01
EnlightenRuntimeManager.PostUpdate	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.UpdateAudio	0.00	0.00	1.00	0 b	0.01	0.00
Shadows.CullDirectionalShadowCasters	0.00	0.00	1.00	0 b	0.01	0.01
Camera.RenderSkybox	0.00	0.00	1.00	0 b	0.01	0.00
Canvas.BuildBatch	0.00	0.00	1.00	0 b	0.01	0.01
LateBehaviourUpdate	0.00	0.00	1.00	0 b	0.00	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.00	0.00	1.00	0 b	0.00	0.00
Physics.Interpolation	0.00	0.00	1.00	0 b	0.00	0.00
PreUpdate.WindUpdate	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.ExecuteMainThreadJobs	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.PhysicsResetInterpolatedTransformPosition	0.00	0.00	1.00	0 b	0.00	0.00
Initialization.AsyncUploadTimeSlicedUpdate	0.00	0.00	1.00	0 b	0.00	0.00
FixedBehaviourUpdate	0.00	0.00	6.65	0 b	0.00	0.00
ConstraintManager::Update::JobSetup	0.00	0.00	1.00	0 b	0.00	0.00
Initialization.XREarlyUpdate	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.DirectorLateUpdate	0.00	0.00	1.00	0 b	0.00	0.00
PreLateUpdate.AIUpdatePostScript	0.00	0.00	1.00	0 b	0.00	0.00
NavMeshManager	0.00	0.00	1.00	0 b	0.00	0.00
PreLateUpdate.EndGraphicsJobsLate	0.00	0.00	1.00	0 b	0.00	0.00
PreLateUpdate.UpdateMasterServerInterface	0.00	0.00	1.00	0 b	0.00	0.00
PreUpdate.IMGUISendQueuedEvents	0.00	0.00	1.00	0 b	0.00	0.00
PreUpdate.UpdateVideo	0.00	0.00	1.00	0 b	0.00	0.00
FixedUpdate.ScriptRunBehaviourFixedUpdate	0.00	0.00	6.65	0 b	0.00	0.00

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Figure B.23: Profiler data stats for oop - Average values

Figure B.24: Profiler data stats for oop - Max values

Profiler Data E:

Statistics: Min Values — Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	58.30	0.00	1.00	0 b	78.55	0.01
Camera.Render	58.20	0.10	1.00	0 b	78.48	0.24
Drawing	45.40	0.00	1.00	0 b	61.68	0.01
Render.OpaqueGeometry	42.00	0.50	1.00	0 b	56.91	0.72
RenderForwardOpaque.Render	30.90	0.20	1.00	0 b	42.41	0.30
Shadows.RenderShadowMap	18.70	2.30	1.00	0 b	26.05	3.36
SendMouseEvents.DoSendMouseEvents()	13.70	0.00	1.00	0 b	19.75	0.01
Monobehaviour.OnMouse_	13.70	0.00	1.00	0 b	19.75	0.00
PreUpdate.SendMouseEvents	13.70	0.00	1.00	0 b	19.75	0.00
Physics.Raycast	13.70	3.40	1.00	0 b	19.74	5.02
Update.ScriptRunBehaviourUpdate	10.50	0.00	1.00	0 b	15.60	0.00
BehaviourUpdate	10.50	0.00	1.00	0 b	15.60	0.00
graph.Update()	10.50	10.50	1.00	0 b	15.59	15.59
Physics.SyncColliderTransform	10.20	9.90	1.00	0 b	14.67	14.08
Shadows.RenderJob	8.30	0.00	4.00	0 b	11.51	0.00
Shadows.RenderJobDir	8.30	6.20	4.00	0 b	11.50	7.99
PostLateUpdate.UpdateAllRenderers	7.10	0.00	1.00	0 b	10.41	0.00
RenderForwardOpaque.Prepare	5.00	5.00	1.00	0 b	7.38	7.38
UpdateDepthTexture	4.50	2.00	1.00	0 b	6.60	2.76
Shadows.PrepareShadowmap	4.00	0.00	1.00	0 b	5.73	0.10
CullResults.CreateSharedRendererScene	3.60	0.00	1.00	0 b	5.08	0.01
DestroyCullResults	2.40	0.00	1.00	0 b	3.46	0.01
Render.Prepare	2.30	2.30	1.00	0 b	3.11	3.08
DepthPass.Job	1.90	1.30	1.00	0 b	2.72	1.94
ExtractRenderNodeQueue	1.40	1.40	1.00	0 b	1.97	1.97
Shadows.Sort	0.60	0.60	4.00	0 b	0.97	0.97
Batch.DrawInstanced	0.40	0.40	1.00	0 b	0.68	0.68
Shadows.CullingCallbacks	0.40	0.40	1.00	0 b	0.68	0.68
Culling	0.40	0.00	1.00	0 b	0.64	0.01
SceneCulling	0.40	0.00	1.00	0 b	0.62	0.00
CullSceneDynamicObjects	0.00	0.00	1.00	0 b	0.11	0.06

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics: Min Values — Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
CullSendEvents	0.00	0.00	1.00	0 b	0.08	0.07
PrepareSceneNodesJob	0.00	0.00	1.00	0 b	0.07	0.07
Profiler.CollectGlobalStats	0.00	0.00	1.00	0 b	0.06	0.04
PostLateUpdate.ProfilerEndFrame	0.00	0.00	1.00	0 b	0.06	0.00
Shadows.CullShadowCastersDirectional	0.00	0.00	1.00	0 b	0.05	0.00
CullObjectsWithoutUmbra	0.00	0.00	1.00	0 b	0.04	0.04
RenderForwardOpaque.CollectShadows	0.00	0.00	1.00	0 b	0.04	0.01
EarlyUpdate.RendererNotifyInvisible	0.00	0.00	1.00	0 b	0.04	0.04
Shadows.CullShadowCastersWithoutUmbra	0.00	0.00	1.00	0 b	0.03	0.03
Shadows.CollectShadows	0.00	0.00	1.00	0 b	0.03	0.01
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.02	0.00
Shadows.CombineDirectionalShadowCasterCullignIndexListsAndDestroy	0.00	0.00	1.00	0 b	0.02	0.00
CullSceneDynamicObjectsCombineJob	0.00	0.00	1.00	0 b	0.02	0.00
WaitForJobGroupID	0.00	0.00	1.00	0 b	0.02	0.00
Render.TransparentGeometry	0.00	0.00	1.00	0 b	0.02	0.01
Profiler.CollectMemoryAllocationStats	0.00	0.00	1.00	0 b	0.01	0.01
Canvas.RenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
UGUI.Rendering.RenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
UIEvents.CanvasManagerRenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
Shadows.CullShadowCastersDirectionalDetail	0.00	0.00	1.00	0 b	0.01	0.01
Graphics.Blit	0.00	0.00	1.00	0 b	0.01	0.01
Canvas.BuildBatch	0.00	0.00	1.00	0 b	0.01	0.01
CullPerObjectLights	0.00	0.00	1.00	0 b	0.01	0.01
Camera.RenderSkybox	0.00	0.00	1.00	0 b	0.01	0.00
EnlightenRuntimeManager.PostUpdate	0.00	0.00	1.00	0 b	0.01	0.00
CullAllVisibleLights	0.00	0.00	1.00	0 b	0.01	0.00
FixedUpdate.ScriptRunBehaviourFixedUpdate	0.00	0.00	6.00	0 b	0.00	0.00
Network.Update	0.00	0.00	1.00	0 b	0.00	0.00
FixedBehaviourUpdate	0.00	0.00	6.00	0 b	0.00	0.00
Initialization.AsyncUploadTimeSlicedUpdate	0.00	0.00	1.00	0 b	0.00	0.00
Initialization.XREarlyUpdate	0.00	0.00	1.00	0 b	0.00	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.25: Profiler data stats for dod - Minimum values

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	50.00	0.04	1.00	0 b	13.66	0.01
Gfx.WaitForPresent	47.61	47.61	1.00	0 b	13.14	13.14
PointMovementSystem	37.87	0.00	1.00	0 b	8.14	0.00
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	22.68	0.00	1.09	0 b	4.90	0.00
ExecuteJobFunction.Invoke()	22.66	22.66	1.09	0 b	4.89	4.89
Unaccounted time between: PostLateUpdate.ProfilerEndFrame and EndFrameBarrier	7.84	7.84	1.00	0 b	1.64	1.64
MeshInstanceRendererSystem	6.41	0.00	1.00	0 b	1.39	0.00
WaitForJobGroupID	5.59	0.14	1.00	0 b	1.20	0.03
UpdateFunction.Invoke()	3.44	1.68	1.00	0 b	0.74	0.36
Camera.Render	1.89	0.07	1.00	0 b	0.41	0.03
PostLateUpdate.ProfilerEndFrame	1.00	0.00	1.00	0 b	0.22	0.00
Profiler.CollectGlobalStats	0.99	0.71	1.00	0 b	0.21	0.16
Drawing	0.97	0.00	1.00	0 b	0.21	0.01
Render.OpaqueGeometry	0.76	0.00	1.00	0 b	0.17	0.00
RenderForwardOpaque.Render	0.54	0.06	1.00	0 b	0.12	0.02
TransformSystem	0.33	0.00	1.00	0 b	0.07	0.00
Shadows.RenderShadowMap	0.26	0.15	1.00	0 b	0.06	0.04
Culling	0.27	0.05	1.00	0 b	0.06	0.02
EarlyUpdate.PerformanceAnalyticsUpdate	0.19	0.19	1.00	0 b	0.05	0.05
Initialization.PlayerUpdateTime	0.23	0.23	1.00	0 b	0.05	0.05
PreUpdate.SendMouseEvents	0.19	0.00	1.00	0 b	0.04	0.00
Monobehaviour.OnMouse_	0.18	0.00	1.00	0 b	0.04	0.00
SendMouseEvents.DoSendMouseEvents()	0.18	0.15	1.00	0 b	0.04	0.04
Profiler.CollectMemoryAllocationStats	0.12	0.12	1.00	0 b	0.03	0.03
SceneCulling	0.11	0.00	1.00	0 b	0.03	0.00
Update.ScriptRunBehaviourUpdate	0.10	0.00	1.00	0 b	0.03	0.00
UGUI.Rendering.RenderOverlays	0.11	0.00	1.00	0 b	0.03	0.00
UpdateDepthTexture	0.09	0.08	1.00	0 b	0.03	0.02
UIEvents.CanvasManagerRenderOverlays	0.11	0.00	1.00	0 b	0.03	0.00
EndFrameBarrier	0.09	0.00	1.00	0 b	0.03	0.00
CullResults.CreateSharedRendererScene	0.14	0.06	1.00	0 b	0.03	0.02

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.PrepareJob	0.00	0.00	1.00	0 b	0.03	0.00
RenderForwardOpaque.CollectShadows	0.13	0.00	1.00	0 b	0.03	0.00
Canvas.RenderOverlays	0.06	0.01	1.00	0 b	0.02	0.01
BehaviourUpdate	0.09	0.00	1.00	0 b	0.02	0.01
Shadows.CollectShadows	0.07	0.05	1.00	0 b	0.02	0.02
Shadows.PrepareShadowmap	0.09	0.06	1.00	0 b	0.02	0.02
Graphics.Blit	0.04	0.04	1.00	0 b	0.01	0.01
Shadows.CullDirectionalCascades	0.00	0.00	1.00	0 b	0.01	0.01
WatermarkRender	0.04	0.00	1.00	0 b	0.01	0.01
UIEvents.IMGUIRenderOverlays	0.01	0.00	1.00	0 b	0.01	0.00
GUI.Repaint	0.01	0.01	1.00	0 b	0.01	0.00
ProfilerConnection Poll	0.04	0.04	1.00	0 b	0.01	0.01
EarlyUpdate.PollPlayerConnection	0.04	0.00	1.00	0 b	0.01	0.00
Profiler.CollectAudioStats	0.05	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.PlayerUpdateCanvases	0.04	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.UpdateAudio	0.01	0.00	1.00	0 b	0.01	0.00
Shadows.ExtractCasters	0.00	0.00	1.00	0 b	0.01	0.01
UIEvents.WillRenderCanvases	0.04	0.00	1.00	0 b	0.01	0.00
UGUI.Rendering.UpdateBatches	0.04	0.00	1.00	0 b	0.01	0.00
Canvas.SendWillRenderCanvases()	0.03	0.00	1.00	0 b	0.01	0.00
Layout	0.03	0.02	1.00	0 b	0.01	0.01
Canvas.BuildBatch	0.01	0.01	1.00	0 b	0.01	0.01
EarlyUpdate.UpdateCanvasRectTransform	0.01	0.01	1.00	0 b	0.01	0.01
FpsDisplayer.Update()	0.04	0.04	1.00	0 b	0.01	0.01
PostLateUpdate.ClearImmediateRenderers	0.00	0.00	1.00	0 b	0.01	0.01
EarlyUpdate.UpdateInputManager	0.00	0.00	1.00	0 b	0.01	0.01
FixedUpdate.PhysicsFixedUpdate	0.01	0.01	1.49	0 b	0.01	0.00
AudioProfiler.CaptureFrame	0.03	0.00	1.00	0 b	0.01	0.00
AudioManager.Update	0.00	0.00	1.00	0 b	0.01	0.00
GUI.ProcessEvents	0.00	0.00	1.00	0 b	0.01	0.01
Camera.ImageEffects	0.03	0.00	1.00	0 b	0.01	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.26: Profiler data stats for dod - Average values

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	71.20	0.30	1.00	0 b	24.29	0.06
Gfx.WaitForPresent	69.70	69.70	1.00	0 b	23.73	23.73
PointMovementSystem	69.50	0.00	1.00	0 b	11.70	0.00
UpdateFunction.Invoke()	69.50	32.70	1.00	0 b	11.70	5.67
WaitForJobGroupID	48.50	35.00	1.00	0 b	8.17	5.71
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	48.30	0.10	3.00	0 b	8.14	0.01
ExecuteJobFunction.Invoke()	48.20	48.20	3.00	0 b	8.13	8.13
TransformSystem	32.70	0.00	1.00	0 b	5.60	0.00
Unaccounted time between: PostLateUpdate.ProfilerEndFrame and EndFrameBarrier	16.20	16.20	1.00	0 b	3.17	3.17
MeshInstanceRendererSystem	17.30	0.00	1.00	0 b	3.05	0.00
Update.ScriptRunBehaviourUpdate	7.30	0.00	1.00	0 b	1.26	0.00
BehaviourUpdate	7.20	0.40	1.00	0 b	1.25	0.14
FpsDisplayer.Update()	7.10	7.10	1.00	0 b	1.22	1.22
PostLateUpdate.ProfilerEndFrame	4.30	0.10	1.00	0 b	0.73	0.02
Profiler.CollectGlobalStats	4.30	2.60	1.00	0 b	0.73	0.44
Initialization.PlayerUpdateTime	4.10	4.10	1.00	0 b	0.72	0.72
Camera.Render	3.90	0.50	1.00	0 b	0.63	0.12
EarlyUpdate.PerformanceAnalyticsUpdate	3.50	3.50	1.00	0 b	0.58	0.58
ProfilerConnection Poll	3.00	3.00	1.00	0 b	0.53	0.53
EarlyUpdate.PollPlayerConnection	3.00	0.00	1.00	0 b	0.53	0.00
PreUpdate.SendMouseEvents	3.00	0.00	1.00	0 b	0.51	0.00
Monobehaviour.OnMouse_	3.00	0.00	1.00	0 b	0.51	0.00
SendMouseEvents.DoSendMouseEvents()	3.00	2.90	1.00	0 b	0.50	0.50
Profiler.CollectMemoryAllocationStats	2.90	2.90	1.00	0 b	0.50	0.50
FixedUpdate.NewInputEndFixedUpdate	1.40	1.40	3.00	0 b	0.45	0.24
NativeInputSystem.NotifyUpdate()	1.20	1.20	3.00	0 b	0.44	0.44
Drawing	2.50	0.10	1.00	0 b	0.41	0.01
Render.OpaqueGeometry	2.10	0.00	1.00	0 b	0.34	0.00
RenderForwardOpaque.Render	1.90	1.20	1.00	0 b	0.30	0.19
TransformInputBarrier	1.60	0.00	1.00	0 b	0.28	0.00
Initialization.AsyncUploadTimeSlicedUpdate	1.60	1.60	1.00	0 b	0.27	0.27

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
UIEvents.IMGUIRenderOverlays	1.40	0.00	1.00	0 b	0.24	0.00
EarlyUpdate.UpdateCanvasRectTransform	1.50	1.50	1.00	0 b	0.24	0.24
GUI.Repaint	1.40	1.30	1.00	0 b	0.23	0.23
FixedUpdate.PhysicsFixedUpdate	1.30	1.30	3.00	0 b	0.22	0.22
EndFrameBarrier	1.00	0.70	1.00	0 b	0.17	0.13
EarlyUpdate.UpdateInputManager	0.50	0.50	1.00	0 b	0.16	0.16
Shadows.RenderShadowMap	0.70	0.30	1.00	0 b	0.15	0.06
CopyInitialTransformFromGameObjectSystem	0.40	0.00	1.00	0 b	0.15	0.00
CopyTransformToGameObjectSystem	0.40	0.00	1.00	0 b	0.14	0.00
PostLateUpdate.PlayerUpdateCanvases	0.60	0.10	1.00	0 b	0.14	0.01
UIEvents.WillRenderCanvases	0.60	0.00	1.00	0 b	0.14	0.00
UGUI.Rendering.UpdateBatches	0.60	0.00	1.00	0 b	0.14	0.00
Transform2DSystem	0.40	0.40	1.00	0 b	0.14	0.13
Camera.ImageEffects	0.80	0.00	1.00	0 b	0.14	0.02
Culling	0.70	0.20	1.00	0 b	0.13	0.06
Graphics.Blit	0.80	0.80	1.00	0 b	0.13	0.13
HeadingSystem	0.30	0.30	1.00	0 b	0.12	0.11
Canvas.SendWillRenderCanvases()	0.40	0.00	1.00	0 b	0.12	0.00
Profiler.CollectDrawStats	0.60	0.60	1.00	0 b	0.11	0.11
Layout	0.40	0.40	1.00	0 b	0.11	0.11
Profiler.CollectAudioStats	0.60	0.20	1.00	0 b	0.10	0.03
PlayerEndOfFrame	0.60	0.60	1.00	0 b	0.10	0.10
EarlyUpdate.ScriptRunDelayedStartupFrame	0.50	0.10	1.00	0 b	0.10	0.01
AudioManager.FixedUpdate	0.60	0.60	3.00	0 b	0.10	0.10
FixedUpdate.AudioFixedUpdate	0.60	0.00	3.00	0 b	0.10	0.00
PreUpdate.WindUpdate	0.50	0.50	1.00	0 b	0.10	0.10
PostLateUpdate.PlayerSendFrameComplete	0.60	0.00	1.00	0 b	0.10	0.00
AudioProfiler.CaptureFrame	0.60	0.00	1.00	0 b	0.09	0.01
FixedUpdate.DirectorFixedSampleTime	0.60	0.60	3.00	0 b	0.09	0.09
UpdateDepthTexture	0.50	0.50	1.00	0 b	0.09	0.09
Camera.RenderSkybox	0.60	0.30	1.00	0 b	0.09	0.05

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
UpdatePreloading	0.10	0.00	1.00	0 b	0.04	0.00
EarlyUpdate.NewInputBeginFrame	0.10	0.10	1.00	0 b	0.04	0.02
PlayerCleanupCachedData	0.10	0.10	1.00	0 b	0.04	0.04
FixedUpdate.Physics2DFixedUpdate	0.20	0.00	3.00	0 b	0.04	0.00
EarlyUpdate.UpdatePreloading	0.10	0.00	1.00	0 b	0.04	0.00
EarlyUpdate.ClearLines	0.20	0.20	1.00	0 b	0.04	0.04
Loading.UpdatePreloading	0.10	0.00	1.00	0 b	0.04	0.01
Physics2D.Simulate	0.20	0.20	3.00	0 b	0.03	0.03
Shadows.PrepareJob	0.00	0.00	1.00	0 b	0.03	0.00
PostLateUpdate.UpdateVideoTextures	0.20	0.20	1.00	0 b	0.03	0.03
ReflectionProbeAnchorManagerUpdate	0.10	0.10	1.00	0 b	0.03	0.03
CullAllVisibleLights	0.20	0.00	1.00	0 b	0.03	0.01
PostLateUpdate.UpdateAudio	0.10	0.00	1.00	0 b	0.02	0.00
EarlyUpdate.GpuTimestamp	0.10	0.10	1.00	0 b	0.02	0.02
EarlyUpdate.XRUpdate	0.00	0.00	1.00	0 b	0.02	0.02
PreUpdate.IMGUISendQueuedEvents	0.10	0.00	1.00	0 b	0.02	0.00
AudioManager.Update	0.10	0.10	1.00	0 b	0.02	0.02
WatermarkRender	0.10	0.10	1.00	0 b	0.02	0.01
PostLateUpdate.PlayerSendFrameStarted	0.10	0.00	1.00	0 b	0.02	0.00
Shadows.CullDirectionalShadowCasters	0.10	0.10	1.00	0 b	0.02	0.02
PreUpdate.NewInputUpdate	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.MemoryFrameMaintenance	0.00	0.00	1.00	0 b	0.01	0.01
PostLateUpdate.UpdateCustomRenderTextures	0.00	0.00	1.00	0 b	0.01	0.00
EntityManager	0.10	0.00	1.00	0 b	0.01	0.01
UGUI.Rendering.EmitWorldScreenspaceCameraGeometry	0.00	0.00	1.00	0 b	0.01	0.01
Shadows.ExtractCasters	0.00	0.00	1.00	0 b	0.01	0.01
UIEvents.CanvasManagerEmitOffScreenGeometry	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.PlayerEmitCanvasGeometry	0.00	0.00	1.00	0 b	0.01	0.00
FrameEvents.NewInputSystemBeforeRenderSendEvents	0.00	0.00	1.00	0 b	0.01	0.00
WaitForRenderJobs	0.00	0.00	1.00	0 b	0.01	0.01

Figure B.27: Profiler data stats for dod - Max values

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	23.90	0.00	1.00	400 B	10.73	0.01
Camera.Render	23.30	0.00	1.00	0 b	10.33	0.03
Drawing	14.00	0.00	1.00	0 b	5.90	0.01
Render.OpaqueGeometry	13.50	0.00	1.00	0 b	5.68	0.01
PreLateUpdate.DirectorUpdateAnimationEnd	8.70	0.00	1.00	0 b	4.35	0.00
RenderForwardOpaque.Render	9.40	0.00	1.00	0 b	3.66	0.02
PostLateUpdate.UpdateAllSkinnedMeshes	7.00	0.00	1.00	0 b	3.36	0.02
MeshSkinning.Update	6.90	0.30	1.00	0 b	3.33	0.16
MeshSkinning.Prepare	6.50	1.90	1.00	0 b	3.17	0.96
PreUpdate.SendMouseEvents	6.30	0.00	1.00	0 b	2.45	0.00
Monobehaviour.OnMouse_	6.30	0.00	1.00	0 b	2.45	0.00
SendMouseEvents.DoSendMouseEvents()	6.30	0.00	1.00	0 b	2.45	0.01
Animators.IKJob	3.30	0.00	2.00	0 b	1.67	0.04
Shadows.RenderShadowMap	3.70	0.10	1.00	0 b	1.48	0.04
ViewSynchronizationSystem	2.90	0.00	1.00	0 b	1.39	0.00
PreLateUpdate.DirectorUpdateAnimationBegin	2.90	0.00	1.00	0 b	1.37	0.00
Shadows.PrepareShadowmap	3.20	2.10	1.00	0 b	1.26	0.81
Animators.WriteJob	2.00	0.10	3.00	0 b	1.12	0.08
Animators.Update	2.10	0.00	1.00	0 b	1.05	0.01
PathFinderSystem:PathfinderJob	2.00	0.00	1.00	10.8 KB	1.02	0.00
ExecuteJobFunction.Invoke()	2.00	2.00	1.00	10.8 KB	1.02	1.01
Shadows.RenderJobDir	2.30	1.00	2.00	0 b	0.95	0.38
Shadows.RenderJob	2.30	0.00	2.00	0 b	0.95	0.00
CullResults.CreateSharedRendererScene	2.20	1.50	1.00	0 b	0.86	0.61
PostLateUpdate.UpdateAllRenderers	1.70	0.00	1.00	0 b	0.80	0.00
Animator.WriteAnimatedValues	1.60	1.60	6.00	0 b	0.71	0.71
UpdateDepthTexture	1.50	0.20	1.00	0 b	0.68	0.09
Shadows.PrepareJob	1.40	0.00	1.00	0 b	0.51	0.00
DepthPass.Job	1.10	0.50	1.00	0 b	0.50	0.20
Animators.RetargeterJob	1.00	0.00	1.00	0 b	0.50	0.02
Animators.ProcessRootMotionJob	0.90	0.00	1.00	0 b	0.45	0.02

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
MoveSystem	0.80	0.00	1.00	0 b	0.40	0.00
Animators.ProcessAnimationsJob	0.80	0.00	1.00	0 b	0.37	0.02
Animators.PrepareFirstPass	0.70	0.70	1.00	0 b	0.34	0.34
Shadows.ExtractCasters	0.80	0.80	1.00	0 b	0.30	0.30
RenderForwardOpaque.Prepare	0.70	0.70	1.00	0 b	0.30	0.30
BehaviourUpdate	0.70	0.20	1.00	308 B	0.27	0.09
Update.ScriptRunBehaviourUpdate	0.70	0.00	1.00	308 B	0.27	0.00
Animator.EvaluateRetargeter	0.40	0.40	1.00	0 b	0.25	0.25
Director.PrepareFrameJob	0.50	0.00	7.00	0 b	0.24	0.02
Animators.PrepareSecondPass	0.40	0.30	1.00	0 b	0.21	0.21
Shadows.CullDirectionalCascades	0.50	0.50	1.00	0 b	0.20	0.20
Animators.ApplyOnAnimatorMove	0.30	0.10	1.00	0 b	0.18	0.06
DestroyCullResults	0.30	0.00	1.00	0 b	0.18	0.00
MeshSkinning.Render	0.20	0.20	1672.00	0 b	0.13	0.13
Animator.EvaluateIK	0.30	0.30	2.00	0 b	0.12	0.12
GameAgentSystem	0.20	0.00	1.00	0 b	0.12	0.00
Culling	0.20	0.00	1.00	0 b	0.11	0.01
Animator.ApplyOnAnimatorMove	0.20	0.10	400.00	0 b	0.11	0.07
ExtractRenderNodeQueue	0.20	0.20	1.00	0 b	0.11	0.11
UIEvents.IMGUIRenderOverlays	0.20	0.00	1.00	400 B	0.10	0.00
Animators.DirtySceneObjects	0.20	0.20	1.00	0 b	0.10	0.10
GUI.Repaint	0.20	0.00	1.00	400 B	0.10	0.04
Shadows.Sort	0.10	0.10	2.00	0 b	0.09	0.09
AbilityAgentSystem	0.20	0.00	1.00	0 b	0.09	0.00
Animators.SortWriteJob	0.10	0.10	2.00	0 b	0.09	0.09
Profiler.CollectGlobalStats	0.20	0.00	1.00	0 b	0.08	0.00
DepthPass.Sort	0.10	0.10	1.00	0 b	0.08	0.08
Shadows.CullingCallbacks	0.10	0.10	1.00	0 b	0.08	0.08
SceneCulling	0.10	0.00	1.00	0 b	0.08	0.00
PostLateUpdate.ProfilerEndFrame	0.20	0.00	1.00	0 b	0.08	0.00
PostLateUpdate.DirectorLateUpdate	0.10	0.10	1.00	0 b	0.07	0.07

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.ProfilerEndFrame	0.20	0.00	1.00	0 b	0.08	0.00
PostLateUpdate.DirectorLateUpdate	0.10	0.10	1.00	0 b	0.07	0.07
Animator.ProcessRootMotion	0.10	0.10	8.00	0 b	0.07	0.07
AnimatorControllerPlayable.PrepareFrame	0.10	0.10	7.00	0 b	0.07	0.07
PrepareSceneNodesCombineJob	0.10	0.00	1.00	0 b	0.05	0.00
Render.Prepare	0.10	0.10	1.00	0 b	0.05	0.05
EventSystem.Update()	0.00	0.00	1.00	0 b	0.04	0.04
Profiler.CollectDrawStats	0.10	0.10	1.00	0 b	0.04	0.04
RenderForwardOpaque.CollectShadows	0.00	0.00	1.00	0 b	0.04	0.01
MouseController.Update()	0.10	0.00	1.00	0 b	0.04	0.02
Animator.ApplyBuiltinRootMotion	0.00	0.00	400.00	0 b	0.03	0.03
Animators.FireAnimationEventsAndBehaviours	0.00	0.00	1.00	0 b	0.03	0.02
AnimationViewSystem	0.00	0.00	1.00	0 b	0.03	0.00
AbilityUpdateSystem	0.00	0.00	1.00	0 b	0.03	0.00
Shadows.CollectShadows	0.00	0.00	1.00	0 b	0.02	0.01
Profiler.CollectMemoryAllocationStats	0.00	0.00	1.00	0 b	0.02	0.02
Batch.DrawDynamic	0.00	0.00	9.00	0 b	0.02	0.02
TempAlloc.Overflow	0.00	0.00	2.00	0 b	0.02	0.00
Director.SampleTime	0.00	0.00	1.00	0 b	0.02	0.02
PathFinderSystem	0.00	0.00	1.00	0 b	0.02	0.00
Running Path Modifiers	0.00	0.00	1.00	56 B	0.02	0.02
Physics.Raycast	0.00	0.00	1.00	0 b	0.02	0.02
EarlyUpdate.DirectorSampleTime	0.00	0.00	1.00	0 b	0.02	0.00
TransformSystem	0.00	0.00	1.00	0 b	0.01	0.00
EnlightenRuntimeManager.PostUpdate	0.00	0.00	1.00	0 b	0.01	0.00
UIEvents.CanvasManagerRenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
Canvas.RenderOverlays	0.00	0.00	2.00	0 b	0.01	0.00
UGUI.Rendering.RenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.01	0.00
GUIUtility.BeginGUI()	0.00	0.00	3.00	400 B	0.01	0.01
JobAlloc.Overflow	0.00	0.00	1.00	0 b	0.01	0.01

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.28: DOD DwarfHeim profiler minimum elapsed time statistics for test with animation and models activated

Profiler Data E:

Statistics: Average Values — Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	36.80	0.00	1.00	400 B	13.88	0.01
Camera.Render	36.13	0.11	1.00	0 b	13.63	0.06
Drawing	21.17	0.03	1.00	0 b	8.00	0.03
Render.OpaqueGeometry	20.62	0.00	1.00	0 b	7.79	0.02
RenderForwardOpaque.Render	14.08	0.01	1.00	0 b	5.34	0.03
PreLateUpdate.DirectorUpdateAnimationEnd	14.03	0.00	1.00	0 b	5.24	0.00
PostLateUpdate.UpdateAllSkinnedMeshes	10.47	0.06	1.00	0 b	3.95	0.04
MeshSkinning.Update	10.35	0.60	1.00	0 b	3.91	0.24
MeshSkinning.Prepare	9.71	3.04	1.00	0 b	3.66	1.16
PreUpdate.SendMouseEvents	9.53	0.00	1.00	0 b	3.60	0.00
Monobehaviour.OnMouse_	9.53	0.00	1.00	0 b	3.60	0.00
SendMouseEvents.DoSendMouseEvents()	9.53	0.01	1.00	0 b	3.60	0.02
Update.ScriptRunBehaviourUpdate	8.86	0.00	1.00	43.88 KB	3.55	0.00
BehaviourUpdate	8.86	0.32	1.00	43.88 KB	3.55	0.13
Animators.Update	8.80	0.29	1.00	0 b	3.29	0.12
MeshSkinning.CalcMatrices	6.52	6.52	1973.65	0 b	2.47	2.47
Shadows.RenderShadowMap	6.03	0.13	1.00	0 b	2.29	0.07
RVOSimulator.Update()	4.97	4.97	1.00	0 b	2.04	2.04
Animators.IKJob	5.18	0.15	2.00	0 b	1.94	0.07
Physics.Raycast	4.77	0.50	1.00	0 b	1.81	0.20
PreLateUpdate.DirectorUpdateAnimationBegin	4.74	0.00	1.00	0 b	1.79	0.00
Shadows.PrepareShadowmap	4.67	3.19	1.00	0 b	1.77	1.21
CullResults.CreateSharedRendererScene	4.64	3.28	1.00	0 b	1.75	1.24
ViewSynchronizationSystem	4.56	0.00	1.00	0 b	1.72	0.00
Animator.EvaluateIK	4.44	4.44	47.73	0 b	1.66	1.66
Shadows.RenderJob	4.11	0.00	2.00	0 b	1.57	0.00
Shadows.RenderJobDir	4.11	1.96	2.00	0 b	1.57	0.76
UpdateDepthTexture	3.62	0.55	1.00	0 b	1.37	0.22
Animators.WriteJob	3.63	0.29	3.00	0 b	1.36	0.12
Culling	3.49	0.03	1.00	0 b	1.35	0.02
SceneCulling	3.38	0.19	1.00	0 b	1.31	0.08

File path
\profiler_data.json Open Edit

Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics: Average Values — Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
RenderForward.RenderLoopJob	3.18	2.29	1.00	0 b	1.20	0.87
CullSendEvents	3.01	0.06	1.00	0 b	1.16	0.03
AstarPath.Update()	2.80	1.53	1.00	43.58 KB	1.10	0.60
PostLateUpdate.UpdateAllRenderers	2.92	0.00	1.00	0 b	1.10	0.00
Animator.WriteAnimatedValues	2.85	2.85	50.87	0 b	1.07	1.07
DepthPass.Job	2.74	1.30	1.00	0 b	1.04	0.50
PathFinderSystem:PathfinderJob	2.00	0.00	1.00	10.8 KB	1.02	0.00
ExecuteJobFunction.Invoke()	2.00	2.00	1.00	10.8 KB	1.02	1.01
Director.ProcessFrame	2.51	0.00	1.26	0 b	0.94	0.00
Physics.SyncColliderTransform	2.47	2.01	1.00	0 b	0.94	0.76
Running Path Modifiers	1.75	1.75	57.21	3.66 KB	0.69	0.69
PrepareSceneNodesCombineJob	1.73	0.00	1.00	0 b	0.68	0.00
Shadows.PrepareJob	1.78	0.00	1.00	0 b	0.67	0.00
Physics.SyncRigidbodyTransform	1.75	1.13	1.00	0 b	0.67	0.44
RenderForwardOpaque.Prepare	1.70	1.48	1.00	0 b	0.65	0.57
WaitForJobGroupID	1.57	0.50	1.28	2.22 B	0.60	0.20
Animators.RetargeterJob	1.48	0.03	1.00	0 b	0.57	0.03
Animators.ProcessRootMotionJob	1.45	0.05	1.00	0 b	0.55	0.04
Animators.ProcessAnimationsJob	1.39	0.06	1.00	0 b	0.53	0.04
MoveSystem	1.32	0.00	1.00	2.05 KB	0.51	0.00
Animator.EvaluateRetargeter	1.34	1.34	49.49	0 b	0.51	0.51
Calling Path Callbacks	1.24	0.29	1.00	35.99 KB	0.49	0.12
Animator.ProcessRootMotion	1.18	1.18	50.72	0 b	0.46	0.45
Animators.PrepareFirstPass	1.17	1.17	1.00	0 b	0.45	0.45
Animator.ProcessAnimations	1.14	1.14	50.02	0 b	0.44	0.44
BasicChainUpdateSystem	1.12	0.00	1.00	52.93 KB	0.44	0.00
GameAgentSystem	1.03	0.00	1.00	5.4 KB	0.41	0.00
Shadows.ExtractCasters	0.98	0.98	1.00	0 b	0.37	0.37
DestroyCullResults	0.88	0.00	1.00	0 b	0.35	0.00
Director.PrepareFrameJob	0.88	0.38	50.56	0 b	0.34	0.16
Animators.PrepareSecondPass	0.74	0.73	1.00	0 b	0.29	0.29

File path
\profiler_data.json Open Edit

Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics: Average Values — Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
AbilityAgentSystem	0.73	0.00	1.00	15.73 KB	0.29	0.00
Shadows.CullDirectionalCascades	0.75	0.75	1.00	0 b	0.29	0.29
MeshSkinning.Render	0.72	0.72	1826.54	0 b	0.28	0.28
UpdateRendererBoundingVolumes	0.72	0.34	8.60	0 b	0.27	0.13
Animators.ApplyOnAnimatorMove	0.68	0.18	1.00	0 b	0.27	0.08
TransformChangedDispatch	0.67	0.32	1.63	0 b	0.26	0.13
BatchRenderer.Flush	0.65	0.45	1487.57	0 b	0.26	0.18
MovingTestSystem	0.60	0.00	1.00	33.83 KB	0.25	0.00
SkinnedMeshUpdateAllNeeded	0.46	0.46	2.13	0 b	0.18	0.18
AnimatorControllerPlayable.PrepareFrame	0.44	0.44	50.56	0 b	0.18	0.18
Animator.ApplyOnAnimatorMove	0.44	0.28	400.00	0 b	0.18	0.12
Animators.DirtySceneObjects	0.44	0.44	1.00	0 b	0.18	0.18
ExtractRenderNodeQueue	0.43	0.43	2.57	0 b	0.17	0.17
GUI.Repaint	0.41	0.16	1.00	400 B	0.17	0.08
UIEvents.IMGUIRenderOverlays	0.41	0.00	1.00	400 B	0.17	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.38	0.00	1.00	0 b	0.15	0.00
LateBehaviourUpdate	0.38	0.00	1.00	0 b	0.15	0.00
NetworkManager.LateUpdate()	0.37	0.37	1.00	0 b	0.15	0.15
PostLateUpdate.ProfilerEndFrame	0.37	0.00	1.00	0 b	0.15	0.00
RenderLoop.CleanupNodeQueue	0.37	0.37	1.67	0 b	0.15	0.15
Shadows.CullingCallbacks	0.38	0.38	1.00	0 b	0.15	0.15
Profiler.CollectGlobalStats	0.37	0.00	1.00	0 b	0.15	0.01
SkinnedMeshFinalizeUpdate	0.36	0.12	1.00	0 b	0.14	0.05
FinalizeUpdateRendererBoundingVolumes	0.30	0.00	1.00	0 b	0.12	0.00
MouseController.Update()	0.28	0.13	1.00	0 b	0.12	0.06
Animators.SortWriteJob	0.24	0.24	2.00	0 b	0.11	0.11
Render.Prepare	0.26	0.26	1.00	0 b	0.11	0.11
PostLateUpdate.DirectorLateUpdate	0.24	0.23	1.00	0 b	0.10	0.10
DepthPass.Sort	0.23	0.23	1.00	0 b	0.10	0.10
Shadows.Sort	0.24	0.24	2.00	0 b	0.10	0.10
UpdateFunction.Invoke()	0.25	0.24	1.00	3.2 KB	0.10	0.09

File path
\profiler_data.json Open Edit

Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics: Average Values — Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Profiler.CollectDrawStats	0.19	0.19	1.00	0 b	0.09	0.09
PrepareSceneNodesJob	0.14	0.14	1.35	0 b	0.07	0.07
Batch.DrawDynamic	0.14	0.14	22.11	0 b	0.07	0.07
AbilityUpdateSystem	0.14	0.00	1.00	6.31 KB	0.07	0.00
RenderForwardOpaque.CollectShadows	0.13	0.00	1.00	0 b	0.06	0.01
AnimationViewSystem	0.13	0.00	1.00	2.87 KB	0.06	0.00
Director.PrepareFrame	0.16	0.02	1.26	0 b	0.06	0.01
EventSystem.Update()	0.11	0.11	1.00	0 b	0.05	0.05
Animator.ApplyBuiltinRootMotion	0.12	0.12	400.00	0 b	0.05	0.05
PrepareSceneNodes	0.08	0.08	1.00	0 b	0.04	0.04
Shadows.CollectShadows	0.08	0.01	1.00	0 b	0.04	0.02
EarlyUpdate.DirectorSampleTime	0.07	0.00	1.00	0 b	0.04	0.00
Director.SampleTime	0.06	0.06	1.00	0 b	0.04	0.04
Profiler.CollectMemoryAllocationStats	0.07	0.07	1.00	0 b	0.04	0.04
Animators.FireAnimationEventsAndBehaviours	0.09	0.03	1.00	0 b	0.04	0.03
TempAlloc.Overflow	0.05	0.00	2.00	0 b	0.03	0.00
Canvas.RenderOverlays	0.02	0.00	2.00	0 b	0.03	0.01
UGUI.Rendering.RenderOverlays	0.04	0.00	1.00	0 b	0.03	0.00
UIEvents.CanvasManagerRenderOverlays	0.04	0.00	1.00	0 b	0.03	0.00
PathFinderSystem	0.02	0.00	1.00	0 b	0.03	0.00
JobAlloc.Overflow	0.05	0.05	1.50	0 b	0.03	0.03
NetworkAgent.Update()	0.02	0.02	400.00	0 b	0.03	0.03
Render.Mesh	0.03	0.03	485.34	0 b	0.03	0.03
Render.TransparentGeometry	0.02	0.00	1.00	0 b	0.02	0.01
TransformSystem	0.00	0.00	1.00	0 b	0.02	0.00
Material.SetPassUncached	0.02	0.02	5.67	0 b	0.02	0.02
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.02	0.00
CullAllVisibleLights	0.03	0.00	1.00	0 b	0.02	0.00
GUIUtility.BeginGUI()	0.01	0.01	3.00	400 B	0.02	0.02
Graphics.Blit	0.01	0.01	1.00	0 b	0.02	0.02
PostLateUpdate.PlayerUpdateCanvases	0.01	0.00	1.00	0 b	0.01	0.00

File path
\profiler_data.json Open Edit

Export
Profiler Data | Current Frame Data | Selected Function Data

Figure B.29: DOD DwarfHeim profiler average elapsed time statistics for test with animations and models activated

Profiler Data E:
Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	54.20	0.40	1.00	400 B	29.75	0.15
Camera.Render	53.90	0.50	1.00	0 b	29.58	0.23
Drawing	42.90	0.40	1.00	0 b	23.53	0.13
Render.OpaqueGeometry	42.60	0.20	1.00	0 b	23.41	0.08
RenderForwardOpaque.Render	37.20	0.20	1.00	0 b	20.43	0.07
SendMouseEvents.DoSendMouseEvents()	35.00	0.20	1.00	0 b	16.30	0.07
Monobehaviour.OnMouse_	35.00	0.00	1.00	0 b	16.30	0.01
PreUpdate.SendMouseEvents	35.00	0.00	1.00	0 b	16.30	0.00
Physics.Raycast	34.90	1.70	1.00	0 b	16.26	0.68
RenderLoop.CleanupNodeQueue	25.80	25.80	3.00	0 b	14.15	14.15
Physics.SyncRigidbodyTransform	30.00	28.90	1.00	0 b	13.90	13.45
UpdateFunction.Invoke()	26.10	25.70	1.00	461.3 KB	12.21	12.03
MovingTestSystem	26.10	0.00	1.00	461.3 KB	12.21	0.00
BehaviourUpdate	26.30	0.70	1.00	199.5 KB	12.07	0.29
Update.ScriptRunBehaviourUpdate	26.30	0.00	1.00	199.5 KB	12.07	0.00
AstarPath.Update()	25.10	19.50	1.00	199.2 KB	11.63	8.31
Calling Path Callbacks	21.30	18.80	1.00	170.3 KB	9.90	8.73
GameAgentSystem	24.90	0.00	1.00	48.3 KB	9.67	0.00
WaitForJobGroupID	24.20	24.20	3.00	10.8 KB	9.50	9.50
BasicChainUpdateSystem	18.60	0.00	1.00	243 KB	9.13	0.01
RVOSimulator.Update()	22.80	22.80	1.00	0 b	9.08	9.08
Animators.Update	18.90	1.40	1.00	0 b	7.03	0.50
Director.ProcessFrame	18.90	0.00	3.00	0 b	7.03	0.00
PreLateUpdate.DirectorUpdateAnimationEnd	18.90	0.00	1.00	0 b	7.03	0.02
PostLateUpdate.UpdateAllSkinnedMeshes	15.80	0.20	1.00	0 b	6.92	0.08
MeshSkinning.Update	15.70	1.20	1.00	0 b	6.86	0.47
MeshSkinning.Prepare	15.00	4.90	1.00	0 b	6.49	1.96
Physics.SyncColliderTransform	13.30	12.50	1.00	0 b	6.31	5.92
RenderForward.RenderLoopJob	14.20	10.40	1.00	0 b	6.11	4.48
Culling	13.90	9.20	1.00	0 b	5.48	3.16
SceneCulling	13.80	10.60	1.00	0 b	5.45	4.50

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
CullResults.CreateSharedRendererScene	13.20	12.20	1.00	0 b	5.40	4.61
CullSendEvents	13.70	9.60	1.00	0 b	5.24	3.55
Shadows.RenderShadowMap	12.60	0.40	1.00	0 b	5.07	0.15
UpdateDepthTexture	13.40	8.70	1.00	0 b	5.03	3.14
PrepareSceneNodesJob	11.70	11.70	2.00	0 b	5.02	5.02
DepthPass.Job	11.80	6.80	1.00	0 b	4.75	2.50
FinalizeUpdateRendererBoundingVolumes	12.10	0.00	1.00	0 b	4.65	0.00
SkinnedMeshFinalizeUpdate	12.10	1.40	1.00	0 b	4.64	0.59
PrepareSceneNodesCombineJob	10.90	0.00	1.00	0 b	4.63	0.00
PrepareSceneNodes	10.70	10.70	1.00	0 b	4.58	4.58
MeshSkinning.CalcMatrices	10.00	10.00	2000.00	0 b	4.51	4.51
Shadows.RenderJob	10.30	0.00	2.00	0 b	4.13	0.00
Shadows.RenderJobDir	10.30	5.10	2.00	0 b	4.13	2.30
Shadows.PrepareShadowmap	8.10	4.80	1.00	0 b	3.53	1.86
BatchRenderer.Flush	8.60	8.20	2293.00	0 b	3.46	3.33
SkinnedMeshUpdateAllNeeded	7.00	7.00	4.00	0 b	3.22	3.22
MeshSkinning.Render	7.90	7.90	1979.00	0 b	3.17	3.17
ViewSynchronizationSystem	7.90	0.00	1.00	0 b	2.99	0.00
PreLateUpdate.DirectorUpdateAnimationBegin	7.40	0.00	1.00	0 b	2.86	0.00
RenderForwardOpaque.Prepare	8.10	4.20	1.00	0 b	2.70	1.50
Render.Prepare	6.60	6.60	1.00	0 b	2.55	2.55
CullAllVisibleLights	5.70	0.10	1.00	0 b	2.42	0.04
Shadows.CullDirectionalShadowCasters	5.60	5.60	1.00	0 b	2.39	2.39
Batch.DrawDynamic	5.80	5.80	28.00	0 b	2.36	2.35
Animators.IKJob	7.30	5.00	2.00	0 b	2.28	2.08
Animator.EvaluateIK	6.30	6.30	71.00	0 b	2.06	2.06
PostLateUpdate.UpdateAllRenderers	4.80	0.00	1.00	0 b	1.85	0.00
UpdateRendererBoundingVolumes	4.70	3.90	11.00	0 b	1.82	1.31
Animators.WriteJob	5.20	3.20	3.00	0 b	1.79	1.45
Running Path Modifiers	4.30	4.30	144.00	8.7 KB	1.60	1.60
ExtractRenderNodeQueue	4.50	4.50	5.00	0 b	1.54	1.54

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Animator.WriteAnimatedValues	4.10	4.10	65.00	0 b	1.52	1.52
Animators.PrepareSecondPass	2.90	2.90	1.00	0 b	1.37	1.37
Shadows.CullingCallbacks	3.20	3.20	1.00	0 b	1.06	1.06
AbilityAgentSystem	2.40	0.00	1.00	57.2 KB	1.05	0.00
PathFinderSystem:PathfinderJob	2.00	0.00	1.00	10.8 KB	1.02	0.00
ExecuteJobFunction.Invoke()	2.00	2.00	1.00	10.8 KB	1.02	1.01
JobAlloc.Overflow	2.60	2.60	2.00	0 b	0.99	0.99
Animators.PrepareFirstPass	2.20	2.20	1.00	0 b	0.92	0.92
Animators.SortWriteJob	1.90	1.90	2.00	0 b	0.90	0.90
MoveSystem	2.30	0.00	1.00	6 KB	0.88	0.00
Animators.RetargeterJob	2.00	1.80	1.00	0 b	0.85	0.68
Animators.ProcessRootMotionJob	2.20	2.20	1.00	0 b	0.83	0.77
FPSCounter.Update()	2.00	2.00	1.00	212 B	0.82	0.82
LateBehaviourUpdate	2.00	0.00	1.00	0 b	0.79	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	2.00	0.00	1.00	0 b	0.79	0.00
NetworkManager.LateUpdate()	2.00	2.00	1.00	0 b	0.78	0.78
Animators.ProcessAnimationsJob	2.20	2.20	1.00	0 b	0.78	0.77
Shadows.PrepareJob	2.00	0.00	1.00	0 b	0.75	0.00
AbilityUpdateSystem	1.70	0.00	1.00	81.2 KB	0.75	0.00
Animator.EvaluateRetargeter	1.90	1.90	66.00	0 b	0.71	0.71
DestroyCullResults	2.10	0.00	1.00	0 b	0.67	0.01
Animator.ProcessRootMotion	1.90	1.90	70.00	0 b	0.66	0.66
Director.PrepareFrame	1.70	1.60	3.00	0 b	0.66	0.66
Animator.ProcessAnimations	1.90	1.90	67.00	0 b	0.64	0.64
CombineJobResults	1.50	1.50	2.00	0 b	0.60	0.60
AnimationViewSystem	1.50	0.00	1.00	25.8 KB	0.60	0.00
TransformChangedDispatch	1.60	1.40	4.00	0 b	0.60	0.60
Animators.DirtySceneObjects	1.60	1.60	1.00	0 b	0.59	0.59
Animators.ApplyOnAnimatorMove	1.50	0.50	1.00	0 b	0.58	0.19
PhotonHandler.Update()	1.50	0.00	1.00	0 b	0.56	0.02
SendOutgoingCommands	1.40	1.40	11.00	0 b	0.54	0.54

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Director.PrepareFrameJob	1.40	0.90	74.00	0 b	0.52	0.32
PostLateUpdate.ProfilerEndFrame	1.30	0.00	1.00	0 b	0.48	0.02
Profiler.CollectGlobalStats	1.30	0.10	1.00	0 b	0.48	0.05
Shadows.ExtractCasters	1.10	1.10	1.00	0 b	0.44	0.44
UIEvents.IMGUIRenderOverlays	1.30	0.00	1.00	400 B	0.44	0.00
GUI.Repaint	1.30	0.50	1.00	400 B	0.44	0.21
Animator.ApplyOnAnimatorMove	1.00	0.80	400.00	0 b	0.43	0.32
MouseController.Update()	0.90	0.50	1.00	0 b	0.42	0.19
AnimatorControllerPlayable.PrepareFrame	1.10	1.10	74.00	0 b	0.39	0.39
RenderForwardOpaque.CollectShadows	0.90	0.10	1.00	0 b	0.36	0.04
Shadows.CullDirectionalCascades	0.90	0.90	1.00	0 b	0.33	0.33
Profiler.CollectDrawStats	1.10	1.10	1.00	0 b	0.33	0.33
Shadows.CollectShadows	0.80	0.30	1.00	0 b	0.32	0.12
EventSystem.Update()	0.80	0.80	1.00	0 b	0.31	0.31
PostLateUpdate.DirectorLateUpdate	0.70	0.70	1.00	0 b	0.31	0.31
GC.Alloc	0.60	0.60	11600.00	461.3 KB	0.27	0.27
CullSceneDynamicObjects	0.60	0.30	3.00	0 b	0.26	0.11
Material.SetPassFast	0.70	0.30	18.00	0 b	0.26	0.13
Animator.ApplyBuiltinRootMotion	0.70	0.70	400.00	0 b	0.25	0.25
CullObjectsWithoutUmbra	0.60	0.60	3.00	0 b	0.25	0.25
UIEvents.CanvasManagerRenderOverlays	0.50	0.00	1.00	0 b	0.21	0.00
Render.Mesh	0.50	0.50	736.00	0 b	0.20	0.20
EarlyUpdate.PollPlayerConnection	0.50	0.00	1.00	0 b	0.20	0.00
UGUI.Rendering.RenderOverlays	0.50	0.10	1.00	0 b	0.20	0.04
ProfilerConnection Poll	0.50	0.50	1.00	0 b	0.20	0.20
DepthPass.Sort	0.50	0.50	1.00	0 b	0.20	0.20
Shadows.Sort	0.50	0.50	2.00	0 b	0.19	0.19
Profiler.CollectMemoryAllocationStats	0.30	0.30	1.00	0 b	0.16	0.16
Canvas.RenderOverlays	0.40	0.30	2.00	0 b	0.16	0.11
UGUI.Rendering.UpdateBatches	0.30	0.00	1.00	0 b	0.15	0.02
UIEvents.WillRenderCanvases	0.30	0.00	1.00	0 b	0.15	0.00

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Figure B.30: DOD DwarfHeim profiler maximum elapsed time statistics for test with animations and models activated

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Monobehaviour.OnMouse_	13.50	0.00	1.00	0 b	2.20	0.00
SendMouseEvents.DoSendMouseEvents()	13.50	0.00	1.00	0 b	2.20	0.01
PreUpdate.SendMouseEvents	13.50	0.00	1.00	0 b	2.20	0.00
ViewSynchronizationSystem	8.10	0.00	1.00	0 b	1.37	0.00
MoveSystem	2.40	0.00	1.00	0 b	0.39	0.00
PostLateUpdate.FinishFrameRendering	2.30	0.00	1.00	400 B	0.37	0.00
Camera.Render	1.50	0.10	1.00	0 b	0.24	0.01
Update.ScriptRunBehaviourUpdate	1.80	0.00	1.00	96 B	0.24	0.00
BehaviourUpdate	1.80	0.60	1.00	96 B	0.24	0.09
GameAgentSystem	1.00	0.00	1.00	0 b	0.14	0.00
Drawing	0.70	0.00	1.00	0 b	0.11	0.00
UIEvents.IMGUIRenderOverlays	0.60	0.00	1.00	400 B	0.09	0.00
AbilityAgentSystem	0.60	0.00	1.00	0 b	0.09	0.00
GUI.Repaint	0.60	0.20	1.00	400 B	0.09	0.04
Render.OpaqueGeometry	0.50	0.00	1.00	0 b	0.09	0.00
RenderForwardOpaque.Render	0.40	0.00	1.00	0 b	0.07	0.00
MouseController.Update()	0.40	0.20	1.00	0 b	0.04	0.01
EventSystem.Update()	0.20	0.20	1.00	0 b	0.03	0.03
AbilityUpdateSystem	0.10	0.00	1.00	0 b	0.03	0.00
PostLateUpdate.ProfilerEndFrame	0.20	0.00	1.00	0 b	0.03	0.00
Profiler.CollectGlobalStats	0.20	0.00	1.00	0 b	0.03	0.00
Culling	0.20	0.00	1.00	0 b	0.03	0.00
UpdateDepthTexture	0.20	0.10	1.00	0 b	0.03	0.01
RenderForwardOpaque.CollectShadows	0.10	0.00	1.00	0 b	0.02	0.00
SceneCulling	0.10	0.00	1.00	0 b	0.02	0.00
Running Path Modifiers	0.20	0.20	1.00	348 B	0.02	0.02
MeshInstanceRendererSystem	0.10	0.00	1.00	0 b	0.02	0.00
Profiler.CollectMemoryAllocationStats	0.10	0.10	1.00	0 b	0.02	0.02
Physics.Raycast	0.20	0.20	1.00	0 b	0.02	0.02
JobAlloc.Overflow	0.10	0.10	2.00	0 b	0.01	0.01
GUIUtility.BeginGUI()	0.00	0.00	3.00	400 B	0.01	0.01

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.PrepareShadowmap	0.00	0.00	1.00	0 b	0.01	0.00
CullAllVisibleLights	0.00	0.00	1.00	0 b	0.01	0.00
Shadows.CollectShadows	0.10	0.00	1.00	0 b	0.01	0.01
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.01	0.00
CullResults.CreateSharedRendererScene	0.00	0.00	1.00	0 b	0.01	0.00
Shadows.RenderShadowMap	0.10	0.00	1.00	0 b	0.01	0.00
PathFinderSystem	0.10	0.00	1.00	0 b	0.01	0.00
DepthPass.Job	0.10	0.00	1.00	0 b	0.01	0.00
TransformSystem	0.10	0.00	1.00	0 b	0.01	0.00
UIEvents.CanvasManagerRenderOverlays	0.10	0.00	1.00	0 b	0.01	0.00
UGUI.Rendering.RenderOverlays	0.10	0.00	1.00	0 b	0.01	0.00
Canvas.RenderOverlays	0.00	0.00	2.00	0 b	0.01	0.00
TempAlloc.Overflow	0.10	0.00	2.00	0 b	0.01	0.00
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	0.00	0.00	1.00	0 b	0.01	0.00
NetworkAgent.Update()	0.10	0.10	400.00	0 b	0.01	0.01
MouseInputSystem	0.00	0.00	1.00	0 b	0.00	0.00
MouseSelectionSystem	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.PerformanceAnalyticsUpdate	0.00	0.00	1.00	0 b	0.00	0.00
MoveForward2DSystem	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.ClearImmediateRenderers	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.MemoryFrameMaintenance	0.00	0.00	1.00	0 b	0.00	0.00
AutoAttack	0.00	0.00	1.00	0 b	0.00	0.00
PlayerEndOfFrame	0.00	0.00	1.00	0 b	0.00	0.00
CoroutinesDelayedCalls	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.TriggerEndOfFrameCallbacks	0.00	0.00	1.00	0 b	0.00	0.00
FrameEvents.XREndFrame	0.00	0.00	1.00	0 b	0.00	0.00
AnimationViewSystem	0.00	0.00	1.00	0 b	0.00	0.00
Director.ProcessFrame	0.00	0.00	1.00	0 b	0.00	0.00
Director.PrepareFrame	0.00	0.00	1.00	0 b	0.00	0.00
Update.DirectorUpdate	0.00	0.00	1.00	0 b	0.00	0.00
Transform2DSystem	0.00	0.00	1.00	0 b	0.00	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.31: DOD DwarfHeim profiler minimum elapsed time statistics for test with animation and models deactivated

Profiler Data E

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
SendMouseEvents.DoSendMouseEvents()	40.16	0.21	1.00	0 b	2.80	0.01
PreUpdate.SendMouseEvents	40.18	0.00	1.00	0 b	2.80	0.00
Monobehaviour.OnMouse_	40.18	0.00	1.00	0 b	2.80	0.00
ViewSynchronizationSystem	22.53	0.00	1.00	0 b	1.56	0.00
Physics.Raycast	20.22	3.10	1.00	0 b	1.41	0.22
Update.ScriptRunBehaviourUpdate	10.00	0.00	1.00	7.86 KB	0.98	0.00
BehaviourUpdate	9.98	1.68	1.00	7.86 KB	0.98	0.12
Physics.SyncColliderTransform	10.53	8.94	1.00	0 b	0.73	0.62
Running Path Modifiers	5.89	5.89	53.33	3.41 KB	0.64	0.64
MoveSystem	7.03	0.00	1.00	484.35 B	0.48	0.00
PostLateUpdate.FinishFrameRendering	6.54	0.09	1.00	400 B	0.45	0.00
Physics.SyncRigidbodyTransform	6.54	4.29	1.00	0 b	0.45	0.30
RVOSimulator.Update()	3.47	3.47	1.00	0 b	0.45	0.45
Camera.Render	4.15	0.27	1.00	0 b	0.29	0.02
GameAgentSystem	3.14	0.00	1.00	1.16 KB	0.22	0.00
AstarPath.Update()	1.71	0.95	1.00	7.69 KB	0.19	0.10
TransformChangedDispatch	2.36	1.15	1.62	0 b	0.16	0.08
AbilityAgentSystem	1.87	0.00	1.00	3.04 KB	0.14	0.00
Drawing	1.91	0.07	1.00	0 b	0.13	0.00
UIEvents.IMGUIRenderOverlays	1.69	0.00	1.00	400 B	0.12	0.00
GUI.Repaint	1.67	0.80	1.00	400 B	0.11	0.05
Render.OpaqueGeometry	1.53	0.01	1.00	0 b	0.10	0.00
MouseController.Update()	1.27	0.68	1.00	0 b	0.09	0.05
Calling Path Callbacks	0.74	0.14	1.00	6.35 KB	0.08	0.02
BasicChainUpdateSystem	0.81	0.00	1.00	9.59 KB	0.08	0.00
WaitForJobGroupID	1.16	0.24	1.00	0 b	0.08	0.02
RenderForwardOpaque.Render	1.18	0.09	1.00	0 b	0.08	0.00
UpdateFunction.Invoke()	0.99	0.97	1.00	625.12 B	0.07	0.07
MovingTestSystem	0.46	0.00	1.00	6.15 KB	0.06	0.00
Profiler.CollectGlobalStats	0.71	0.09	1.00	0 b	0.05	0.00
PostLateUpdate.ProfilerEndFrame	0.72	0.00	1.00	0 b	0.05	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Culling	0.72	0.16	1.00	0 b	0.05	0.01
UpdateDepthTexture	0.56	0.26	1.00	0 b	0.04	0.01
EventSystem.Update()	0.64	0.64	1.00	0 b	0.04	0.04
AbilityUpdateSystem	0.59	0.00	1.00	1.75 KB	0.04	0.00
PostLateUpdate.PlayerUpdateCanvases	0.58	0.00	1.00	0 b	0.04	0.00
UIEvents.WillRenderCanvases	0.57	0.00	1.00	0 b	0.04	0.00
UGUI.Rendering.UpdateBatches	0.56	0.00	1.00	0 b	0.04	0.00
SceneCulling	0.40	0.07	1.00	0 b	0.03	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.36	0.00	1.00	21.5 B	0.03	0.00
ExecuteJobFunction.Invoke()	0.37	0.37	1.08	0 b	0.03	0.03
LateBehaviourUpdate	0.36	0.00	1.00	21.5 B	0.03	0.00
NetworkManager.LateUpdate()	0.36	0.36	1.00	21.5 B	0.03	0.03
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	0.39	0.00	1.08	0 b	0.03	0.00
Canvas.SendWillRenderCanvases()	0.43	0.00	1.00	0 b	0.03	0.00
Profiler.CollectMemoryAllocationStats	0.45	0.45	1.00	0 b	0.03	0.03
JobAlloc.Overflow	0.41	0.41	2.00	0 b	0.03	0.03
TempAlloc.Overflow	0.41	0.00	2.00	0 b	0.03	0.00
TransformSystem	0.40	0.00	1.00	0 b	0.03	0.00
NetworkAgent.Update()	0.47	0.47	400.00	0 b	0.03	0.03
Layout	0.43	0.40	1.00	0 b	0.03	0.03
PathFinderSystem	0.34	0.00	1.00	0 b	0.02	0.00
MeshInstanceRendererSystem	0.39	0.00	1.00	0 b	0.02	0.00
SendOutgoingCommands	0.23	0.23	1.42	0 b	0.02	0.02
RenderForwardOpaque.CollectShadows	0.36	0.01	1.00	0 b	0.02	0.00
Shadows.RenderShadowMap	0.28	0.11	1.00	0 b	0.02	0.01
Shadows.CollectShadows	0.29	0.15	1.00	0 b	0.02	0.01
UGUI.Rendering.RenderOverlays	0.22	0.00	1.00	0 b	0.01	0.00
UIEvents.CanvasManagerRenderOverlays	0.22	0.00	1.00	0 b	0.01	0.00
DepthPass.Job	0.24	0.09	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.18	0.00	1.00	0 b	0.01	0.00
EnlightenRuntimeManager.PostUpdate	0.13	0.09	1.00	0 b	0.01	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.32: DOD DwarfHeim profiler average elapsed time statistics for test with animations and models deactivated

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Update.ScriptRunBehaviourUpdate	62.50	0.00	1.00	170.9 KB	12.08	0.00
BehaviourUpdate	62.50	4.30	1.00	170.9 KB	12.08	0.35
UpdateFunction.Invoke()	62.00	61.00	1.00	461.3 KB	10.51	10.35
MovingTestSystem	62.00	0.00	1.00	461.3 KB	10.51	0.00
RVOSimulator.Update()	55.80	55.80	1.00	0 b	8.88	8.88
Monobehaviour.OnMouse_	51.20	0.00	1.00	0 b	4.95	0.00
PreUpdate.SendMouseEvents	51.20	0.00	1.00	0 b	4.95	0.00
SendMouseEvents.DoSendMouseEvents()	51.10	0.40	1.00	0 b	4.94	0.04
Physics.Raycast	50.80	12.10	1.00	0 b	4.91	1.05
AstarPath.Update()	30.50	16.50	1.00	170.6 KB	3.55	1.93
Physics.SyncColliderTransform	29.70	26.10	1.00	0 b	2.99	2.64
ViewSynchronizationSystem	30.50	0.00	1.00	0 b	2.52	0.00
GameAgentSystem	23.60	0.00	1.00	34 KB	2.41	0.00
WaitForJobGroupID	21.90	21.90	1.00	0 b	2.20	2.20
Physics.SyncRigidbodyTransform	20.40	13.80	1.00	0 b	1.72	1.23
BasicChainUpdateSystem	15.20	0.00	1.00	196.9 KB	1.67	0.00
Calling Path Callbacks	14.50	2.70	1.00	141 KB	1.65	0.30
Running Path Modifiers	11.90	11.90	119.00	7.2 KB	1.33	1.32
MoveSystem	11.90	0.00	1.00	4.3 KB	0.85	0.00
PostLateUpdate.FinishFrameRendering	10.50	0.30	1.00	400 B	0.83	0.02
AbilityAgentSystem	6.50	0.00	1.00	43.9 KB	0.77	0.00
PhotonHandler.Update()	8.90	0.20	1.00	0 b	0.61	0.01
NetworkManager.LateUpdate()	4.90	4.90	1.00	6.3 KB	0.59	0.59
LateBehaviourUpdate	4.90	0.00	1.00	6.3 KB	0.59	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	4.90	0.00	1.00	6.3 KB	0.59	0.00
SendOutgoingCommands	8.60	8.60	19.00	0 b	0.59	0.59
Camera.Render	6.90	1.20	1.00	0 b	0.55	0.09
AbilityUpdateSystem	5.50	0.00	1.00	63.5 KB	0.51	0.00
TransformChangedDispatch	6.30	5.70	4.00	0 b	0.43	0.36
MouseController.Update()	3.50	1.60	1.00	0 b	0.27	0.12
UIEvents.IMGUIRenderOverlays	3.20	0.00	1.00	400 B	0.25	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
GUI.Repaint	3.20	1.80	1.00	400 B	0.24	0.13
UGUI.Rendering.UpdateBatches	3.50	0.10	1.00	0 b	0.24	0.01
Profiler.CollectGlobalStats	3.00	0.30	1.00	0 b	0.24	0.02
PostLateUpdate.ProfilerEndFrame	3.10	0.00	1.00	0 b	0.24	0.00
PostLateUpdate.PlayerUpdateCanvases	3.50	0.00	1.00	0 b	0.24	0.00
NetworkAgent.Update()	3.00	3.00	400.00	0 b	0.24	0.24
UIEvents.WillRenderCanvases	3.50	0.00	1.00	0 b	0.24	0.00
PathFinderSystem	2.80	0.00	1.00	0 b	0.23	0.00
Canvas.SendWillRenderCanvases()	3.20	0.00	1.00	0 b	0.22	0.00
Layout	3.20	3.10	1.00	0 b	0.22	0.21
Profiler.CollectMemoryAllocationStats	2.70	2.70	1.00	0 b	0.21	0.21
Drawing	3.30	0.20	1.00	0 b	0.20	0.01
GC.Alloc	1.80	1.80	11600.00	461.3 KB	0.17	0.17
Render.OpaqueGeometry	2.70	0.10	1.00	0 b	0.16	0.01
Culling	2.10	0.50	1.00	0 b	0.16	0.04
ProfilerConnection Poll	2.00	2.00	1.00	0 b	0.15	0.15
EarlyUpdate.PollPlayerConnection	2.00	0.00	1.00	0 b	0.15	0.00
RenderForwardOpaque.Render	2.10	0.20	1.00	0 b	0.14	0.01
TransformSystem	1.50	0.00	1.00	0 b	0.12	0.00
ExecuteJobFunction.Invoke()	1.60	1.60	5.00	0 b	0.11	0.11
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	1.60	0.00	5.00	0 b	0.11	0.00
FPSDisplay.Update()	1.30	1.30	1.00	0 b	0.10	0.10
EventSystem.Update()	1.30	1.30	1.00	0 b	0.10	0.10
SceneCulling	1.20	0.30	1.00	0 b	0.09	0.02
UpdateDepthTexture	1.00	0.50	1.00	0 b	0.08	0.03
Profiler.CollectDrawStats	1.40	1.40	1.00	0 b	0.08	0.08
IdleUtility	0.80	0.00	1.00	6.9 KB	0.08	0.00
Canvas.RenderOverlays	1.20	1.00	2.00	0 b	0.07	0.06
UGUI.Rendering.RenderOverlays	1.20	0.00	1.00	0 b	0.07	0.00
UIEvents.CanvasManagerRenderOverlays	1.20	0.00	1.00	0 b	0.07	0.00
TempAlloc.Overflow	1.40	0.00	2.00	0 b	0.07	0.00

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
MeshInstanceRendererSystem	1.00	0.00	1.00	0 b	0.07	0.00
JobAlloc.Overflow	1.30	1.30	2.00	0 b	0.07	0.07
CullAllVisibleLights	0.60	0.20	1.00	0 b	0.05	0.01
IdleSystem	0.70	0.00	1.00	0 b	0.05	0.00
Material.SetPassFast	0.80	0.10	18.00	0 b	0.05	0.01
RenderForwardOpaque.CollectShadows	0.80	0.10	1.00	0 b	0.05	0.01
FPSCounter.Update()	0.70	0.70	1.00	220 B	0.05	0.04
Shadows.RenderShadowMap	0.50	0.20	1.00	0 b	0.04	0.02
PostLateUpdate.EnlightenRuntimeUpdate	0.60	0.00	1.00	0 b	0.04	0.00
RenderForward.RenderLoopJob	0.70	0.30	1.00	0 b	0.04	0.02
EnlightenRuntimeManager.PostUpdate	0.60	0.50	1.00	0 b	0.04	0.03
Material.SetPassUncached	0.70	0.70	14.00	0 b	0.04	0.04
CullResults.CreateSharedRendererScene	0.50	0.30	1.00	0 b	0.04	0.02
DepthPass.Job	0.50	0.20	1.00	0 b	0.04	0.01
Shadows.CullDirectionalShadowCasters	0.50	0.50	1.00	0 b	0.04	0.04
Shadows.CollectShadows	0.70	0.30	1.00	0 b	0.04	0.02
Camera.FindStacks	0.30	0.30	2.00	0 b	0.03	0.03
PathBarrier	0.60	0.00	1.00	4.8 KB	0.03	0.00
Canvas.BuildBatch	0.40	0.40	1.00	0 b	0.03	0.03
GUIUtility.BeginGUI()	0.40	0.40	3.00	400 B	0.03	0.03
Shadows.PrepareShadowmap	0.40	0.30	1.00	0 b	0.03	0.02
PostLateUpdate.UpdateCustomRenderTextures	0.30	0.00	1.00	0 b	0.02	0.00
CanvasScaler.Update()	0.20	0.20	2.00	0 b	0.02	0.02
EarlyUpdate.UpdatePreloading	0.30	0.00	1.00	0 b	0.02	0.00
WatermarkRender	0.30	0.10	1.00	0 b	0.02	0.01
CullPerObjectLights	0.30	0.30	1.00	0 b	0.02	0.02
PostLateUpdate.PlayerSendFramePostPresent	0.30	0.00	1.00	0 b	0.02	0.00
UpdatePreloading	0.30	0.00	1.00	0 b	0.02	0.00
CustomRenderTextures.Update	0.30	0.30	1.00	0 b	0.02	0.02
Application.Integrate Assets in Background	0.30	0.00	1.00	0 b	0.02	0.00
Preload Single Step	0.30	0.30	1.00	0 b	0.02	0.02

File path

\profiler_data.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.33: DOD DwarfHeim profiler maximum elapsed time statistics for test with animations and models deactivated

Profiler Data E:

Statistics Min Values | Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
MoveSystem	3.30	0.00	1.00	0 b	0.35	0.00
PostLateUpdate.FinishFrameRendering	3.60	0.00	1.00	400 B	0.27	0.00
Camera.Render	2.20	0.10	1.00	0 b	0.17	0.01
Update.ScriptRunBehaviourUpdate	2.00	0.00	1.00	96 B	0.16	0.00
BehaviourUpdate	2.00	0.70	1.00	96 B	0.16	0.05
GameAgentSystem	1.10	0.00	1.00	0 b	0.09	0.00
Drawing	1.10	0.00	1.00	0 b	0.08	0.00
Render.OpaqueGeometry	0.80	0.00	1.00	0 b	0.06	0.00
GUI.Repaint	0.70	0.30	1.00	400 B	0.06	0.03
UIEvents.IMGUIRenderOverlays	0.70	0.00	1.00	400 B	0.06	0.00
AbilityAgentSystem	0.90	0.00	1.00	0 b	0.05	0.00
PreUpdate.SendMouseEvents	0.60	0.00	1.00	0 b	0.05	0.00
Monobehaviour.OnMouse_	0.60	0.00	1.00	0 b	0.05	0.00
SendMouseEvents.DoSendMouseEvents()	0.60	0.00	1.00	0 b	0.05	0.00
MouseController.Update()	0.60	0.30	1.00	0 b	0.04	0.01
RenderForwardOpaque.Render	0.60	0.00	1.00	0 b	0.04	0.00
AbilityUpdateSystem	0.30	0.00	1.00	0 b	0.03	0.00
Physics.Raycast	0.30	0.30	1.00	0 b	0.02	0.02
EventSystem.Update()	0.20	0.20	1.00	0 b	0.02	0.02
NetworkAgent.Update()	0.10	0.10	400.00	0 b	0.02	0.02
Running Path Modifiers	1.20	1.10	1.00	614.4 B	0.02	0.02
UpdateDepthTexture	0.30	0.10	1.00	0 b	0.02	0.01
Culling	0.30	0.00	1.00	0 b	0.02	0.00
PostLateUpdate.ProfilerEndFrame	0.30	0.00	1.00	0 b	0.02	0.00
Profiler.CollectGlobalStats	0.30	0.00	1.00	0 b	0.02	0.00
MeshInstanceRendererSystem	0.20	0.00	1.00	0 b	0.01	0.00
RenderForwardOpaque.CollectShadows	0.10	0.00	1.00	0 b	0.01	0.00
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	0.10	0.00	1.00	0 b	0.01	0.00
SceneCulling	0.10	0.00	1.00	0 b	0.01	0.00
Profiler.CollectMemoryAllocationStats	0.10	0.10	1.00	0 b	0.01	0.01
CullResults.CreateSharedRendererScene	0.10	0.00	1.00	0 b	0.01	0.00

File path

\profiler_data.json | Open | Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.34: DOD DwarfHeim profiler minimum elapsed time statistics for test with animation and models deactivated + several object-oriented components removed such as transform position

Profiler Data E:

Statistics Average Values | Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Running Path Modifiers	11.76	11.70	57.14	3.66 KB	0.71	0.71
Update.ScriptRunBehaviourUpdate	17.84	0.00	1.00	2.04 KB	0.48	0.00
BehaviourUpdate	17.82	5.13	1.00	2.04 KB	0.48	0.10
MoveSystem	24.27	0.00	1.00	58.71 B	0.47	0.00
PostLateUpdate.FinishFrameRendering	19.51	0.24	1.00	400 B	0.38	0.00
Camera.Render	12.44	0.90	1.00	0 b	0.24	0.01
GameAgentSystem	8.76	0.00	1.00	369.32 B	0.17	0.00
RVOSimulator.Update()	1.47	1.47	1.00	0 b	0.13	0.13
Drawing	6.06	0.32	1.00	0 b	0.12	0.00
AbilityAgentSystem	5.52	0.00	1.00	722.94 B	0.11	0.00
PreUpdate.SendMouseEvents	5.70	0.00	1.00	0 b	0.11	0.00
Monobehaviour.OnMouse_	5.69	0.00	1.00	0 b	0.11	0.00
SendMouseEvents.DoSendMouseEvents()	5.66	0.52	1.00	0 b	0.11	0.01
UIEvents.IMGUIRenderOverlays	5.30	0.00	1.00	400 B	0.10	0.00
GUI.Repaint	5.28	2.44	1.00	400 B	0.10	0.04
MouseController.Update()	5.23	2.34	1.00	0 b	0.10	0.04
Render.OpaqueGeometry	4.85	0.16	1.00	0 b	0.09	0.00
Physics.Raycast	3.96	3.95	1.00	0 b	0.08	0.08
RenderForwardOpaque.Render	3.70	0.39	1.00	0 b	0.07	0.00
AstarPath.Update()	0.97	0.58	1.00	1.92 KB	0.05	0.02
Culling	2.11	0.47	1.00	0 b	0.04	0.00
MovingTestSystem	0.47	0.00	1.00	1.54 KB	0.04	0.00
Profiler.CollectGlobalStats	2.16	0.38	1.00	0 b	0.04	0.00
PostLateUpdate.ProfilerEndFrame	2.18	0.00	1.00	0 b	0.04	0.00
EventSystem.Update()	2.05	2.05	1.00	0 b	0.04	0.04
UpdateDepthTexture	1.57	0.76	1.00	0 b	0.03	0.01
AbilityUpdateSystem	1.92	0.00	1.00	527.02 B	0.03	0.00
NetworkAgent.Update()	1.70	1.70	400.00	0 b	0.03	0.03
UpdateFunction.Invoke()	1.21	1.15	1.00	156.2 B	0.02	0.02
Calling Path Callbacks	0.35	0.07	1.00	1.58 KB	0.02	0.00
BasicChainUpdateSystem	0.40	0.00	1.00	2.38 KB	0.02	0.00

File path

\profiler_data.json | Open | Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Average Values | Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
ExecuteJobFunction.Invoke()	1.17	1.17	1.07	0 b	0.02	0.02
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	1.21	0.00	1.07	0 b	0.02	0.00
SceneCulling	1.30	0.28	1.00	0 b	0.02	0.00
Profiler.CollectMemoryAllocationStats	1.41	1.41	1.00	0 b	0.02	0.02
TransformSystem	1.43	0.00	1.00	0 b	0.02	0.00
MeshInstanceRendererSystem	1.27	0.00	1.00	0 b	0.02	0.00
RenderForwardOpaque.CollectShadows	1.08	0.16	1.00	0 b	0.02	0.00
Shadows.RenderShadowMap	1.01	0.52	1.00	0 b	0.02	0.01
NetworkManager.LateUpdate()	0.18	0.18	1.00	0 b	0.01	0.01
DepthPass.Job	0.71	0.31	1.00	0 b	0.01	0.00
CullResults.CreateSharedRendererScene	0.86	0.42	1.00	0 b	0.01	0.00
UGUI.Rendering.RenderOverlays	0.74	0.02	1.00	0 b	0.01	0.00
GUIUtility.BeginGUI()	0.57	0.56	3.00	400 B	0.01	0.01
UIEvents.CanvasManagerRenderOverlays	0.76	0.00	1.00	0 b	0.01	0.00
Canvas.RenderOverlays	0.66	0.33	2.00	0 b	0.01	0.00
Canvas.BuildBatch	0.48	0.48	1.00	0 b	0.01	0.01
PathFinderSystem	0.97	0.00	1.00	0 b	0.01	0.00
LateBehaviourUpdate	0.20	0.00	1.00	0 b	0.01	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.25	0.00	1.00	0 b	0.01	0.00
WaitForJobGroupID	0.61	0.42	1.00	0 b	0.01	0.01
Shadows.CollectShadows	0.87	0.51	1.00	0 b	0.01	0.01
Shadows.PrepareShadowmap	0.77	0.52	1.00	0 b	0.01	0.01
Material.SetPassFast	0.40	0.04	4.00	0 b	0.01	0.00
CullAllVisibleLights	0.57	0.13	1.00	0 b	0.01	0.00
Material.SetPassUncached	0.67	0.67	5.67	0 b	0.01	0.01
RenderForward.RenderLoopJob	0.51	0.32	1.00	0 b	0.01	0.00
PreLateUpdate.ConstraintManagerUpdate	0.00	0.00	1.00	0 b	0.00	0.00
AudioManager.Update	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.UpdateAudio	0.00	0.00	1.00	0 b	0.00	0.00
FrameEvents.XREndFrame	0.00	0.00	1.00	0 b	0.00	0.00
AddCustomActiveLocalLights	0.00	0.00	1.00	0 b	0.00	0.00

File path

\profiler_data.json | Open | Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.35: DOD DwarfHeim profiler minimum elapsed time statistics for test with animation and models deactivated + several object-oriented components removed such as transform position

Profiler Data E: | Statistics: Max Values | Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
UpdateFunction.Invoke()	82.40	81.20	1.00	461.3 KB	11.33	11.16
MovingTestSystem	82.40	0.00	1.00	461.3 KB	11.33	0.00
BehaviourUpdate	83.40	7.20	1.00	200.4 KB	9.51	0.17
Update.ScriptRunBehaviourUpdate	83.40	0.00	1.00	200.4 KB	9.51	0.00
RVOSimulator.Update()	80.20	80.20	1.00	0 b	9.15	9.15
AstarPath.Update()	51.50	29.70	1.00	200.1 KB	3.98	1.92
Calling Path Callbacks	26.30	5.00	1.00	171.9 KB	2.05	0.39
BasicChainUpdateSystem	28.40	0.00	1.00	240.4 KB	1.73	0.00
GameAgentSystem	19.10	0.00	1.00	55.3 KB	1.72	0.00
Running Path Modifiers	21.10	21.10	144.00	8.8 KB	1.65	1.65
WaitForJobGroupID	17.80	17.80	1.00	0 b	1.60	1.60
MoveSystem	31.00	0.00	1.00	4.3 KB	0.90	0.00
AbilityAgentSystem	11.20	0.00	1.00	54 KB	0.87	0.00
PostLateUpdate.FinishFrameRendering	28.40	0.50	1.00	400 B	0.64	0.01
LateBehaviourUpdate	8.40	0.20	1.00	0 b	0.51	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	8.40	0.10	1.00	0 b	0.51	0.00
NetworkManager.LateUpdate()	8.30	8.30	1.00	0 b	0.50	0.50
AbilityUpdateSystem	3.40	0.00	1.00	72 KB	0.47	0.00
Camera.Render	19.60	3.90	1.00	0 b	0.40	0.08
SendMouseEvents.DoSendMouseEvents()	11.20	2.90	1.00	0 b	0.29	0.06
Monobehaviour.OnMouse_	11.20	0.00	1.00	0 b	0.29	0.00
PreUpdate.SendMouseEvents	11.20	0.00	1.00	0 b	0.29	0.00
MouseController.Update()	9.30	4.60	1.00	0 b	0.22	0.09
Physics.Raycast	10.40	10.40	1.00	0 b	0.22	0.22
UIEvents.IMGUIRenderOverlays	9.50	0.10	1.00	400 B	0.21	0.00
GUI.Repaint	9.50	5.60	1.00	400 B	0.21	0.10
Drawing	12.50	1.80	1.00	0 b	0.20	0.02
GC.Alloc	1.20	1.20	11600.00	461.3 KB	0.16	0.16
EarlyUpdate.PollPlayerConnection	2.10	0.00	1.00	0 b	0.16	0.00
Render.OpaqueGeometry	11.00	0.60	1.00	0 b	0.16	0.01
ProfilerConnection Poll	2.10	2.10	1.00	0 b	0.16	0.16

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E: | Statistics: Max Values | Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
RenderForwardOpaque.Render	9.80	0.60	1.00	0 b	0.14	0.01
Material.SetPassFast	5.00	3.40	18.00	0 b	0.11	0.05
PostLateUpdate.ProfilerEndFrame	6.20	0.00	1.00	0 b	0.11	0.00
EventSystem.Update()	4.40	4.40	1.00	0 b	0.10	0.10
Material.SetPassUncached	4.50	4.50	14.00	0 b	0.10	0.10
Profiler.CollectGlobalStats	6.20	1.80	1.00	0 b	0.10	0.03
HammerMovement	4.10	0.00	1.00	0 b	0.10	0.00
RenderForward.RenderLoopJob	6.70	6.10	1.00	0 b	0.09	0.08
UIEvents.WillRenderCanvases	3.90	0.10	1.00	0 b	0.09	0.00
TransformSystem	3.90	0.00	1.00	0 b	0.09	0.00
MeshInstanceRendererSystem	3.90	0.00	1.00	0 b	0.09	0.00
PostLateUpdate.PlayerUpdateCanvases	4.00	0.00	1.00	0 b	0.09	0.00
UGUI.Rendering.UpdateBatches	3.80	0.20	1.00	0 b	0.09	0.00
Profiler.CollectMemoryAllocationStats	4.00	4.00	1.00	0 b	0.08	0.08
Culling	4.30	1.60	1.00	0 b	0.08	0.03
PathFinderSystem	3.10	0.00	1.00	0 b	0.07	0.00
Layout	3.00	2.80	1.00	0 b	0.07	0.06
Canvas.SendWillRenderCanvases()	3.00	2.30	1.00	0 b	0.07	0.05
TransformSystem:UpdateRotTransTransformNoHierarchyRoots	3.70	0.00	4.00	0 b	0.07	0.00
ExecuteJobFunction.Invoke()	3.70	3.70	4.00	0 b	0.07	0.07
NetworkAgent.Update()	3.50	3.50	400.00	0 b	0.06	0.06
UpdateDepthTexture	2.80	1.30	1.00	0 b	0.06	0.03
Shadows.RenderShadowMap	3.60	2.90	1.00	0 b	0.06	0.04
SceneCulling	3.10	2.10	1.00	0 b	0.06	0.04
PrepareUpdateRendererBoundingVolumes	2.80	2.80	1.00	0 b	0.05	0.05
Shadows.RenderJobDir	1.90	0.30	2.00	0 b	0.04	0.00
Shadows.CollectShadows	1.80	0.90	1.00	0 b	0.04	0.02
Shadows.PrepareShadowmap	2.70	2.20	1.00	0 b	0.04	0.04
Shadows.RenderJob	2.00	0.00	2.00	0 b	0.04	0.00
IdleUtility	1.90	0.00	1.00	6.9 KB	0.04	0.00
RenderForwardOpaque.CollectShadows	2.10	0.30	1.00	0 b	0.04	0.00

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E: | Statistics: Max Values | Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
BatchRenderer.Flush	2.50	2.50	6.00	0 b	0.04	0.04
CullResults.CreateSharedRendererScene	3.30	1.40	1.00	0 b	0.04	0.03
Camera.FindStacks	0.40	0.40	2.00	0 b	0.04	0.04
PreUpdate.AIUpdate	1.80	1.50	1.00	0 b	0.03	0.03
NavMesh.Internal_CallOnNavMeshPreUpdate()	1.70	1.70	1.00	0 b	0.03	0.03
CullPerObjectLights	2.60	2.60	1.00	0 b	0.03	0.03
UIEvents.CanvasManagerRenderOverlays	1.40	0.00	1.00	0 b	0.03	0.00
PhotonHandler.Update()	1.70	1.60	1.00	0 b	0.03	0.03
IdleSystem	1.30	0.00	1.00	0 b	0.03	0.00
UGUI.Rendering.RenderOverlays	1.40	0.20	1.00	0 b	0.03	0.00
Canvas.RenderOverlays	1.30	0.70	2.00	0 b	0.03	0.02
PathBarrier	1.50	0.00	1.00	4.8 KB	0.03	0.00
GameManager.Update()	1.80	1.80	1.00	96 B	0.03	0.03
PostLateUpdate.UpdateCanvasRectTransform	1.60	1.60	1.00	0 b	0.03	0.03
GUIUtility.BeginGUI()	1.80	1.80	3.00	400 B	0.03	0.03
AstarPath.OnGUI()	1.50	1.50	1.00	0 b	0.02	0.02
FPSDisplay.Update()	1.40	1.40	1.00	0 b	0.02	0.02
EarlyUpdate.UpdateCanvasRectTransform	1.10	1.10	1.00	0 b	0.02	0.02
Profiler.CollectDrawStats	1.10	1.10	1.00	0 b	0.02	0.02
RenderForwardAlpha.Render	0.90	0.80	1.00	0 b	0.02	0.02
CullAllVisibleLights	1.40	0.50	1.00	0 b	0.02	0.01
EndFrameBarrier	1.50	0.00	1.00	0 b	0.02	0.00
PostLateUpdate.MemoryFrameMaintenance	1.20	1.20	1.00	0 b	0.02	0.02
PostLateUpdate.EnlightenRuntimeUpdate	0.90	0.10	1.00	0 b	0.02	0.00
Render.TransparentGeometry	1.10	0.60	1.00	0 b	0.02	0.01
Canvas.BuildBatch	1.00	1.00	1.00	0 b	0.02	0.02
DepthPass.Job	1.40	0.70	1.00	0 b	0.02	0.01
Transform2DSystem	1.20	0.00	1.00	0 b	0.02	0.00
MeshLODSystem	2.00	1.90	1.00	0 b	0.02	0.02
PostLateUpdate.UpdateAllSkinnedMeshes	1.10	1.10	1.00	0 b	0.02	0.02
FPSCounter.Update()	0.70	0.70	1.00	224 B	0.01	0.01

File path
\profiler_data.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Figure B.36: DOD DwarfHeim profiler minimum elapsed time statistics for test with animation and models deactivated + several object-oriented components removed such as transform position

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	20.70	0.00	1.00	400 B	9.13	0.01
Camera.Render	20.30	0.00	1.00	0 b	8.97	0.03
Drawing	10.40	0.00	1.00	0 b	4.79	0.01
Render.OpaqueGeometry	10.20	0.00	1.00	0 b	4.66	0.01
PreLateUpdate.DirectorUpdateAnimationEnd	7.60	0.00	1.00	0 b	4.21	0.00
Update.ScriptRunBehaviourUpdate	7.70	0.00	1.00	308 B	3.36	0.00
BehaviourUpdate	7.70	1.00	1.00	308 B	3.36	0.45
PostLateUpdate.UpdateAllSkinnedMeshes	6.10	0.00	1.00	0 b	3.29	0.03
MeshSkinning.Update	6.00	0.30	1.00	0 b	3.25	0.15
MeshSkinning.Prepare	5.60	1.80	1.00	0 b	3.09	0.93
RenderForwardOpaque.Render	6.50	0.00	1.00	0 b	3.01	0.01
PreUpdate.SendMouseEvents	5.70	0.00	1.00	0 b	2.55	0.00
Monobehaviour.OnMouse_	5.70	0.00	1.00	0 b	2.55	0.00
SendMouseEvents.DoSendMouseEvents()	5.70	0.00	1.00	0 b	2.55	0.01
Physics.Raycast	5.60	0.60	1.00	0 b	2.53	0.27
GameAgent.Update()	4.40	4.40	400.00	0 b	2.03	2.03
PreLateUpdate.DirectorUpdateAnimationBegin	2.90	0.00	1.00	0 b	1.43	0.00
Physics.SyncColliderTransform	2.90	2.30	1.00	0 b	1.22	1.01
Shadows.RenderShadowMap	2.70	0.00	1.00	0 b	1.20	0.04
Shadows.PrepareShadowmap	2.80	2.00	1.00	0 b	1.06	0.76
Physics.SyncRigidbodyTransform	2.00	1.20	1.00	0 b	0.96	0.58
PostLateUpdate.UpdateAllRenderers	1.60	0.00	1.00	0 b	0.81	0.00
RenderForwardOpaque.Sort	2.10	2.10	1.00	0 b	0.78	0.78
Shadows.RenderJobDir	1.70	0.70	2.00	0 b	0.73	0.28
Shadows.RenderJob	1.70	0.00	2.00	0 b	0.73	0.00
CullResults.CreateSharedRendererScene	1.70	1.40	1.00	0 b	0.65	0.51
UpdateDepthTexture	1.60	0.10	1.00	0 b	0.63	0.08
DepthPass.Job	1.20	0.50	1.00	0 b	0.48	0.20
Animators.ProcessRootMotionJob	0.80	0.00	1.00	0 b	0.47	0.02
Shadows.PrepareJob	1.20	0.00	1.00	0 b	0.42	0.00
Animator.ProcessRootMotion	0.70	0.70	1.00	0 b	0.40	0.40

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Animators.PrepareFirstPass	0.70	0.70	1.00	0 b	0.32	0.32
RenderForwardOpaque.Prepare	0.50	0.50	1.00	0 b	0.26	0.26
Shadows.ExtractCasters	0.70	0.70	1.00	0 b	0.25	0.25
AbilityAgent.Update()	0.50	0.50	400.00	0 b	0.20	0.20
WorldObjectAgent.Update()	0.50	0.50	400.00	0 b	0.19	0.19
Animators.ApplyOnAnimatorMove	0.40	0.10	1.00	0 b	0.18	0.05
Shadows.CullDirectionalCascades	0.40	0.40	1.00	0 b	0.16	0.16
MeshSkinning.Render	0.30	0.30	1664.00	0 b	0.13	0.13
Animator.ApplyOnAnimatorMove	0.20	0.10	400.00	0 b	0.11	0.05
Director.PrepareFrameJob	0.20	0.00	2.00	0 b	0.11	0.01
GUI.Repaint	0.20	0.10	1.00	400 B	0.10	0.04
UIEvents.IMGUIRenderOverlays	0.20	0.00	1.00	400 B	0.10	0.00
Culling	0.20	0.00	1.00	0 b	0.10	0.01
Profiler.CollectGlobalStats	0.20	0.00	1.00	0 b	0.09	0.00
PostLateUpdate.ProfilerEndFrame	0.20	0.00	1.00	0 b	0.09	0.00
Shadows.Sort	0.10	0.10	2.00	0 b	0.08	0.08
Animators.Update	0.20	0.00	1.00	0 b	0.08	0.00
SceneCulling	0.20	0.00	1.00	0 b	0.07	0.00
DepthPass.Sort	0.10	0.10	1.00	0 b	0.07	0.07
PostLateUpdate.DirectorLateUpdate	0.10	0.10	1.00	0 b	0.07	0.07
ExtractRenderNodeQueue	0.10	0.10	1.00	0 b	0.06	0.06
Shadows.CullingCallbacks	0.10	0.10	1.00	0 b	0.05	0.05
DestroyCullResults	0.10	0.00	1.00	0 b	0.05	0.00
Render.Prepare	0.10	0.10	1.00	0 b	0.05	0.05
PrepareSceneNodesCombineJob	0.10	0.00	1.00	0 b	0.05	0.00
EventSystem.Update()	0.10	0.10	1.00	0 b	0.05	0.05
Animator.ApplyBuiltinRootMotion	0.10	0.10	400.00	0 b	0.04	0.04
Profiler.CollectDrawStats	0.10	0.10	1.00	0 b	0.04	0.04
CommandAgent.Update()	0.10	0.10	400.00	0 b	0.04	0.04
RenderForwardOpaque.CollectShadows	0.00	0.00	1.00	0 b	0.03	0.00
Animators.FireAnimationEventsAndBehaviours	0.00	0.00	1.00	0 b	0.03	0.02

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Director.SampleTime	0.00	0.00	1.00	0 b	0.02	0.02
Running Path Modifiers	0.00	0.00	1.00	56 B	0.02	0.02
NetworkAgent.Update()	0.00	0.00	400.00	0 b	0.02	0.02
EarlyUpdate.DirectorSampleTime	0.00	0.00	1.00	0 b	0.02	0.00
Animators.IKJob	0.00	0.00	2.00	0 b	0.02	0.00
Animator.EvaluateIK	0.00	0.00	1.00	0 b	0.02	0.02
Profiler.CollectMemoryAllocationStats	0.00	0.00	1.00	0 b	0.02	0.02
Batch.DrawDynamic	0.00	0.00	16.00	0 b	0.02	0.02
Shadows.CollectShadows	0.00	0.00	1.00	0 b	0.02	0.01
UGUI.Rendering.RenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
Canvas.RenderOverlays	0.00	0.00	2.00	0 b	0.01	0.00
GUIUtility.BeginGUI()	0.00	0.00	3.00	400 B	0.01	0.01
PrepareSceneNodesJob	0.00	0.00	1.00	0 b	0.01	0.01
EnlightenRuntimeManager.PostUpdate	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.01	0.00
UIEvents.CanvasManagerRenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
CullAllVisibleLights	0.00	0.00	1.00	0 b	0.01	0.00
AnimatorControllerPlayable.PrepareFrame	0.00	0.00	2.00	0 b	0.01	0.01
Animator.ProcessAnimations	0.00	0.00	1.00	0 b	0.01	0.01
JobAlloc.Overflow	0.00	0.00	1.00	0 b	0.01	0.01
TempAlloc.Overflow	0.00	0.00	2.00	0 b	0.01	0.00
Animators.ProcessAnimationsJob	0.00	0.00	1.00	0 b	0.01	0.00
Animator.EvaluateRetargeter	0.00	0.00	1.00	0 b	0.01	0.01
Animators.RetargeterJob	0.00	0.00	1.00	0 b	0.01	0.00
Animator.WriteAnimatedValues	0.00	0.00	1.00	0 b	0.01	0.01
Animators.WriteJob	0.00	0.00	3.00	0 b	0.01	0.00
Render.TransparentGeometry	0.00	0.00	1.00	0 b	0.01	0.00
CullSendEvents	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.ProfilerSynchronizeStats	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.UpdateVideoTextures	0.00	0.00	1.00	0 b	0.00	0.00
PreUpdate.WindUpdate	0.00	0.00	1.00	0 b	0.00	0.00

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.37: OOP DwarfHeim profiler minimum elapsed time statistics for test with animation and models activated

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
PostLateUpdate.FinishFrameRendering	32.89	0.00	1.00	400 B	11.72	0.01
Camera.Render	32.28	0.10	1.00	0 b	11.50	0.04
Update.ScriptRunBehaviourUpdate	21.40	0.00	1.00	166.45 KB	7.98	0.00
BehaviourUpdate	21.40	1.50	1.00	166.45 KB	7.98	0.55
Drawing	18.21	0.01	1.00	0 b	6.48	0.02
Render.OpaqueGeometry	17.72	0.01	1.00	0 b	6.31	0.02
PreLateUpdate.DirectorUpdateAnimationEnd	13.86	0.00	1.00	0 b	4.90	0.00
RenderForwardOpaque.Render	11.82	0.00	1.00	0 b	4.22	0.02
PostLateUpdate.UpdateAllSkinnedMeshes	10.44	0.09	1.00	0 b	3.72	0.04
MeshSkinning.Update	10.30	0.61	1.00	0 b	3.67	0.23
PreUpdate.SendMouseEvents	9.74	0.00	1.00	2 b	3.49	0.00
Monobehaviour.OnMouse_	9.74	0.00	1.00	2 b	3.49	0.00
SendMouseEvents.DoSendMouseEvents()	9.74	0.00	1.00	2 b	3.49	0.02
Physics.Raycast	9.67	1.07	1.00	0 b	3.47	0.39
MeshSkinning.Prepare	9.64	3.02	1.00	0 b	3.44	1.09
Animators.Update	8.79	0.30	1.00	0 b	3.12	0.12
GameAgent.Update()	7.32	6.97	400.00	12.79 KB	2.63	2.50
MeshSkinning.CalcMatrices	6.29	6.29	1915.29	0 b	2.25	2.25
Shadows.RenderShadowMap	5.20	0.12	1.00	0 b	1.86	0.06
Animators.IKJob	5.12	0.12	2.00	0 b	1.82	0.05
RVOSimulator.Update()	4.65	4.65	2.00	0 b	1.82	1.82
PreLateUpdate.DirectorUpdateAnimationBegin	5.03	0.00	1.00	0 b	1.79	0.00
Physics.SyncColliderTransform	4.99	4.15	1.00	0 b	1.79	1.49
Image.Start() [Coroutine: MoveNext]	4.23	4.19	89.81	56.19 KB	1.63	1.61
Shadows.PrepareShadowmap	4.49	3.27	1.00	0 b	1.60	1.17
Animator.EvaluateIK	4.45	4.45	49.05	0 b	1.58	1.58
Culling	4.33	0.00	1.00	0 b	1.56	0.01
SceneCulling	4.26	0.08	1.00	0 b	1.53	0.03
CullSendEvents	3.88	0.01	1.00	0 b	1.40	0.00
CullResults.CreateSharedRendererScene	3.63	2.84	1.00	0 b	1.30	1.02
Physics.SyncRigidbodyTransform	3.52	2.23	1.00	0 b	1.27	0.82

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.RenderJob	3.49	0.00	2.00	0 b	1.26	0.00
Animators.WriteJob	3.54	0.30	3.00	0 b	1.25	0.12
Shadows.RenderJobDir	3.49	1.63	2.00	0 b	1.25	0.60
UpdateDepthTexture	3.41	0.48	1.00	0 b	1.23	0.18
AbilityAgent.Update()	3.08	3.03	400.00	92.89 KB	1.19	1.17
PostLateUpdate.UpdateAllRenderers	2.99	0.00	1.00	0 b	1.07	0.00
Animator.WriteAnimatedValues	2.77	2.77	51.22	0 b	0.98	0.98
RenderForward.RenderLoopJob	2.70	1.86	1.00	0 b	0.97	0.67
DepthPass.Job	2.64	1.34	1.00	0 b	0.96	0.49
Director.ProcessFrame	2.52	0.00	1.23	0 b	0.89	0.00
AstarPath.Update()	2.20	1.19	1.00	23.43 KB	0.86	0.46
RenderForwardOpaque.Sort	2.10	2.10	1.00	0 b	0.78	0.78
Running Path Modifiers	1.93	1.92	67.69	4.34 KB	0.76	0.76
Shadows.PrepareJob	1.74	0.00	1.00	0 b	0.63	0.00
WaitForJobGroupID	1.59	0.49	1.29	0 b	0.57	0.19
PrepareSceneNodesCombineJob	1.55	0.00	1.00	0 b	0.57	0.00
Animators.ProcessRootMotionJob	1.55	0.03	1.00	0 b	0.56	0.03
Animators.RetargeterJob	1.53	0.02	1.00	0 b	0.55	0.03
Update.ScriptRunDelayedDynamicFrameRate	1.35	0.00	1.00	17.16 KB	0.51	0.00
Animator.EvaluateRetargeter	1.38	1.38	50.21	0 b	0.50	0.50
Animators.ProcessAnimationsJob	1.33	0.01	1.00	0 b	0.48	0.02
Animator.ProcessRootMotion	1.28	1.28	50.14	0 b	0.46	0.46
RenderForwardOpaque.Prepare	1.24	1.19	1.00	0 b	0.45	0.44
Animators.PrepareFirstPass	1.18	1.18	1.00	0 b	0.43	0.43
Animator.ProcessAnimations	1.13	1.13	50.51	0 b	0.41	0.41
Calling Path Callbacks	0.99	0.12	1.00	16.09 KB	0.39	0.05
Shadows.ExtractCasters	0.94	0.94	1.00	0 b	0.34	0.34
WarriorCreator.Update()	0.87	0.83	1.00	37.11 KB	0.33	0.32
Director.PrepareFrameJob	0.89	0.41	49.87	0 b	0.33	0.16
Animators.PrepareSecondPass	0.79	0.79	1.00	0 b	0.30	0.30
WorldObjectAgent.Update()	0.78	0.78	400.00	0 b	0.29	0.29

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
MeshSkinning.Render	0.71	0.71	1799.35	0 b	0.27	0.27
Shadows.CullDirectionalCascades	0.73	0.73	1.00	0 b	0.27	0.27
UpdateRendererBoundingVolumes	0.75	0.35	8.58	0 b	0.27	0.13
Animators.ApplyOnAnimatorMove	0.71	0.20	1.00	0 b	0.26	0.08
TransformChangedDispatch	0.65	0.31	1.53	0 b	0.24	0.12
SkinnedMeshUpdateAllNeeded	0.52	0.52	2.35	0 b	0.20	0.20
BatchRenderer.Flush	0.53	0.39	1464.43	0 b	0.20	0.15
Animator.ApplyOnAnimatorMove	0.46	0.30	400.00	0 b	0.18	0.12
AnimatorControllerPlayable.PrepareFrame	0.43	0.43	49.87	0 b	0.16	0.16
LateBehaviourUpdate	0.40	0.00	1.00	0 b	0.16	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.40	0.00	1.00	0 b	0.16	0.00
Profiler.CollectGlobalStats	0.41	0.00	1.00	0 b	0.16	0.01
PostLateUpdate.ProfilerEndFrame	0.41	0.00	1.00	0 b	0.16	0.00
NetworkManager.LateUpdate()	0.39	0.39	1.00	0 b	0.16	0.16
GUI.Repaint	0.38	0.15	1.00	400 B	0.15	0.06
UIEvents.IMGUIRenderOverlays	0.38	0.00	1.00	400 B	0.15	0.00
SkinnedMeshFinalizeUpdate	0.38	0.14	1.00	0 b	0.14	0.05
DestroyCullResults	0.33	0.00	1.00	0 b	0.13	0.00
FinalizeUpdateRendererBoundingVolumes	0.31	0.00	1.00	0 b	0.12	0.00
Animators.DirtySceneObjects	0.28	0.28	1.00	0 b	0.11	0.11
Animators.SortWriteJob	0.28	0.28	2.00	0 b	0.11	0.11
ExtractRenderNodeQueue	0.29	0.29	2.67	0 b	0.11	0.11
CoroutinesDelayedCalls	0.27	0.01	1.16	3.43 KB	0.10	0.00
Profiler.CollectDrawStats	0.24	0.24	1.00	0 b	0.10	0.10
Render.Prepare	0.24	0.24	1.00	0 b	0.10	0.10
Shadows.CullingCallbacks	0.25	0.25	1.00	0 b	0.10	0.10
PostLateUpdate.DirectorLateUpdate	0.23	0.23	1.00	0 b	0.09	0.09
Shadows.Sort	0.21	0.21	2.00	0 b	0.09	0.09
DepthPass.Sort	0.20	0.20	1.00	0 b	0.08	0.08
RenderLoop.CleanupNodeQueue	0.20	0.20	1.67	0 b	0.08	0.08
EventSystem.Update()	0.14	0.14	1.00	0 b	0.07	0.07

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
CommandAgent.Update()	0.16	0.16	400.00	0 b	0.07	0.07
FPSCounter.Update()	0.12	0.12	1.00	212 B	0.07	0.07
PrepareSceneNodes	0.15	0.15	1.00	0 b	0.06	0.06
Director.PrepareFrame	0.17	0.03	1.23	0 b	0.06	0.01
Animators.FireAnimationEventsAndBehaviours	0.11	0.08	1.00	0 b	0.05	0.04
RenderForwardOpaque.CollectShadows	0.11	0.00	1.00	0 b	0.05	0.01
Animator.ApplyBuiltinRootMotion	0.12	0.12	400.00	0 b	0.05	0.05
PrepareSceneNodesJob	0.06	0.06	1.14	0 b	0.04	0.04
Batch.DrawDynamic	0.08	0.07	18.04	0 b	0.04	0.04
Profiler.CollectMemoryAllocationStats	0.07	0.07	1.00	0 b	0.04	0.04
EarlyUpdate.DirectorSampleTime	0.07	0.00	1.00	0 b	0.04	0.00
CullAllVisibleLights	0.07	0.05	1.00	0 b	0.03	0.02
TempAlloc.Overflow	0.05	0.00	2.00	0 b	0.03	0.00
Director.SampleTime	0.06	0.06	1.00	0 b	0.03	0.03
NetworkAgent.Update()	0.05	0.05	400.00	0 b	0.03	0.03
Shadows.CollectShadows	0.05	0.01	1.00	0 b	0.03	0.02
JobAlloc.Overflow	0.04	0.04	1.67	0 b	0.03	0.03
Render.Mesh	0.03	0.03	477.21	0 b	0.02	0.02
GUIUtility.BeginGUI()	0.01	0.01	3.00	400 B	0.02	0.02
Render.TransparentGeometry	0.01	0.00	1.00	0 b	0.02	0.01
UGUI.Rendering.RenderOverlays	0.01	0.00	1.00	0 b	0.02	0.00
Canvas.RenderOverlays	0.01	0.00	2.00	0 b	0.02	0.01
UIEvents.CanvasManagerRenderOverlays	0.01	0.00	1.00	0 b	0.02	0.00
Camera.RenderSkybox	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.PlayerUpdateCanvases	0.03	0.00	1.00	0 b	0.01	0.00
CombineJobResults	0.02	0.02	1.28	0 b	0.01	0.01
Camera.ImageEffects	0.00	0.00	1.00	0 b	0.01	0.00
Batch.ProcessGeometryJob	0.00	0.00	1.17	0 b	0.01	0.01
Material.SetPassUncached	0.01	0.01	5.67	0 b	0.01	0.01
Material.SetPassFast	0.01	0.00	4.00	0 b	0.01	0.00
Graphics.Blit	0.00	0.00	1.00	0 b	0.01	0.01

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Update.ScriptRunBehaviourUpdate	52.90	0.00	1.00	512 KB	30.61	0.00
BehaviourUpdate	52.90	2.30	1.00	512 KB	30.61	0.85
FPSCounter.Update()	35.40	35.40	1.00	212 B	20.49	20.49
PreUpdate.SendMouseEvents	42.10	0.00	1.00	24 B	19.33	0.00
Monobehaviour.OnMouse_	42.10	0.00	1.00	24 B	19.33	0.00
SendMouseEvents.DoSendMouseEvents()	42.10	0.10	1.00	24 B	19.33	0.04
Physics.Raycast	42.00	2.20	1.00	0 b	19.31	0.86
Physics.SyncRigidbodyTransform	37.40	36.30	1.00	0 b	17.20	16.69
PostLateUpdate.FinishFrameRendering	43.90	0.00	1.00	400 B	16.92	0.03
Camera.Render	43.30	1.40	1.00	0 b	16.75	0.56
WarriorCreator.Update()	30.50	30.10	1.00	506 KB	13.44	13.26
CoroutinesDelayedCalls	26.60	0.30	3.00	161.1 KB	11.51	0.17
Update.ScriptRunDelayedDynamicFrameRate	26.60	0.00	1.00	161.1 KB	11.51	0.00
Image.Start() [Coroutine: MoveNext]	26.20	26.10	217.00	161.1 KB	11.35	11.31
AbilityAgent.Update()	23.00	22.90	400.00	381.3 KB	10.83	10.78
Drawing	27.60	0.10	1.00	0 b	10.03	0.05
Render.OpaqueGeometry	27.10	2.60	1.00	0 b	9.84	0.93
SkinnedMeshFinalizeUpdate	24.40	24.40	1.00	0 b	9.67	9.66
FinalizeUpdateRendererBoundingVolumes	24.40	0.00	1.00	0 b	9.67	0.00
RVOSimulator.Update()	21.80	21.80	2.00	0 b	8.32	8.32
RenderForwardOpaque.Render	20.10	0.10	1.00	0 b	7.65	0.06
GameAgent.Update()	16.30	11.40	400.00	135.3 KB	7.50	3.78
PreLateUpdate.DirectorUpdateAnimationEnd	18.80	0.00	1.00	0 b	6.44	0.00
Director.ProcessFrame	18.80	0.00	3.00	0 b	6.44	0.00
Animators.Update	18.80	1.30	1.00	0 b	6.44	0.48
Physics.SyncColliderTransform	16.50	15.70	1.00	0 b	5.65	5.35
UpdateDepthTexture	11.80	2.60	1.00	0 b	5.61	0.94
PostLateUpdate.UpdateAllSkinnedMeshes	16.00	0.30	1.00	0 b	5.44	0.13
MeshSkinning.Update	15.80	1.00	1.00	0 b	5.37	0.37
DepthPass.Job	11.10	7.70	1.00	0 b	5.29	3.34
MeshSkinning.Prepare	14.80	5.00	1.00	0 b	5.08	1.60

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Culling	14.40	0.00	1.00	0 b	4.80	0.03
SceneCulling	14.30	12.40	1.00	0 b	4.76	4.28
CullSendEvents	12.80	1.70	1.00	0 b	4.73	0.52
WaitForJobGroupID	12.80	12.70	3.00	0 b	4.71	4.70
CullAllVisibleLights	14.10	14.10	1.00	0 b	4.57	4.55
BatchRenderer.Flush	8.80	8.60	2260.00	0 b	4.19	4.09
CullResults.CreateSharedRendererScene	11.90	9.40	1.00	0 b	4.12	3.38
RenderForward.RenderLoopJob	9.20	6.50	1.00	0 b	4.03	2.87
PrepareSceneNodes	10.90	10.90	1.00	0 b	3.93	3.93
Shadows.RenderShadowMap	10.20	0.30	1.00	0 b	3.91	0.12
AstarPath.Update()	9.90	5.50	1.00	114.8 KB	3.74	1.96
MeshSkinning.CalcMatrices	10.00	10.00	2000.00	0 b	3.47	3.47
PrepareSceneNodesCombineJob	9.50	0.00	1.00	0 b	3.36	0.00
PreLateUpdate.DirectorUpdateAnimationBegin	8.00	0.00	1.00	0 b	2.88	0.00
Shadows.RenderJobDir	8.20	4.20	2.00	0 b	2.81	1.52
Shadows.RenderJob	8.20	0.00	2.00	0 b	2.81	0.00
Shadows.PrepareShadowmap	8.00	5.30	1.00	0 b	2.53	1.89
FindActiveLights	6.90	6.90	1.00	0 b	2.49	2.49
MeshSkinning.Render	7.90	7.90	1984.00	0 b	2.44	2.44
Animators.IKJob	7.10	4.60	2.00	0 b	2.28	1.65
Animator.EvaluateIK	6.30	6.30	69.00	0 b	2.11	2.11
Calling Path Callbacks	4.70	0.60	1.00	83.7 KB	1.88	0.29
PostLateUpdate.UpdateAllRenderers	5.50	0.00	1.00	0 b	1.78	0.00
PrepareSceneNodesJob	5.40	5.40	2.00	0 b	1.76	1.76
UpdateRendererBoundingVolumes	5.40	3.90	14.00	0 b	1.74	1.31
Animators.WriteJob	5.60	4.50	3.00	0 b	1.66	1.45
Running Path Modifiers	4.00	4.00	153.00	9.6 KB	1.61	1.61
Animator.WriteAnimatedValues	4.50	4.50	69.00	0 b	1.38	1.38
RenderForwardOpaque.Prepare	3.40	3.40	1.00	0 b	1.30	1.30
SkinnedMeshUpdateAllNeeded	2.90	2.90	5.00	0 b	1.03	1.03
Animators.ProcessRootMotionJob	2.50	1.80	1.00	0 b	0.90	0.59

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.PrepareJob	2.60	0.00	1.00	0 b	0.86	0.01
Render.Prepare	2.30	2.30	1.00	0 b	0.85	0.85
Animator.ProcessRootMotion	2.30	2.30	70.00	0 b	0.83	0.83
RenderForwardOpaque.Sort	2.10	2.10	1.00	0 b	0.78	0.78
TransformChangedDispatch	1.80	1.70	5.00	0 b	0.76	0.73
Animators.ProcessAnimationsJob	2.10	1.30	1.00	0 b	0.75	0.49
RenderLoop.CleanupNodeQueue	2.40	2.40	3.00	0 b	0.74	0.74
Animators.RetargeterJob	2.10	1.70	1.00	0 b	0.74	0.61
NetworkManager.LateUpdate()	1.60	1.60	1.00	0 b	0.73	0.73
LateBehaviourUpdate	1.60	0.00	1.00	0 b	0.73	0.01
PreLateUpdate.ScriptRunBehaviourLateUpdate	1.60	0.00	1.00	0 b	0.73	0.00
Animators.PrepareFirstPass	1.90	1.90	1.00	0 b	0.70	0.70
Animators.PrepareSecondPass	1.80	1.80	1.00	0 b	0.70	0.70
Animator.EvaluateRetargeter	2.00	2.00	67.00	0 b	0.67	0.67
Director.PrepareFrame	1.90	1.70	3.00	0 b	0.64	0.51
Animators.ApplyOnAnimatorMove	1.80	0.50	1.00	0 b	0.64	0.20
Animator.ProcessAnimations	1.80	1.80	71.00	0 b	0.63	0.63
Profiler.CollectGlobalStats	1.80	0.20	1.00	0 b	0.62	0.08
PostLateUpdate.ProfilerEndFrame	1.90	0.00	1.00	0 b	0.62	0.01
Batch.DrawDynamic	1.50	1.50	19.00	0 b	0.52	0.51
Shadows.ExtractCasters	1.50	1.50	1.00	0 b	0.51	0.51
WorldObjectAgent.Update()	1.50	1.50	400.00	0 b	0.50	0.50
Director.PrepareFrameJob	1.40	1.00	73.00	0 b	0.50	0.39
Profiler.CollectDrawStats	1.40	1.40	1.00	0 b	0.48	0.48
Animator.ApplyOnAnimatorMove	1.20	0.70	400.00	0 b	0.43	0.27
UIEvents.IMGUIRenderOverlays	1.00	0.00	1.00	400 B	0.42	0.00
GUI.Repaint	1.00	0.50	1.00	400 B	0.41	0.18
AnimatorControllerPlayable.PrepareFrame	1.30	1.30	73.00	0 b	0.40	0.40
DestroyCullResults	0.90	0.10	1.00	0 b	0.36	0.04
Shadows.CullDirectionalCascades	1.00	1.00	1.00	0 b	0.34	0.34
GC.Alloc	0.70	0.70	12418.00	506 KB	0.32	0.32

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Animators.DirtySceneObjects	0.80	0.80	1.00	0 b	0.32	0.32
PostLateUpdate.DirectorLateUpdate	0.70	0.70	1.00	0 b	0.25	0.25
PhotonHandler.Update()	0.50	0.00	1.00	0 b	0.24	0.03
SendOutgoingCommands	0.50	0.50	3.00	0 b	0.22	0.22
ExtractRenderNodeQueue	0.60	0.60	5.00	0 b	0.22	0.22
EarlyUpdate.DirectorSampleTime	0.40	0.00	1.00	0 b	0.22	0.02
DepthPass.Sort	0.50	0.50	1.00	0 b	0.21	0.21
Shadows.CullingCallbacks	0.60	0.60	1.00	0 b	0.21	0.21
Render.Mesh	0.50	0.50	713.00	0 b	0.21	0.21
Animators.SortWriteJob	0.60	0.60	2.00	0 b	0.21	0.21
Director.SampleTime	0.40	0.40	1.00	0 b	0.20	0.20
Shadows.CullDirectionalShadowCasters	0.50	0.50	1.00	0 b	0.20	0.20
CombineJobResults	0.50	0.50	2.00	0 b	0.18	0.18
Animators.FireAnimationEventsAndBehaviours	0.50	0.30	1.00	0 b	0.18	0.13
Material.SetPassFast	0.40	0.00	18.00	0 b	0.18	0.02
PostLateUpdate.PlayerUpdateCanvases	0.40	0.00	1.00	0 b	0.17	0.00
EventSystem.Update()	0.40	0.40	1.00	0 b	0.17	0.17
UIEvents.WillRenderCanvases	0.40	0.00	1.00	0 b	0.17	0.00
EarlyUpdate.PollPlayerConnection	0.40	0.00	1.00	0 b	0.16	0.00
ProfilerConnection Poll	0.40	0.40	1.00	0 b	0.16	0.16
UGUI.Rendering.UpdateBatches	0.40	0.00	1.00	0 b	0.16	0.01
Animator.ApplyBuiltinRootMotion	0.40	0.40	400.00	0 b	0.16	0.16
RenderForwardOpaque.CollectShadows	0.30	0.20	1.00	0 b	0.15	0.09
Material.SetPassUncached	0.30	0.30	14.00	0 b	0.15	0.15
Shadows.Sort	0.40	0.40	2.00	0 b	0.13	0.13
Shadows.CollectShadows	0.30	0.20	1.00	0 b	0.13	0.08
Canvas.SendWillRenderCanvases()	0.30	0.00	1.00	0 b	0.12	0.00
CommandAgent.Update()	0.30	0.30	400.00	0 b	0.12	0.12
Profiler.CollectMemoryAllocationStats	0.30	0.30	1.00	0 b	0.12	0.12
JobAlloc.Overflow	0.30	0.30	2.00	0 b	0.11	0.11
Layout	0.30	0.30	1.00	0 b	0.11	0.11

File path

\Build\profiler_data_type1.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Update.ScriptRunBehaviourUpdate	29.40	0.00	1.00	96 B	3.08	0.00
BehaviourUpdate	29.40	2.00	1.00	96 B	3.07	0.45
Monobehaviour.OnMouse_	14.80	0.00	1.00	0 b	2.56	0.00
SendMouseEvents.DoSendMouseEvents()	14.80	0.00	1.00	0 b	2.56	0.01
PreUpdate.SendMouseEvents	14.80	0.00	1.00	0 b	2.56	0.00
Physics.Raycast	14.70	1.90	1.00	0 b	2.54	0.33
GameAgent.Update()	9.70	9.00	400.00	0 b	2.02	2.02
Physics.SyncColliderTransform	8.20	7.10	1.00	0 b	1.38	1.18
Physics.SyncRigidbodyTransform	4.30	2.80	1.00	0 b	0.80	0.52
PostLateUpdate.FinishFrameRendering	3.00	0.00	1.00	400 B	0.75	0.00
Camera.Render	2.50	0.00	1.00	0 b	0.62	0.01
Drawing	1.60	0.00	1.00	0 b	0.38	0.00
Render.OpaqueGeometry	1.40	0.00	1.00	0 b	0.35	0.00
RenderForwardOpaque.Render	1.00	0.00	1.00	0 b	0.25	0.00
WorldObjectAgent.Update()	1.10	1.10	400.00	0 b	0.20	0.20
PostLateUpdate.UpdateAllRenderers	1.20	0.00	1.00	0 b	0.20	0.00
AbilityAgent.Update()	1.10	1.10	400.00	0 b	0.18	0.18
Shadows.RenderShadowMap	0.50	0.00	1.00	0 b	0.12	0.01
GUI.Repaint	0.30	0.10	1.00	400 B	0.09	0.04
UIEvents.IMGUIRenderOverlays	0.30	0.00	1.00	400 B	0.09	0.00
UpdateDepthTexture	0.30	0.10	1.00	0 b	0.08	0.02
CullResults.CreateSharedRendererScene	0.20	0.00	1.00	0 b	0.06	0.00
Shadows.RenderJob	0.20	0.00	2.00	0 b	0.05	0.00
Shadows.RenderJobDir	0.20	0.00	2.00	0 b	0.05	0.02
EventSystem.Update()	0.20	0.20	1.00	0 b	0.04	0.04
Shadows.PrepareShadowmap	0.20	0.00	1.00	0 b	0.04	0.01
Culling	0.20	0.00	1.00	0 b	0.04	0.01
RenderForwardOpaque.Prepare	0.10	0.10	1.00	0 b	0.04	0.04
DepthPass.Job	0.10	0.00	1.00	0 b	0.04	0.01
PostLateUpdate.ProfilerEndFrame	0.10	0.00	1.00	0 b	0.03	0.00
Profiler.CollectGlobalStats	0.10	0.00	1.00	0 b	0.03	0.00

File path

\Build\profiler_data_type2.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Min Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
CommandAgent.Update()	0.20	0.20	400.00	0 b	0.03	0.03
RenderForwardOpaque.CollectShadows	0.10	0.00	1.00	0 b	0.02	0.00
Running Path Modifiers	0.20	0.20	1.00	340 B	0.02	0.02
NetworkAgent.Update()	0.10	0.10	400.00	0 b	0.02	0.02
SceneCulling	0.10	0.00	1.00	0 b	0.02	0.00
Profiler.CollectMemoryAllocationStats	0.00	0.00	1.00	0 b	0.02	0.02
Canvas.RenderOverlays	0.00	0.00	2.00	0 b	0.01	0.00
GUIUtility.BeginGUI()	0.00	0.00	3.00	400 B	0.01	0.01
Shadows.CollectShadows	0.00	0.00	1.00	0 b	0.01	0.01
UIEvents.CanvasManagerRenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.00	0.00	1.00	0 b	0.01	0.00
UGUI.Rendering.RenderOverlays	0.00	0.00	1.00	0 b	0.01	0.00
ExtractRenderNodeQueue	0.00	0.00	1.00	0 b	0.01	0.01
JobAlloc.Overflow	0.00	0.00	2.00	0 b	0.01	0.01
TempAlloc.Overflow	0.00	0.00	2.00	0 b	0.01	0.00
PreLateUpdate.DirectorUpdateAnimationEnd	0.00	0.00	1.00	0 b	0.00	0.00
PlayerCleanupCachedData	0.00	0.00	1.00	0 b	0.00	0.00
CleanUp.TextRenderingGarbageCollect	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.UpdatePreloading	0.00	0.00	1.00	0 b	0.00	0.00
CanvasRenderer.SyncWorldRect	0.00	0.00	1.00	0 b	0.00	0.00
Canvas.BuildBatch	0.00	0.00	1.00	0 b	0.00	0.00
Font.CacheFontForText	0.00	0.00	2.00	0 b	0.00	0.00
RenderForwardAlpha.Sort	0.00	0.00	1.00	0 b	0.00	0.00
Loading.UpdatePreloading	0.00	0.00	1.00	0 b	0.00	0.00
UpdatePreloading	0.00	0.00	1.00	0 b	0.00	0.00
Application.Integrate Assets in Background	0.00	0.00	1.00	0 b	0.00	0.00
Preload Single Step	0.00	0.00	1.00	0 b	0.00	0.00
SendOutgoingCommands	0.00	0.00	1.00	0 b	0.00	0.00
PreUpdate.CheckTexFieldInput	0.00	0.00	1.00	0 b	0.00	0.00
FixedUpdate.NewInputEndFixedUpdate	0.00	0.00	1.00	0 b	0.00	0.00
Cleanup Unused Cached Data	0.00	0.00	1.00	0 b	0.00	0.00

File path

\Build\profiler_data_type2.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Figure B.40: OOP DwarfHeim profiler minimum elapsed time statistics for test with animation and models deactivated

Profiler Data E:
Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Update.ScriptRunBehaviourUpdate	46.40	0.00	1.00	44.54 KB	4.83	0.00
BehaviourUpdate	46.38	5.78	1.00	44.54 KB	4.82	0.55
SendMouseEvents.DoSendMouseEvents()	35.39	0.16	1.00	0 b	3.38	0.01
Monobehaviour.OnMouse_	35.40	0.00	1.00	0 b	3.38	0.00
PreUpdate.SendMouseEvents	35.40	0.00	1.00	0 b	3.38	0.00
Physics.Raycast	35.19	5.16	1.00	0 b	3.36	0.49
GameAgent.Update()	25.12	24.88	400.00	2.38 KB	2.42	2.38
Physics.SyncColliderTransform	19.62	17.08	1.00	0 b	1.88	1.64
Image.Start() [Coroutine: MoveNext]	9.98	9.88	76.71	47.45 KB	1.45	1.44
PostLateUpdate.FinishFrameRendering	10.67	0.07	1.00	400 B	1.02	0.00
Physics.SyncRigidbodyTransform	10.30	6.86	1.00	0 b	0.98	0.65
Camera.Render	8.79	0.24	1.00	0 b	0.84	0.02
Running Path Modifiers	4.92	4.91	63.16	4.04 KB	0.71	0.71
Drawing	5.57	0.05	1.00	0 b	0.53	0.00
RVOSimulator.Update()	3.41	3.41	2.00	0 b	0.52	0.52
AbilityAgent.Update()	4.59	4.55	400.00	25.46 KB	0.50	0.50
Render.OpaqueGeometry	5.14	0.00	1.00	0 b	0.49	0.00
RenderForwardOpaque.Render	3.70	0.09	1.00	0 b	0.36	0.01
PostLateUpdate.UpdateAllRenderers	3.14	0.00	1.00	0 b	0.31	0.00
WorldObjectAgent.Update()	2.82	2.82	400.00	0 b	0.27	0.27
AstarPath.Update()	1.62	0.89	1.00	6.38 KB	0.23	0.13
Shadows.RenderShadowMap	1.84	0.15	1.00	0 b	0.18	0.01
TransformChangedDispatch	1.86	0.91	1.52	0 b	0.18	0.09
Update.ScriptRunDelayedDynamicFrameRate	1.20	0.00	1.00	5.31 KB	0.17	0.00
UIEvents.IMGUIRenderOverlays	1.30	0.00	1.00	400 B	0.12	0.00
GUI.Repaint	1.30	0.60	1.00	400 B	0.12	0.06
WaitForJobGroupID	1.11	0.21	1.00	0 b	0.11	0.02
WarriorCreator.Update()	0.71	0.68	1.00	10.12 KB	0.10	0.10
UpdateDepthTexture	1.09	0.34	1.00	0 b	0.10	0.03
Calling Path Callbacks	0.73	0.09	1.00	4.39 KB	0.10	0.01
Shadows.RenderJobDir	0.85	0.30	2.00	0 b	0.08	0.03

File path
\Build\profiler_data_type2.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.RenderJob	0.85	0.00	2.00	0 b	0.08	0.00
CullResults.CreateSharedRendererScene	0.76	0.09	1.00	0 b	0.07	0.01
CommandAgent.Update()	0.73	0.73	400.00	0 b	0.07	0.07
Shadows.PrepareShadowmap	0.76	0.10	1.00	0 b	0.07	0.01
UpdateRendererBoundingVolumes	0.75	0.38	8.36	0 b	0.07	0.04
DepthPass.Job	0.60	0.21	1.00	0 b	0.06	0.02
RenderForward.RenderLoopJob	0.59	0.36	1.00	0 b	0.06	0.03
Culling	0.63	0.12	1.00	0 b	0.06	0.01
RenderForwardOpaque.Prepare	0.51	0.51	1.00	0 b	0.05	0.05
Profiler.CollectGlobalStats	0.51	0.06	1.00	0 b	0.05	0.00
PostLateUpdate.ProfilerEndFrame	0.51	0.00	1.00	0 b	0.05	0.00
SceneCulling	0.38	0.00	1.00	0 b	0.04	0.00
PreLateUpdate.ScriptRunBehaviourLateUpdate	0.35	0.00	1.00	0 b	0.04	0.00
NetworkManager.LateUpdate()	0.35	0.35	1.00	0 b	0.04	0.04
ExtractRenderNodeQueue	0.44	0.44	1.00	0 b	0.04	0.04
EventSystem.Update()	0.46	0.46	1.00	0 b	0.04	0.04
LateBehaviourUpdate	0.35	0.00	1.00	0 b	0.04	0.00
CoroutinesDelayedCalls	0.27	0.01	1.00	1.18 KB	0.04	0.00
PostLateUpdate.PlayerUpdateCanvases	0.36	0.00	1.00	0 b	0.03	0.00
UGUI.Rendering.UpdateBatches	0.35	0.00	1.00	0 b	0.03	0.00
Profiler.CollectMemoryAllocationStats	0.31	0.31	1.00	0 b	0.03	0.03
Canvas.SendWillRenderCanvases()	0.27	0.00	1.00	0 b	0.03	0.00
Layout	0.26	0.24	1.00	0 b	0.03	0.02
UIEvents.WillRenderCanvases	0.35	0.00	1.00	0 b	0.03	0.00
BatchRenderer.Flush	0.28	0.09	8.38	0 b	0.03	0.01
NetworkAgent.Update()	0.34	0.34	400.00	0 b	0.03	0.03
RenderForwardOpaque.CollectShadows	0.32	0.02	1.00	0 b	0.03	0.00
JobAlloc.Overflow	0.30	0.30	2.00	0 b	0.03	0.03
TempAlloc.Overflow	0.30	0.00	2.00	0 b	0.03	0.00
UGUI.Rendering.RenderOverlays	0.18	0.00	1.00	0 b	0.02	0.00
UIEvents.CanvasManagerRenderOverlays	0.18	0.00	1.00	0 b	0.02	0.00

File path
\Build\profiler_data_type2.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:
Statistics Average Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.CollectShadows	0.23	0.12	1.00	0 b	0.02	0.01
Material.SetPassFast	0.07	0.00	4.00	0 b	0.01	0.00
CullSendEvents	0.14	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.12	0.00	1.00	0 b	0.01	0.00
EnlightenRuntimeManager.PostUpdate	0.09	0.03	1.00	0 b	0.01	0.00
Batch.DrawDynamic	0.11	0.11	3.26	0 b	0.01	0.01
GUIUtility.BeginGUI()	0.12	0.12	3.00	400 B	0.01	0.01
Canvas.RenderOverlays	0.14	0.05	2.00	0 b	0.01	0.00
DestroyCullResults	0.08	0.00	1.00	0 b	0.01	0.00
Render.Prepare	0.10	0.10	1.00	0 b	0.01	0.01
CullAllVisibleLights	0.08	0.00	1.00	0 b	0.01	0.00
Material.SetPassUncached	0.12	0.12	5.67	0 b	0.01	0.01
Shadows.Sort	0.08	0.08	2.00	0 b	0.01	0.01
Shadows.CullingCallbacks	0.07	0.07	1.00	0 b	0.01	0.01
Canvas.BuildBatch	0.09	0.09	1.00	0 b	0.01	0.01
EarlyUpdate.CloudWebServicesUpdate	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.DispatchEventQueueEvents	0.00	0.00	1.00	0 b	0.00	0.00
GlobalEventQueue	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.GpuTimestamp	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.PhysicsSkinnedClothBeginUpdate	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.PerformanceAnalyticsUpdate	0.00	0.00	1.00	0 b	0.00	0.00
FixedUpdate.Physics2DFixedUpdate	0.00	0.00	1.01	0 b	0.00	0.00
PostLateUpdate.MemoryFrameMaintenance	0.00	0.00	1.00	0 b	0.00	0.00
EarlyUpdate.NewInputBeginFrame	0.00	0.00	1.00	0 b	0.00	0.00
Physics2D.Simulate	0.00	0.00	1.01	0 b	0.00	0.00
PostLateUpdate.ClearImmediateRenderers	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.DirectorLateUpdate	0.00	0.00	1.00	0 b	0.00	0.00
Director.ProcessFrame	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.DirectorRenderImage	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.GUIClearEvents	0.00	0.00	1.00	0 b	0.00	0.00
PostLateUpdate.InputEndFrame	0.00	0.00	1.00	0 b	0.00	0.00

File path
\Build\profiler_data_type2.json Open Edit
Export
Profiler Data | Current Frame Data | Selected Function Data

Figure B.41: OOP DwarfHeim profiler average elapsed time statistics for test with animations and models deactivated

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Update.ScriptRunBehaviourUpdate	76.20	0.00	1.00	506.4 KB	15.37	0.00
BehaviourUpdate	76.20	8.10	1.00	506.4 KB	15.37	0.82
CoroutinesDelayedCalls	42.90	0.80	2.00	124.7 KB	11.17	0.09
Update.ScriptRunDelayedDynamicFrameRate	42.90	0.00	1.00	124.7 KB	11.17	0.00
Image.Start() [Coroutine: MoveNext]	42.50	42.40	168.00	124.7 KB	11.08	11.04
WarriorCreator.Update()	58.60	57.60	1.00	506 KB	10.37	10.20
RVOSimulator.Update()	42.60	42.60	2.00	0 b	7.51	7.51
SendMouseEvents.DoSendMouseEvents()	46.90	1.00	1.00	0 b	5.52	0.09
Monobehaviour.OnMouse_	46.90	0.00	1.00	0 b	5.52	0.00
PreUpdate.SendMouseEvents	46.90	0.00	1.00	0 b	5.52	0.00
Physics.Raycast	46.60	9.10	1.00	0 b	5.49	1.24
GameAgent.Update()	36.20	33.80	400.00	86.8 KB	4.41	3.87
AstarPath.Update()	22.40	13.00	1.00	102.1 KB	3.48	1.97
AbilityAgent.Update()	23.30	22.90	400.00	308 KB	3.27	3.22
Physics.SyncColliderTransform	29.40	26.20	1.00	0 b	3.01	2.67
PostLateUpdate.FinishFrameRendering	20.10	0.50	1.00	400 B	1.76	0.05
Calling Path Callbacks	10.30	1.40	1.00	70.5 KB	1.58	0.21
Physics.SyncRigidbodyTransform	16.40	12.00	1.00	0 b	1.56	1.16
Camera.Render	17.00	1.10	1.00	0 b	1.52	0.11
Running Path Modifiers	8.90	8.90	128.00	8.5 KB	1.36	1.35
Drawing	12.80	0.60	1.00	0 b	1.10	0.05
Render.OpaqueGeometry	12.20	0.10	1.00	0 b	1.05	0.01
RenderForwardOpaque.Render	10.00	0.30	1.00	0 b	0.86	0.02
PostLateUpdate.UpdateAllRenderers	6.60	0.10	1.00	0 b	0.61	0.00
UpdateRendererBoundingVolumes	6.50	4.50	11.00	0 b	0.60	0.41
WorldObjectAgent.Update()	5.80	5.80	400.00	0 b	0.57	0.57
WaitForJobGroupID	6.00	2.50	2.00	0 b	0.55	0.22
PreLateUpdate.ScriptRunBehaviourLateUpdate	3.90	0.00	1.00	0 b	0.53	0.00
LateBehaviourUpdate	3.90	0.00	1.00	0 b	0.53	0.00
NetworkManager.LateUpdate()	3.80	3.80	1.00	0 b	0.52	0.52
TransformChangedDispatch	4.70	4.30	4.00	0 b	0.44	0.39

File path

\Build\profiler_data_type2.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Shadows.RenderShadowMap	4.90	1.00	1.00	0 b	0.43	0.08
RenderForward.RenderLoopJob	4.20	2.40	1.00	0 b	0.37	0.21
GUI.Repaint	4.00	2.20	1.00	400 B	0.34	0.19
UIEvents.IMGUIRenderOverlays	4.00	0.00	1.00	400 B	0.34	0.00
Shadows.RenderJob	3.10	0.00	2.00	0 b	0.27	0.00
Shadows.RenderJobDir	3.10	1.10	2.00	0 b	0.27	0.09
GC.Alloc	1.60	1.60	12418.00	506 KB	0.24	0.24
UpdateDepthTexture	2.50	0.80	1.00	0 b	0.23	0.10
CommandAgent.Update()	2.30	2.30	400.00	0 b	0.23	0.23
Shadows.PrepareShadowmap	2.20	1.10	1.00	0 b	0.22	0.09
UGUI.Rendering.UpdateBatches	1.90	0.60	1.00	0 b	0.18	0.08
EarlyUpdate.PollPlayerConnection	2.20	0.00	1.00	0 b	0.18	0.00
ProfilerConnection Poll	2.20	2.20	1.00	0 b	0.18	0.18
Culling	1.90	0.30	1.00	0 b	0.18	0.03
PostLateUpdate.PlayerUpdateCanvases	1.90	0.00	1.00	0 b	0.18	0.00
UIEvents.WillRenderCanvases	1.90	0.00	1.00	0 b	0.18	0.00
CullResults.CreateSharedRendererScene	1.70	1.20	1.00	0 b	0.17	0.10
EarlyUpdate.UpdateCanvasRectTransform	1.30	1.30	1.00	0 b	0.16	0.16
BatchRenderer.Flush	1.90	1.40	27.00	0 b	0.16	0.12
RenderForwardOpaque.Prepare	1.60	1.60	1.00	0 b	0.16	0.16
ExtractRenderNodeQueue	1.50	1.50	2.00	0 b	0.15	0.15
Canvas.SendWillRenderCanvases()	1.70	0.00	1.00	0 b	0.15	0.00
Layout	1.70	1.60	1.00	0 b	0.15	0.15
Profiler.CollectGlobalStats	1.40	0.70	1.00	0 b	0.15	0.09
PostLateUpdate.ProfilerEndFrame	1.40	0.00	1.00	0 b	0.15	0.00
DepthPass.Job	1.50	0.70	1.00	0 b	0.14	0.06
SceneCulling	1.50	0.10	1.00	0 b	0.14	0.01
CullSendEvents	1.30	0.00	1.00	0 b	0.12	0.00
EventSystem.Update()	1.20	1.20	1.00	0 b	0.12	0.12
Batch.DrawDynamic	1.20	1.20	11.00	0 b	0.12	0.12
RenderForwardOpaque.CollectShadows	1.30	0.70	1.00	0 b	0.11	0.06

File path

\Build\profiler_data_type2.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
Material.SetPassFast	1.30	0.40	18.00	0 b	0.11	0.04
JobAlloc.Overflow	1.10	1.10	2.00	0 b	0.10	0.10
PostLateUpdate.PhysicsSkinnedClothBeginUpdate	0.60	0.60	1.00	0 b	0.10	0.10
DestroyCullResults	1.10	0.00	1.00	0 b	0.10	0.00
TempAlloc.Overflow	1.10	0.00	2.00	0 b	0.10	0.00
PreUpdate.NewInputUpdate	1.00	1.00	1.00	0 b	0.10	0.10
Material.SetPassUncached	1.20	1.20	14.00	0 b	0.10	0.10
PhotonHandler.Update()	1.00	0.10	1.00	0 b	0.09	0.01
RenderLoop.CleanupNodeQueue	1.10	1.10	3.00	0 b	0.09	0.09
CullAllVisibleLights	0.60	0.20	1.00	0 b	0.09	0.01
NetworkAgent.Update()	0.60	0.60	400.00	0 b	0.09	0.09
SendOutgoingCommands	0.90	0.90	1.00	0 b	0.08	0.08
Batch.DrawInstanced	1.00	1.00	1.00	0 b	0.08	0.08
Profiler.CollectMemoryAllocationStats	1.00	1.00	1.00	0 b	0.08	0.08
Camera.ImageEffects	1.00	0.00	1.00	0 b	0.08	0.00
FindActiveLights	0.60	0.60	1.00	0 b	0.08	0.08
Graphics.Blit	1.00	0.90	1.00	0 b	0.08	0.08
Render.Prepare	0.90	0.90	1.00	0 b	0.07	0.07
FPSDisplay.Update()	0.70	0.70	1.00	0 b	0.06	0.06
UIEvents.CanvasManagerRenderOverlays	0.70	0.00	1.00	0 b	0.05	0.00
UGUI.Rendering.RenderOverlays	0.70	0.50	1.00	0 b	0.05	0.04
Shadows.CollectShadows	0.60	0.50	1.00	0 b	0.05	0.04
AddDirectionalLights	0.30	0.30	1.00	0 b	0.04	0.04
EarlyUpdate.NewInputBeginFrame	0.50	0.20	1.00	0 b	0.04	0.04
PostLateUpdate.MemoryFrameMaintenance	0.50	0.50	1.00	0 b	0.04	0.04
GUIUtility.BeginGUI()	0.40	0.40	3.00	400 B	0.04	0.03
PostLateUpdate.InputEndFrame	0.20	0.20	1.00	0 b	0.04	0.04
GUIUtility.EndGUI()	0.50	0.50	3.00	0 b	0.04	0.04
BasicCommandChainBuffer.Update()	0.60	0.60	1.00	0 b	0.04	0.04
Device.Present	0.50	0.00	1.00	0 b	0.04	0.00
Graphics.PresentAndSync	0.50	0.00	1.00	0 b	0.04	0.00

File path

\Build\profiler_data_type2.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Profiler Data E:

Statistics Max Values Calculate

Function	Total	Self	Calls	GC Alloc	Time ms	Self ms
NativeInputSystem.NotifyUpdate()	0.50	0.50	2.00	0 b	0.04	0.04
PostLateUpdate.PresentAfterDraw	0.50	0.00	1.00	0 b	0.04	0.00
PostLateUpdate.UpdateCanvasRectTransform	0.30	0.30	1.00	0 b	0.03	0.03
MouseController.OnGUI()	0.30	0.30	2.00	0 b	0.03	0.03
EarlyUpdate.UpdateInputManager	0.40	0.40	1.00	0 b	0.03	0.03
Canvas.BuildBatch	0.30	0.30	1.00	0 b	0.03	0.03
FPSCounter.Update()	0.40	0.40	1.00	220 B	0.03	0.03
Initialization.XREarlyUpdate	0.30	0.30	1.00	0 b	0.03	0.03
WatermarkRender	0.40	0.10	1.00	0 b	0.03	0.00
PostLateUpdate.EnlightenRuntimeUpdate	0.30	0.00	1.00	0 b	0.03	0.00
Canvas.RenderOverlays	0.40	0.20	2.00	0 b	0.03	0.02
Render.TransparentGeometry	0.30	0.20	1.00	0 b	0.02	0.02
PreUpdate.AIUpdate	0.20	0.00	1.00	0 b	0.02	0.00
Shadows.Sort	0.10	0.10	2.00	0 b	0.02	0.02
Flare.Render	0.30	0.30	1.00	0 b	0.02	0.02
FixedUpdate.Physics2DFixedUpdate	0.30	0.30	2.00	0 b	0.02	0.02
EnlightenRuntimeManager.PostUpdate	0.20	0.10	1.00	0 b	0.02	0.01
EarlyUpdate.PerformanceAnalyticsUpdate	0.30	0.30	1.00	0 b	0.02	0.02
PostLateUpdate.UpdateResolution	0.20	0.20	1.00	0 b	0.02	0.02
Shadows.CullingCallbacks	0.20	0.20	1.00	0 b	0.02	0.02
PostLateUpdate.PlayerEmitCanvasGeometry	0.10	0.00	1.00	0 b	0.01	0.00
PostLateUpdate.PlayerSendFrameComplete	0.00	0.00	1.00	0 b	0.01	0.00
CanvasScaler.Update()	0.00	0.00	2.00	0 b	0.01	0.01
PreLateUpdate.ParticleSystemBeginUpdateAll	0.00	0.00	1.00	0 b	0.01	0.00
ResourceManager.RePopulatePool() [Coroutine: System.Collections.IEnumerator.get_Current]	0.10	0.10	168.00	0 b	0.01	0.01
UGUI.Rendering.EmitWorldScreenspaceCameraGeometry	0.10	0.10	1.00	0 b	0.01	0.01
KeyboardController.Update()	0.10	0.10	1.00	0 b	0.01	0.01
Profiler.CollectAudioStats	0.10	0.00	1.00	0 b	0.01	0.00
AudioProfiler.CaptureFrame	0.10	0.10	1.00	0 b	0.01	0.01
PostLateUpdate.PlayerSendFrameStarted	0.10	0.00	1.00	0 b	0.01	0.00
Profiler.CollectDrawStats	0.10	0.10	1.00	0 b	0.01	0.01

File path

\Build\profiler_data_type2.json Open Edit

Export

Profiler Data | Current Frame Data | Selected Function Data

Bibliography

[1] "Polygon mesh," https://en.wikipedia.org/wiki/Polygon_mesh.

[2] "Boxing and unboxing (c programming guide)," https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/boxing-and-unboxing.

[3] "Prototype-based programming," https://en.wikipedia.org/wiki/Prototype-based_programming.

[4] Microsoft, "Value types (c reference)," https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/value-types.

[5] ——, "Reference types (c reference)," https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-types.