Bachelor Project:

Platformer Generation AI

Authors: Christer Peltoperä Somby
Henning Einar Luick
Jonas Dalheim Reitan
Kristoffer Eidså

Date: 18.05.2015

Supervisors: Mariusz Nowostawski
Simon McCallum

Table of Contents

Preface

Thanks to Mariusz Nowostawski for being our supervisor, helping us with the bachelor thesis whenever we needed it, and giving us directions so we were headed the right path. Thanks to Simon J. R. McCallum for answering bachelor thesis related question that we had. Thanks to Ådne Midtlin for being available for business related questions. We would also like to thank everyone that participated in the testing of our algorithm.

List of Figures

List of Tables

List of Listings

Sammendrag

Sammendrag av Bacheloroppgaven

Tittel	Platformer Generation AI
Dato	18.05.2016
Deltakere	Christer P. Somby
	Henning E. Luick
	Kristoffer Eidså
	Jonas D. Reitan
Veileder	Mariusz Nowostawski
Oppdragsgiver	Kremen
Kontaktperson	Jonas Dalheim Reitan Jonasdr@hotmail.com +4791871237
Nøkkelord	Thesis, AI, Game Engine, Programming, C++, Level Generation
Antall sider	?
Antall vedlegg	
Tilgjengelighet	Åpen
Sammendrag	Kremengine er en spillmotor skrevet fra grunnen av i C++14 . Den støtter både 3D og 2D. Vi laget vår egen motor slik at vi kunne ha maksimal kontroll over nivå genererings algoritmen og redusere tiden det tar å teste ting siden vi visste hvordan alt fungerte. Det er mange forskjellige måter å takle oppgaven vi satt for oss selv. Vi eksperimenterte med mange forskjellige algoritmer for å lære datamaskinen hvordan å generere gode nivåer konsekvent. Det vi endte opp med er en nivå generator som bruker linje for linje framgangsmåten for bakken spilleren skal gå på og støy fremgangsmåten for taket i verdenen våres.

Abstract

Abstract for Bachelor Project

Title	Platformer Generation AI
Date	18.05.2016
Participants	Christer P. Somby
	Henning E. Luick
	Kristoffer Eidså
	Jonas D. Reitan
Supervisor	Mariusz Nowostawski
Employer	Kremen
Contact Person	Jonas Dalheim Reitan Jonasdr@hotmail.com +4791871237
Keywords	Thesis, AI, Game Engine, Programming, C++, Level Generation
Pages	?
Attachments	
Availability	Open
Abstract	Kremengine is a game engine written from scratch in C++14. It supports both 3D and 2D. We made our own engine so we could have maximum control of the level generation algorithm and reduce time it take to test things as we knew how everything worked. There is a lot of different ways to approach the task we set for ourselves. We experimented with a lot of different algorithms to learn the computer how to generate good levels consistently. What we ended up with is a level generator that uses the line by line approach for ground for the player to walk on and the noise approach for the ceiling of our world.

Introduction

Project Description

The project we are doing is very interesting and can be complex, what we really wanted to work with was something that learns, that is why we are making a level generator that can learn how to make better levels. The level generator uses user feedback to get a general sense of how the level was, if it was bad, it will down prioritize the patterns that do not work together, likewise, if it was good, if will prioritize the patterns that work together. The reason we chose to do user feedback rather than doing self learning by observing the players behavior as a primary way of getting data, is because user interaction can be both very enjoyable coupled with the brainpower and independence of humans. Seeing something evolve through the playing of many people is something that is fascinating to observe and can also be a powerful tool for level making in games. Just being able to see what the best level is for players of all ages and all difficulties that can be directly used or be the influencer of great level making.

Background

The bachelor group consists of four game programming students who intend to start a game development company after finishing our degree. Because of that we wanted to built a flexible and expandable game engine during last semester in the game programming course, with the intent of writing our bachelor assignment in it. During this semester we focused everything on the game engine, so we did not give the bachelor much thought.

We had a few meetings during the second week of January where we discussed what we wanted to do, and we seemed to be most interested in working with AI or virtual reality. None of us owned the proper hardware to work efficiently with virtual reality so we quickly settled for some research and pioneering in the field of AI. We ended up settling for some form of level generation AI. After discussing this with Associate Professor Simon J. R. McCallum and Associate Professor Mariusz Nowostawski, we decided on creating an AI which would eventually generate generally good levels. The AI would achieve this by sending what it considered good levels to voluntary participants, who in turn would give feedback back to the server and the AI would learn from that.

All four of us have different areas of interest and experience within game development, and the project is fairly large, so we found this to be the perfect fit for us.

Why we chose to do this

Many big companies pay video game testers[1] to test their game and level design, because there is no precise definition of good level design. The quality of a level or a game is subjective, but that do not mean that there is not a generally good way of

building them. The tool we are writing hopes to discover the generally good way to build a generic platform level.

There are many examples of smart level generation, but most of these are using a self written algorithm with no external learning. Our tool will generate better levels as more people use it and give feedback, and will hopefully eventually be able to generate close to perfect generic platform levels.

We were allowed to be four on this project because of its scope and complexity, in addition to the fact that it is a fairly unexplored subject. We are planning to start a game development company after we are are finished with the degree, which is an additional reason for developing this tool, as it could be used by us among others in the future to speed up the development process of platform games.

Academic Background

The entire team is studying game programming at NTNU Gjøvik. We have all made several prototype games as well as a game engine during our study. All members are proficient with c++. Most members also have some experience with other languages. We have made games for both Windows and Android. We are all Windows users, so multiplatform support is very limited.

Audience

Demo Audience

The demo will be used to demonstrate what we have done for NTNU in Gjøvik. It will showcase the strengths and weaknesses of the generation algorithm, and demonstrate how it can be used in a real business setting. Gjøvik University College can use it as advertisement to show off what their students are capable of after a completed bachelor degree. For us it will be a project we can add to our portfolio when searching for a job.

Thesis Audience

The thesis is created for NTNU in Gjøvik for research and teaching. It will also be used by scholars who wish to delve deeper into the generation than we have. We will explain most of the terms in great detail, so the thesis will be understood by most people who have no prior experience in the field of artificial intelligence or world generation. See section Development Tools for full overview of the tools used in this project.

Product Audience

The product is aimed towards game creators and academics. For game programmers it

could be a good tool for them to use to find out how their levels evolve or use it to make their own generation by using out input. For academics, it can be interesting to see our findings and possibly learn from it and maybe even extend it.

Document Structure

Summary of the chapters in the document:

1. Introduction: Introduction to the document, background of the project, the projects framework, terminology and the audiences for the product.
2. Requirements: Describes the requirements we set for the program.
3. Technical Design: Describes the technology, system architecture, program flow and subsystems.
4. Development Process: Describes how we worked, what tools we used and our work hours.
5. World Generation: Describes the different approaches that were attempted, how world collision is handled and what our final choice of algorithm was.
6. User feedback and testing: Describes our methods of aquiring user feedback and the testing process.
7. Deployment: Describes how the installer was built.
8. Discussion: Describes our work and possibilities of further development.
9. Conclusion: Contains a summary of what was done.

Roles

We have no team leader for the project. The reason for that is so we could see how well we work together as a team and how fast we are at arriving at a decision. Christer P. Somby did a lot of odds and ends, working on some level generation, console, installer and database. Henning E. Luick did a lot of stuff to sort out issues in the engine, networking and level generation. Henning's level generation idea was the one that was chosen. Jonas D. Reitan did pathfinding, level generation using noise, engine fixes and graphics. Kristoffer Eidså mostly worked on database integration and user feedback, but he also worked on GUI, engine and level generation. Our supervisor is Mariusz Nowostawski, he helped us with ideas on level generation and all the questions we needed answering on the problems we encountered during the project.

Development Framework

Creating a tool for level generation can be something exciting to work on that requires you to change the projects direction in order to better accomplish the goal by using other types of algorithms. The goal of the project is very ambitious considering the amount of time needed to research different methods and try them. So we neede a framework that

can help us keep track and steer us towards the end goal. That is why we chose to use Scrum as our framework. [2]

The scrum sprints are set to last one week each, with weekly spring meetings to review and refill the sprint backlog. The meetings are held every friday at 16:00. Daily scrums are also held to keep people updated on the progress of the project. The sprint leader was usually Henning E. Luick, he took the initiative to keep us on track when we held the meetings.

A gantt chart was set up so we could keep track of when we should be at a certain point. It is found in Appendix A. There are 4 milestones that to be completed. The earliest prototype was set there so we had something we could show and see if it truly was progressing. The later milestones were for collecting data to see the progress of the algorithm and see whether or not we needed to modify something to progress towards the goal.

Terminology

- IDE - Integrated Development Environment (ex. Visual Studio)
- SDL - Simple DirectMedia Layer. Used to handle low level events from the operating system.
- 2D - Two-Dimensional.
- 3D - Three-Dimensional.
- XML - Extensible Markup Language.
- CPU - Central Processing Unit. Processes computer instructions.
- GPU - Graphics Processing Unit. Processes graphics to be displayed on a display.
- OpenGL - Open Graphics Library. Used to send data to the GPU.
- GUI - Graphical User Interface.
- AI - Artificial Intelligence. Create intelligent software like learning or pathfinding.
- ANN - Artificial Neural Network. Uses neural networks to learn machines.
- MDP - Markov Decision Processes. Mathematical framework for modeling decision making.

Introduction references

[1] Wikipedia. (2016). *Game testing*. [online] Available at: https://en.wikipedia.org/w/index.php?title=Game_testing&oldid=714107300#cite_ref-Bethke52_8-0 [Accessed 18 May 2016].
[2] Wikipedia.org (no date). *Scrum (software development).* [online] Available at: https://en.wikipedia.org/wiki/Scrum_(software_development) [Accessed 18 may 2016].

Requirements

There are some requirements that we have of our code base and program. For the code base the code should be well written, easily maintainable and expandable and well commented. For the level generation we require it to be fast, it can not have it be slow. If it is slow, people will get impatient in between levels, therefore it has to have a clearly defined maximum time it can use to generate a level. For the database, we want it to have expandable space aswell as being reliable. The program should also not crash and be reliable to use.

Technical Design

Technology

The entire system is written in C++, using the C++14 standard. All coding is done in Visual studio 2015. We freely use all features and shortcuts offered by Visual Studio 2015 as we want to familiarize us with this IDE. If the C++ standard library supports a required feature, this should be used rather than any third party library.

The program uses SDL 2.04 to create the programs window and manage input. SDL_Mixer is used for the audiosystem. OpenGL is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The OpenGL version we are using is version 4.5, but support for earlier versions are also there. The system uses XML-files(Extensible Markup Language) for loading actors. We use TinyXML-2 for parsing of XML-files. Notepad and Notepad++ is used to manually modify XML-files. Box2D is the physics system used. Other libraries used are described in the sections about the systems using them.

Table 1:

Library	Short Description
SDL2	Window and input management
SDL_Mixer	Audio
SDL_Font	Handles text rendering
std_image	Image loading library supporting png, jpeg and gif
OpenGL	Graphics renderer
glu	OpenGL utility Library
glew	OpenGL Extension Wranger Library
glm	OpenGL Math. Math library
Box2D	Physics system
libnoise	Noise generation library
Raknet	Networking
mysqlcppconn	MYSQL database driver library
tinyxml2	XML parser
lua	Scripting language

Program Flow

The program starts by initalizing all core components which require initialization. Renderer, external data handlers and subcomponents like the particle system and the database are initialized here.

After general initialization , the application will run through a loop held by the run() function inside the Engine class. The loop checks for the current application State and inititializes said game state. Initialization of application states are mostly just changing the Graphical User Interface. Once the initialization is done, the run() functions calls on a separate loop which is responsible for updating the current state.

The most important application state to update is the running state. This state is where all the game logic and gameplay happens. The other states are mostly traversing menus.

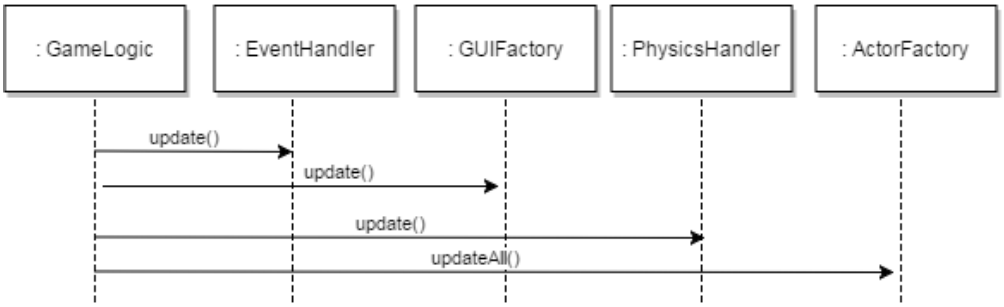


figure 9.1: flow of logic layer

Subsystems

This section describes the game engine's systems. To see the actual API documentation for each system with all the methods and data, refer to the doxygen documentation.

ActorFactory

Engine has one ActorFactory. The ActorFactory is responsible for creating and deleting all drawn objects other than GUI and particles on the screen. We will call these objects entities or game objects. These entities are defined by the Actor class. Actors use the entity-component-system where each component is an instance of the ActorComponent class. Components are identified by its name and can easily be accessed by the actor or

sibling components. In addition to creating actors, the ActorFactory is also responsible for creating components for each actor. This is done by using component factories, and is covered under actors.

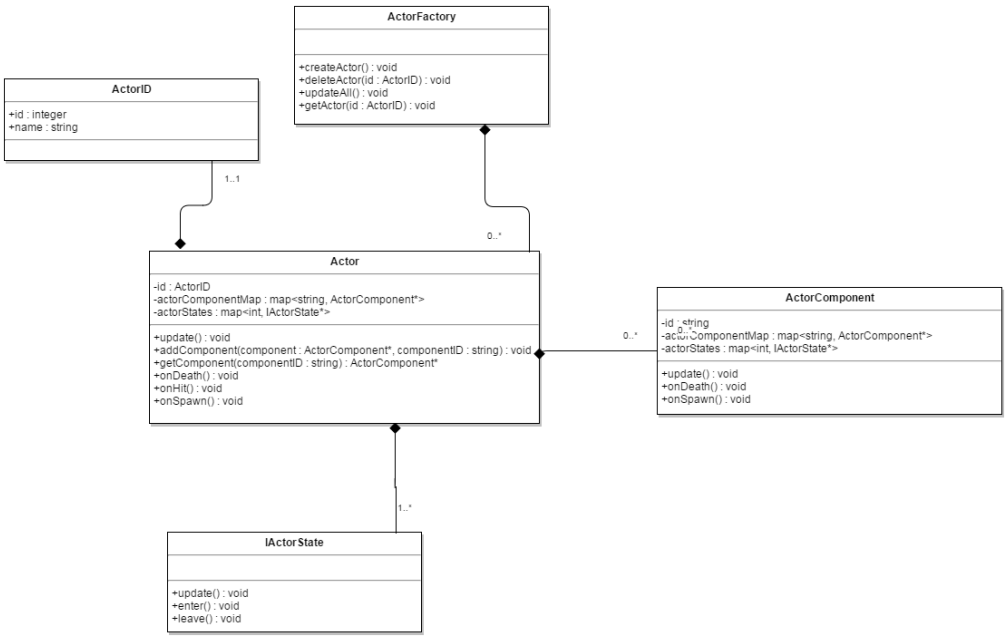


Figure 9.2: ActorUML

Actors

All actors in our program are loaded from XML-files, but it is possible to create actors purely from c++ code if the need rises. You may want to code an actor in c++ if it needs a special relationship to a class outside of Kremengine. The reason for using XML files is that you do not need to recompile the source code for each change, and it also makes it easier to make small changes to actors clutters the code less.

All components used by the actor and details regarding the components are specified in the XML-file when loading from file. Every piece of data is stored inside XML-elements. This is used to make it as easy as possible to extend, with no regards to efficiency. XML-elements are represented by

Listing 9.1: XML-Element example

```
<ElementName> elementValue </ElementName>
```

Components are generally XML-elements with XML-elements inside.

Listing 9.2: A GraphicsComponent example

```
<GraphicsComponent>
 <sizeX>1.0</sizeX>
 <sizeY>1.0</sizeY>
 <sizeZ>0.001</sizeZ>
 <xFrames>4</xFrames>
 <yFrames>2</yFrames>
 <animated>true</animated>
 <meshVerticesPath>res/models/cube.obj</meshVerticesPath>
 <texturePath>res/textures/characters/jakkheim/walking.png</texturePath>
 <shader>res/shaders/basicAnimationShader</shader>
</GraphicsComponent>
```

We have no contingency for faults in the XML-syntax or element contents. The one creating the XML file is responsible for making sure that the syntax and content is correct. Faults in syntax or content may lead to a crash in the application, depending on the severity of the error. All components have default values where it makes sense. An empty physicsComponent will give you a 1 by 1 square box, where all sensors are disabled and there is no rotation, which is what we consider default. However, an empty AudioComponent will not set any default sounds, making the component essentially useless on its own. An exception to this rule is if you have set a category on the component, which is covered below.

Listing 9.3: Example XML-file of an actor

```
<Actor>
 <Type>MovingObject</Type>
 <ActorName>TimedBlock</ActorName>
 <NetworkLevel>1</NetworkLevel>
 <Components>
  <GraphicsComponent>
   <sizeX>1</sizeX>
   <sizeY>1</sizeY>
   <sizeZ>1</sizeZ>
   <animated>true</animated>
   <meshVerticesPath>res/models/cube.obj</meshVerticesPath>
   <texturePath>res/textures/timedBlock.png</texturePath>
   <shader>res/shaders/basicAnimationShader</shader>
  </GraphicsComponent>
  <PhysicsComponent>
   <category>TimedBlock</category>
   <collisionCategory>Object</collisionCategory>
   <shape>box</shape>
   <bodyType>1</bodyType>
   <density>1</density>
   <xInPixels>1</xInPixels>
   <yInPixels>1</yInPixels>
   <restitution>0</restitution> <!--bouncyness-->
   <friction>0.0</friction>
   <gravity>0</gravity>
   <fixedRotation>true</fixedRotation>
  </PhysicsComponent>
 </Components>
</Actor>
```

As mentioned earlier, ActorFactory is also responsible Components through component factories. Component factories have to be registered for the ActorFactory to know what kind of components it is allowed to create. Kremengine registeres all the default components (e.g. graphics, physics and audio) of any game, but custom components can easily be registered should the need arise.

listing 9.4: registering a new component

```
ComponentFactoryCreatorImplementation<FlashOnHitComponent>
flashOnHitFactory("FlashOnHitComponent",
engine.getActorFactory()->getComponentFactoryFactory());
```

In addition to creating custom components, one can also create custom categories on any component. Creating and setting custom categories gives you the option to override key functions of this component. This is very useful for game logic or simple AI. Custom categories have to be registered after the Component is registered.

listing 9.5: registering a new component category for the physics component

```
ComponentCreatorImplementation<PhysicsTimedBlock>
physicsComponent5("TimedBlock",
engine.getActorFactory()->getComponentFactoryFactory()->getFactory("Physi
csComponent"));
```

A component with a custom category can still call on all meaningful methods of the parent.

listing 9.6: overridden update function still calls parent

```
void PhysicsMovingPlatform::update(float deltaTime) override
{
 PhysicsComponent::update(deltaTime); //Here the parent function is being
called from a child

 //below here is code that is specific for the MovingPlatform actor
 float pos = std::cos(this->position += (this->speed * deltaTime));
 if (this->position >= glm::two_pi<float>())
 {
  this->position -= glm::two_pi<float>();
 }
 body->SetLinearVelocity({ this->direction.x * pos, this->direction.y *
pos });
}
```

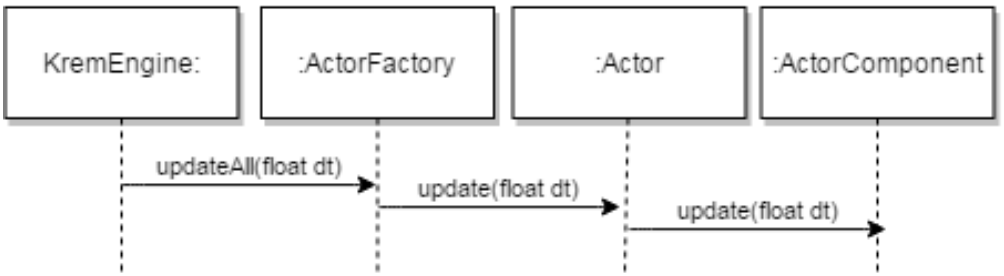


figure 9.3: sequence for actor updating without a custom category

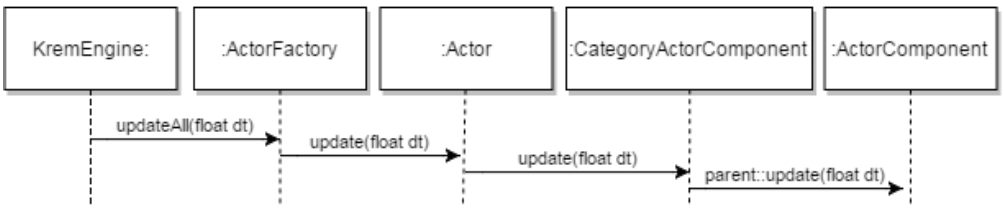


figure 9.4: possible sequence for actor updating with a custom category

ActorStates

ActorStates are states for actors to be in. Every actor has to be in exactly one state at any given time. The states are in a finite state machine.

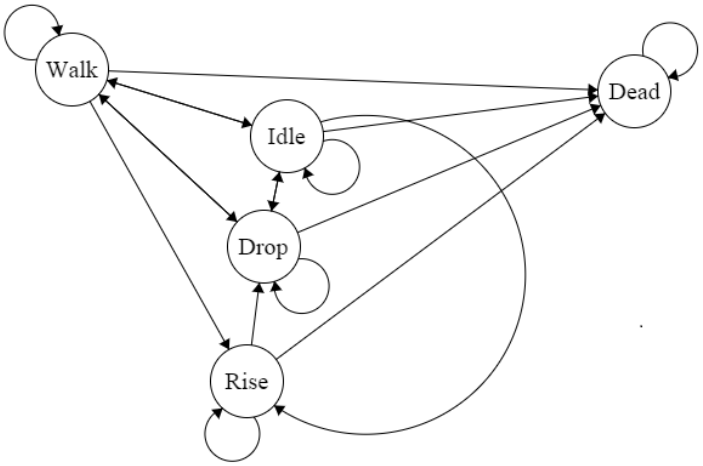


figure 9.5: State machine for actors. Figure created using [4]

The states are designed to handle details surrounding an actor and its movements. An dropping actor may move slower horizontally than a walking one for instance. The state of the actor depends mostly on the velocity. The exception to this is if the actor is dead. An actor is in the dead state when his health drops to zero or is set to be dead manually. The primary responsibility of these states is to make sure that the correct animation is being played for the actor. More features are in place, but are not currently in use.

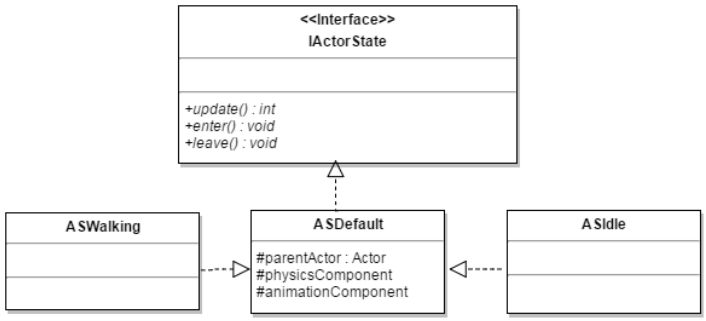


figure 9.6: Actor state hierarchy

The state machine and method can be expanded upon. One might want a concurrent state machine in a more complex game. A concurrent state machine handles multiple concurrent states. An example where this would be useful could be where the character has to act differently depending on e.g weather or equipment, which can not be handled in a global actor state. Game programming patterns[3] was very helpful in learning more about state machines.

Animations

Animations are something we use to change the representation of an object, either at a per frame basis or per action. We use this to give the game more life by having actors look like they are walking by animating their feet or by giving feedback to the user that a button has been pressed.

We use per frame on objects that we want to look like humans performing an action, like

walking, and we use per action on objects that only change when a certain action occurs, for example a mouse click on a button.

The way this system works is by telling the graphical processing unit what part of the texture to sample from when drawing the actors to the screen. We can tell it to sample from texture coordinate 0,0, which will choose the texture in position 0,0 in the tile atlas (see figure 9.7) which would draw a black square.

figure 9.7: example of an tile atlas.

Instead of drawing colored boxes we used it to draw characters walking by simply looping through for example [0,0] up to [4,0], which would give us 0 in the y-axis and 0 to 4 in the x-axis. This will create the illusion of human like actions of the character in the game even if he is just sliding across the floor logic-wise.

The following code block is the shader code we used to tell the graphical processing unit what to sample from the texture.

Listing 9.7: Variables

```
int columns = int(size.x);
 float x = spriteNo % columns;
 float y = spriteNo / columns;
 vec4 texel = texture2D(texture0, (vec2(x, y) + vec2(1.f, 1.f) *
texCoord) / size);
```

It is mathematically figuring out what coordinate the "spriteNo" corresponds to and draws that texture onto the position of the actor on screen. All textures in are in coordinates from [0,0] to [1,1] so we need to divide the texture coordinate by the amount of pictures in the tile atlas on the horizontal and vertical plane, this means that the boxes on figure 9.7 would be divided up to 1/5, or 0.2 per texture. Given the spriteNo 7, we would get [2,1], which would give us the texture coordinate corresponding to the yellow box with a star inside because it is counting from 1 instead of 0.

Everything in the game that need a per frame animation uses the animation class. This class is responsible for telling the animation system what frame to start on, how many frames to animate and how fast to animate them. It consists of a very simple update and getFrame function.

Listing 9.8: animation class

```
void Animation::update(float deltaTime)
{
 this->currentFrame += this->animationSpeed * deltaTime;
}

int Animation::getFrame()
{
 return this->firstFrame + (static_cast<int>(this->currentFrame) %
this->amountOfFrames);
}
```

The update function is responsible for updating the animation per frame by the amount we want, so we can tell it to loop through its entire animation in how many frames we want. For example loop through all of its 5 frames every second. We use getFrame() to figure out what frame we want to display at the current frame the game is currently rendering, which is sent to the animation subsystem. This function takes the offset frame into the tile atlas, named firstFrame in code, and adds the frame it is currently trying to display. An example of this could be a tile atlas with 30 frames and we only want to display the last 10, so we set the firstFrame to be 20 and give it 10 frames to loop through.

Console

Writing the console was a venture into finding a way to make a console dynamic, having only one call for the program to handle. It did not end up being fully implemented, but the planned implementation was also more time consuming compared to doing it the easy way. Which would be to add a parser that just checks the command sent in to a function full of if statements that can call any function that you wanted to add.

The storing of the functions is placed into a class that has the appropriate functions for inserting and searching and calling. It stores any type of class function pointer, and those class functions have to belong to the same class. So a class to store the functions was needed. That class stores a copy of the Engine object, from which it can access most systems within the program. This class' object is stored together with the class function pointer storage object.

Listing 9.9: Function pointer storage

```
typedef void(SomeClass::*funcPtr)(void);
std::map<std::string, std::pair<funcPtr, std::type_index>> pairFuncs;
```

In listing 9.9 we see a type definition for a class function pointer which is call funcPtr

that takes void as an argument. This means that when we store the function pointer, the arguments that the function has is not passed along to the storage of the funcPtr variable. This means that we have to store away the type of function. That is where the pair in the map comes in. It stores the function pointer together with the type of the function.

Listing 9.10: Insertion function to map

```
template<typename func>
void insert(std::string mappingName, func f1)
{
 //find the type of the function pointer
 auto typeT = std::type_index(typeid(f1));
 //pair the function pointer with the type
 pairFuncs.insert(std::make_pair(mappingName,
std::make_pair(reinterpret_cast<funcPtr>f1, typeT)));
}
```

This templated function in listing 9.10 is for inserting the class function pointers into the map that stores the class function pointer and the type of the class function pointer. The typeT variable is to temporarily store and find the type of the function, it stores the arguments of the function. The function then inserts the name of the function and pairs it with the class function pointer after casting it to a function that takes no arguments and pairs it with the type of class function pointer.

Listing 9.11: Search and call function

```
template<typename Type, typename R, typename... Args>
void searchAndCall(std::string mappedName, R object, Args&&... args)
{
 auto mapIter = pairFuncs.find(mappedName);
 auto mappedPair = mapIter->second;
 //Cast the general functionpointer(void) to the type it really is
 auto typeCastedFunc = reinterpret_cast<Type(SomeClass::*)(Args
...)>(mappedPair.first);
 assert(mappedPair.second == std::type_index(typeid(TypeCastedFunc)));
 (object->*TypeCastedFunc)(std::forward<Args>(args)...);
}
```

The searchAndCall function in listing 9.11 takes in the return type of the class function pointer, the object that you want to call the function from and the name of the function together with any arguments that it had originally, like you would send variables into any other function. The functions that are used by a console usually do not return anything, therefore the return type is usually void, but it was written like that because it can be used for various other systems that use function pointers. The first thing the function

does is to find the function within the map, then it finds the pair that is stored within the map that contains the class function pointer and the type of the function and stores it into the mappedPair variable. It then casts the class function pointer back to the correct type. If the type is void, the "Args" are 4 and 6, which are both ints the typeCastedFunc would end up being "void SomeClass::*(int, int)". It the figures out if the typeCastedFunc is the same type as the original type of the class function pointer. Then it calls the type casted function using the class object that was passed in together with the function call.

Event

Engine has one EventHandler. The event handler is responsible for taking in any input that is given to the program, like keyboard or mouse. SDL2 is used for taking care of the events. The events that happen, like a key on the keyboard is pressed, the mouse is moved or one of the mouse buttons are clicked, are packed into the message system that is used. As soon as the keys are pressed the events are triggered by sending out the message to the components that are listening to it.

Listing 9.12: Example sending message

```
this->postMessage(Message(this, "keyDown",
static_cast<int>(SDLEvent.key.keysym.sym)));
```

By invoking the postMessage function you pass the object (this), the type of event ("keydown") and the key that was pressed (SDLEvent.key.keysym.sym). The object is the EventHandler. It all gets packaged into a message and sent.

Listing 9.13: Example receive message

```
void receiveMessage(const Message & message) override
 {
  if (message.getSenderType() == typeid(EventHandler))
  {
   if (static_cast<const char*>(message.getVariable(0)) == "keyDown")
   {
    int number = message.getVariable(1);
```

The receiveMessage is where one receives the message and executes what the message wants you to do. The first thing that is done is to check where the message came from, once it finds that it is from the EventHandler, it the checks what type of event it is, the it extracts the information from the package and executes the actions it is told to do.

Graphical User Interface

Like with the actors, the graphical user interface in our engine is also loaded from XML-files and it can also be written by C++ code for special occasions. The XML-system is the same as explained earlier in the actors segment, just with different names and categories. The graphical user interface is divided into states and each states

has different elements that needs to be displayed. An example of a state might be a main menu, this main menu can then have elements like; four buttons, one slider and a panel. The XML-files for the graphical user interface is per state rather than per element

Here is an example of how the xml-file for running state of our program looks like:

Listing 9.14: XML-file example

```
<GuiElements>
 <Button>
   <name>newlevel</name>
   <fontPath>res/fonts/gui.ttf</fontPath>
   <positionX>-0.9</positionX>
   <positionY>-0.9</positionY>
   <widthInPixels>0.1</widthInPixels>
   <heightInPixels>0.1</heightInPixels>
   <meshVerticesPath>res/models/quad.obj</meshVerticesPath>
   <shaderPath>res/shaders/basicAnimationShader</shaderPath>
   <texturePath>res/gui/textures/buttonAtlas.png</texturePath>
   <textureAtlasSize>2</textureAtlasSize>
   <visible>true</visible>
 </Button>
 <Container>
   <name>actionBar</name>
   <positionX>0</positionX>
   <positionY>-0.9</positionY>
   <visible>true</visible>
   <meshVerticesPath>res/models/quad.obj</meshVerticesPath>
   <shaderPath>res/shaders/basicShader</shaderPath>
   <shaderPathBoxes>res/shaders/basicAnimationShader</shaderPathBoxes>
   <texturePath>res/gui/textures/actionbar.png</texturePath>
   <texturePathBoxes>res/gui/textures/inventoryBox.png</texturePathBoxes>
   <textureAtlasSizeX>1</textureAtlasSizeX>
   <textureAtlasSizeY>3</textureAtlasSizeY>
   <numberOfBoxesX>4</numberOfBoxesX>
   <numberOfBoxesY>1</numberOfBoxesY>
   <distanceBetweenBoxes>0</distanceBetweenBoxes>
   <widthInPixelsBoxes>0.1</widthInPixelsBoxes>
   <heightInPixelsBoxes>0.1</heightInPixelsBoxes>
   <borderTop>0</borderTop>
   <borderBottom>0</borderBottom>
   <borderLeft>0</borderLeft>
   <borderRight>0</borderRight>
   <movableItems>false</movableItems>
   <destroyable>false</destroyable>
   <activatable>false</activatable>
   <presetItems>
    <texturePath>res/gui/textures/actionbarCopy.png</texturePath>
    <texturePath>res/gui/textures/actionbarColorPicker.png</texturePath>
    <texturePath>res/gui/textures/actionbarErase.png</texturePath>
    <texturePath>res/gui/textures/actionbarResize.png</texturePath>
   </presetItems>
 </Container>
</GuiElements>
```

More elements could be added to this state by just adding a new <button> or another

element tag below the first one.

The graphical user interface uses an class named GuiFactory. When the program starts all the XML-Files needed are loaded into memory when its state is ran. If the state is left all the objects get deleted but the XML-file is still in memory so it can easily be accessed if the state needs to be loaded again. The factory creates and handles all the states of the graphical user interface by knowing what state the game is in. The XML-files has to be named after its corresponding game state for it to be loaded whit the correct state. After reading the XML-file for the state its going to load the factory starts creating elements that belongs to the state. An element can be anything that the graphical user interface needs to take care of or draw on the screen.
List of elements currently in our engine:

- Bar - Can be used for health/mana bars, loading bars or other bar looking elements.
- Button - Simply a button that is clickable
- Check box - A box that can be either true or false when you click it.
- Drop down menu - A list of different options that can be chosen.
- Container - This can be used for a lot, inventory system, level selection, character selection, etc.
- Panel - This is just a simple texture displayed, mostly used for backgrounds.
- Slider - Used to for example control the volume of the audio.
- Textbox - text input.

Interacting

when interacting with the graphical user interface you expect stuff to happen. Buttons should be clickable and sliders should be movable, and they should also do stuff based on their current state. To accomplish this the graphical user interface needs to have listeners. Currently you need to manually add a listener in the factory to the specific element you want to listen too. You do this by subscribing to an element using its element name. When that specific is interacted with it will send a message to the factory with its new state and then you can run whatever code or function you want from there to make the interaction do something.

Animations

The graphical user interface does not have its own animation component and does not really use real animations but rather it changes its texture when the state of the element changes. For example the button has three different states, clicked, mouse over and not clicked. Each state uses its own kind of texture to look like it has been pressed or moused over. There is support for having the button use animations, for example having the button shine every few second but this is not in use currently.

Particles

Using particles in games makes the game more alive and satisfying to play. We want to give the player the best experience possible so we added a basic particle system to the engine.

This system is used for:

- Weather
- Explosion
- Debris
- Fountains
- Protjectile trail
- Visual feedback to the user

We added a couple of different weather particle systems so we could easily choose between having snow, rain or objects like leaves fall down from the sky. This helps the user to be more immersed in the game and adds to the realism. For explosions we made objects that collided shoot out smaller fragments of itself so it looks like it exploded on impact, this is similar to debris, which would knock of tiny bits of rocks when a rock collided with objects. Fountains were mostly used for experimenting to see how gravity affected particles and giving us a visual cue so we can check if everything is working as expected. Projectile trail was used to give the player some hints of how the projectiles flew through the air and lastly we used it to give users some feedback so if they pick up an object for example, it would shoot out a few particles to confirm that you actually picked up the item and not just made it disappear.

This system is efficient because we can tell the graphics processing unit to draw all the particles in a single pass thereby giving us the flexibility to have millions of particles on screen without slowing the game down in terms of frames per second. Our graphics library, OpenGL, calls this for instancing [1]. Instancing works by giving the graphics processing unit an array of positions and a single mesh to fill all the positions instead of uploading the same mesh with a single position every draw call. This safes a lot of time since the central processing unit is slow compared to the graphics processing unit, so both can run at maximum speed instead of having to wait for data to be transferred between the units.

Listing 9.15: Uploading all positions to the graphics processing unit

```
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER,
this->vertexArrayBuffer[this->POSITION_VB]);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3) * positions.size(),
&positions[0], GL_STREAM_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
```

As you can see from the code block, all we have to do is buffer the vector "positions" to the graphics processor to process and draw as fast as it possibly can. Positions is a vector which holds all the positions in forms of three floats, one for each axis in a three dimensional world, so the graphics processing unit know where to draw and how many to draw as the amount of positions corresponds to the amount of particles we want to draw on screen.

We are not limited to only upload positions as we write our own shaders and we used this to our advantage as we added support for different textures and colors. This way we can for example fill the sky with snowflakes that vary from completely white to a slight shade of blue, giving us a much more realistic visualization of our own little world.

Listing 9.16: Instancing vertex shader with textures

```
void main()
{
 color = colors;
 texCoord = texCoords;
 gl_Position = transform * vec4(vertex + positions, 1.0);
}
```

Listing 9.17: Instancing fragment shader with colors

```
void main()
{
 vec4 texel = texture2D(texture0, texCoord);
 if (texel.a <= 0.5)
 {
  discard;
 }
 colors = color * texel;
}
```

As we can see from this simple shader, all we do is to tell each object what its offset is, which is named "positions" in our shader, and pass the color to the fragment shader which multiplies it with the color from the texture we sampled.

Physics

We tried and wanted to implement the physics through an interface early on during the development of Medieval Brawl. We wanted to do this to make it easy to replace or expand, as we had planned to use the same codebase for the bachelor thesis back then. After toying around and learning the Box2D library it proved to be somewhat finicky to make a general interface for communication, so we scrapped that idea and coupled box2d loosely with the rest of the game logic. The loosely coupling in this sense means that there are very few lines of code which need replacement inside the main codebase to replace the backend physics library. Most of the physics from Kremengine is kept within the PhysicsHandler and PhysicsComponent class. Custom Physics Components are

written for many actors, which have their own set of Box2D code, but this is isolated outside of Kremengine. The update function for the physics component ensures that the graphical transformation is always identical to the Box2D body. The need to have the physical body at a different location than the transformation is supported through the use of Custom Components.

One of the major advantages of coupling the library directly to the engine is that you can freely use all the tools of the library without making methods through the instance. Creating joints or motors can be done by any actor by speaking directly to the library or by using one of the helper functions in the PhysicsHandler class.

The PhysicsComponent class supports any type of shape supported by Box2D, as well as any body type and size supported. Other settings include restitution, friction, density and sensors which prevent or help the actor to achieve certain actions. Physics component also decides which state the actor is in by checking the velocity.

All the positions and references to points in space are refered to in three dimensions, but Box2D is only a two-dimensional physics library. This is done on purpose to make it easy to replace the 2D physics with a 3D one. One team member spent a day learning and replacing Box2D with a 3D physics library for learning purposes, and to see how easy it was.

Renderer

The renderer is responsible for visualizing everything for the user. Its job is rasterization, which means that it takes the completed scene, either three- or two dimensional, and raster every pixel so the monitor can display what the scene actually looks like to the user. Our renderer is built upon OpenGL which provides us with a software abstraction of the graphics processing unit [2].

How our renderer work is that we give every object in the scene a transformation matrix, which contains position, scaling and rotation, which is used to tell the graphics processing unit how to display the mesh in the scene. This matrix is often referred to as the model matrix multiplied with the view matrix even though you are free to call it whatever you want. The view projection is what we usually referred to as the camera, which controls what the user see in the world.

The camera is not a camera as a normal person might think, which shoots pictures and videos, but just a mathematical matrix. This matrix tells the graphics processing unit what field of view it should display, what aspect ratio the monitor has, which reduces image distortion, and how far or near objects should be drawn. This helps with performance as drawing things we do not see is a waste of processing power. All of this is then multiplied with another matrix, which is usually called view matrix, that keeps track of the cameras position, what direction is up and what way it is facing.

All this information is calculated on the central processing unit and is then given to the graphics processing unit to set up the scene and starts rasterizing the image for the user.

Resource Management

The early days of Medieval Brawl stored all data directly on whichever object used it. This means that if an actor needed a sprite, the sprite would be stored directly on the graphics component of said actor. This is obviously very bad as the memory would be spent very inefficiently. We migrated to a solution where all resources of a certain type would be loaded into its respective handler. The result of this is very many handlers which do very similar jobs.

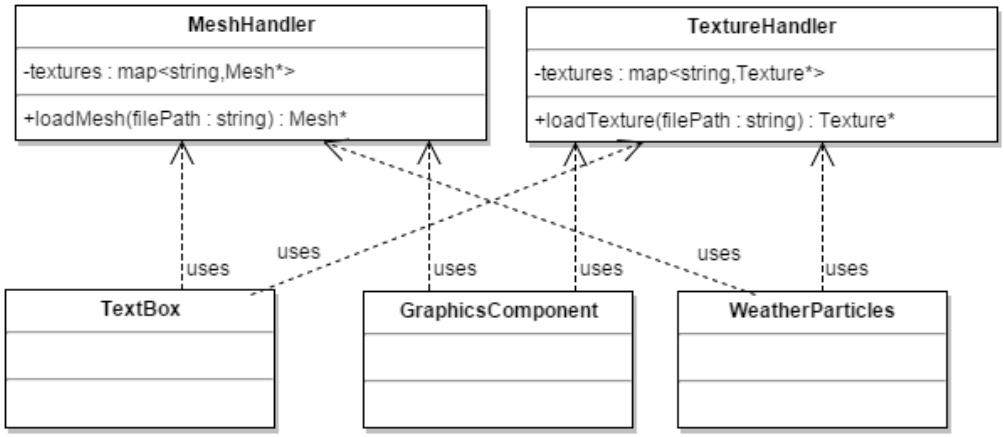


figure 9.8: A simple illustration of our resource management

Each resource handler has a map containing all resources of that type, which can be retrieved by filename.

This system has some flaws, but they have not been prioritized by anyone on the team as the issues are somewhat insignificant for the thesis, but interesting nonetheless. One issue is that there is no limit to how many resources can be loaded, so the application would eventually run out of memory if you loaded too many resources without quitting. This has been a non-issue for both this project and Medieval Brawl as computers today are more than capable of loading the amount of resources the application is using. An issue with speed would likely occur before the issue with memory, as searching through a changing map is slow.

A different issue is regarding code and maintenance of Kremengine as new resource types are required and old types are replaced. This could become difficult to handle and maintain over time and requires a significant amount of refactoring. A common resource handler for all resources is not atypical. All objects wishing for a resource would have to go through this handler. The handler would restrict how much memory any resource type can occupy, and how much total resource can be held in memory. Whenever a new

resource is requested which is not in the cache, the top handler would discard the least or last requested resource to make space for a new one. This is just one possible solution which is already miles ahead of what we currently have.

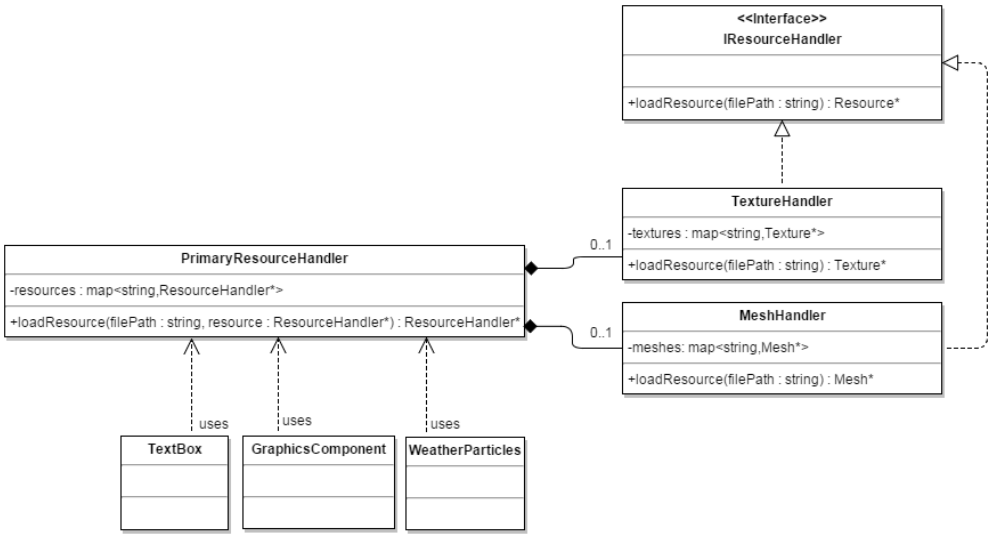


figure 9.9: A simplified version of what an improved resource handler could look like

Technical Design References

[1] Martin Thomas (2013). *OpenGL Instancing Demystified.* [online] Available at: http://www.gamedev.net/page/resources/_/technical/opengl/opengl-instancing-demystified-r3226 [Accessed 17 May 2016].

[2] Wikipedia. (2016). *Game engine.* [online] Available at: https://en.wikipedia.org/w/index.php?title=Game_engine&oldid=716577481#Rendering_engine [Accessed 17 May 2016].

[3] Gameprogrammingpatterns.com. (2014). *State · Design Patterns Revisited · Game Programming Patterns*. [online] Available at: http://gameprogrammingpatterns.com/state.html [Accessed 17 May 2016].

[4] Evan Wallace. (2010). *Finite State Machine Designer - by Evan Wallace*. [online] Available at: http://madebyevan.com/fsm/ [Accessed 17 May 2016].

Development Process

Development Tools

The tools we used for our development were as following:

For coding:

- Visual Studio 2015. This was our integrated development environment of choice because it is something we are all familiar with and it speeds up development with its intelligent code completion and performance measurements.
- SourceTree. We prefer the graphical user interface and ease of use compared to writing commands manually in command line.
- Bitbucket. We used this because we got a educational edition from school, so it was the only free option we had available.
- Notepad++. A light text editor with more features than normal Notepad. Used to take notes or read code without having to open the integrated development environment.
- Notepad. A really simple text editor that comes with Microsoft Windows.

for database:

- MYSQL workbench. Early on we used PHPmyadmin but we switched hosting service and the new one do not support PHPmyadmin easily because of security reasons, so we decided to switch since we where not really bothered and switching was easier than getting PHPmyadmin to work with the new host.

For art, sketching, graphs and discussing ideas:

- Adobe Photoshop. We really like the flexibility it gives us when it comes to in-game art and textures.
- Gimp. A free alternative to Photoshop because not everyone had Photoshop available.
- MS Paint. It has less features than other image editors, so it is faster and easier to use when sketching ideas.
- draw.io. For creating graphs.

For talking and working together:

- Microsoft Skype. A voice over internet protocol program with screen sharing, which is exactly what we need when we are not able to meet.
- Google Docs. A real time text editor makes it easier to see what others are doing so we do not end up doing the same thing.

Development Workflow

Every new sprint each group member would pick up tasks from the sprint backlog, which were placed in the in progress tab. The group members would the work on their own branches and do their work. That way, you would not be detrimental to the peoples work. When the work was fully implemented, it would be pushed onto the main branch.

If someone needed help, another group member would usually assist them as long as they were not busy. Many times there would be an issue that was very complicated, therefore two people would usually work together to get it solved.

When we started the project, we were not very proficient with working on branches, so there was usually a lot of mess. This got better after a month when people started getting accustomed to how to work with branches. We did not have an intermediary branch, therefore we usually pushed the changes we had on our branches to the main branch. When the tasks were done, the member would then go and place the issue as solved. Then they would pick up a new task to be done. There were exceptions where a large task had a part of it done, it would then be placed back into the backlog. Which would then be picked up later when it needed finishing.

Project Workflow

Scrum

We mostly followed the scrum workflow. We had a meeting every morning at 09:00 to discuss what we were working on as well as what we planned to work. We did not write logs for the daily scrum meetings. The daily scrum was not very strict, it was not mandatory due to the fact that we were four people in the team. Most mornings there would be two to four people joining the meetings. Most of the meetings were under 15 minutes, the shortest were 5 minutes.

Each friday we had a sprint meeting where we discussed the previous sprint and planned the next. This process usually took 1 hour. During the meeting we discussed what we needed to add to the backlog as well as what was finished. This was also an extended meeting where we could discuss various topics that popped up during the week. We also discussed improvements in our workflow during the meetings.

All of these meeting were done over skype.

Working Hours

During the development the team was working for at least 35 hours a week. There was no set schedule, the team worked at whatever hour they wished. There was also a weekly gathering set up at 09:00 on tuesdays so the team could join up and have face to face

interaction to help each other with the code and do code reviews, as well as come up with ideas on how to solve problems they encountered.

World Generation

Platform games have branched into many different subgenres, and the levels in the subgenres often have different subtypes og gimmicks.

Most of the very early platform games were in a two dimensional world, using only the x- and y-axis for width and heigth, respectively. You would traverse the x-axis by using the mapped keys on the input device for left and right movement. Traditional platform games let you jump with a jump key, which was the primary way of traversing the y-axis. Climbing was also introduced at a very early stage of platform history, as seen in the original Donkey Kong (1981 [13]). This was usually done by pressing the up-key while colliding or standing at the bottom of a climbable object.

Scrolling

Scrolling in computer games is the act of shifting the view from one part of the game world to a different view in the same level or world. Bubble Bobble(1986 [14]) solved this was to swap the whole screen with a neighboring screen once the player reached the edge. In Super Mario Bros(1985 [15]) the camera was locked to the player (Mario or Luigi), and would only scroll horizontally, meaning that the movement and vertical platforming was limited to the height of the screen. Later iterations of the Super Mario series and newer games improved on this and allowed for vertical scrolling in different forms [16].

The original The Legend of Zelda (1986 [17]) combined both seamless scrolling (like in Super Mario Bros.) and moving the screen once the player reached the edge. (One would assume that there were technical limitations for this. All technical difficulties to scrolling the screen have since been overcome.

Platforms

Classic platform games like the original Mega Man series, (1987 [20]) and many other (even to this day), use tile based platforming. This means that the level is divided in a grid with a set size.

figure 11.1: Small platform

Figure 11.1 shows a small section of a tile based platform game. Red blocks are solid

tiles and yellow blocks are enemy spawn points. Even though the world and spawn points were in this grid, the movement of all objects on the screen were fluid, meaning you did not move from grid to grid, but from position to position.

figure 11.2: without grid-locked movement

The other popular terrain generation method is to place each object in a precice location instead of in a grid. This requires more time, but will generally look better and give the level/game a feeling of higher quality.

Approaches

Creating a level generator that actually learns is something very few have tried before, there is really no information on the topic other than some Super Mario clones and maze generators. This lead to a huge challenge as we had to improvise and try things ourselves to find the perfect solution, which was not all that easy.

We came up with a few ideas on how to do this that might actually work and a bunch more than would never work. Our ideas will be discussed in more detail but here is a quick overview:

- Artificial Neural Network (ANN)
- Noise (Perlin, Fractal, Worms)
- Markov Decision Process (MDP)
- Deterministic Random Generation

ANN used neural networks to recognize patterns in a level and tried to reproduce them.

Noise approach used Perlin noise to generate a curve (height map) to place the world on, which would generate the floor of the level. There is a lot of different noise functions and it is hard to find the best one, but we ended up using Perlin.

MDP is not actually using Markovs algorithm but is heavily inspired by it. You pick a starting point (a node) and try to get to a certain point (goal node) and by giving every node you visit a score, you can determine if the path you followed was good or not. You will end up with a bunch of good and bad paths to pick between when generating a level.

Deterministic random generation is a random generation based on point to point generation in random directions with determined parameters.

Deterministic Random Generation

This approach is not based on any set algorithm. It uses random generation to generate levels. The basic premise is that you pick a point, which is usually the starting point of the level, then you choose a direction and give it a length. Then you continue doing that until you have your level. But there is a problem if you just leave it at that, there will be walls that are unscalable, holes that are too deep and just flat levels. To solve this, we need to have something to fill in the faults of the level. This is best explained with an example: Your jump height is 3 blocks. You have a level that is generated. It goes: Right 4, right 4, up 5, right 5, down 3, right 6. See figure 11.3.

Figure 11.3: Level example

From the figure, we can see that you can not possibly scale the purple wall with only the ability to jump 3 high. To solve this we need to give the generation a way to handle it. We could just put a block there or place stairs, but that would not make for an interesting level. So the way we do it is by going up one on the purple wall, skipping one block towards the left, the start generating to the left. Then, after a length is given to it, it will go that far, then go one more to the left and start generating blocks upward until it reaches the level of the purple wall. Then it skips one block the the right and generates blocks until it touches the purple wall. The length it is given is left 4. See figure 11.4.

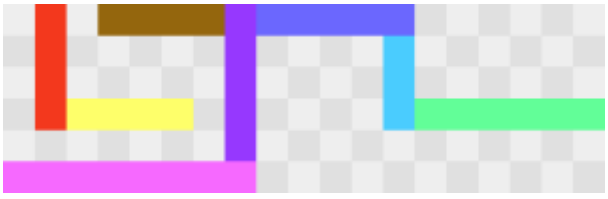


Figure 11.3: More advanced level

This makes for a lot more interesting way of generating levels. But the problem with this is that the learning potential of it is rather limited. You can teach it good lengths for platforms, how far it should go up and down and how far it should go in a straight line before making other obstacles.

Although this generation can be good and also fast, it requires huge amount of work to make it good.

Line by line Approach, our selected approach

This approach is similar to the tile-by-tile approach in the sense that it generates objects in a given direction based on the input to the AI, except this time whole lines are generated instead of just a tile at a time. Like the first approach this will also start generating from a location which will act as the starting position for the player, and generate lines either vertically or to the left(towards the goal). This approach also generates bottomless pits at random intervals affected by the AI.

The lines generated have a random component to them. Multiple straight lines can also be generated in a row if the AI decides so.

The green tiles are all one output from the AI. One can view them as: "Generate 5 horizontal tiles".

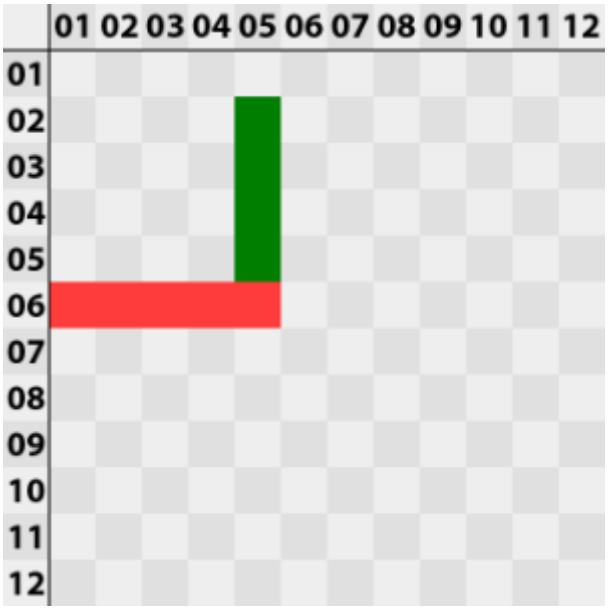


Figure 11.5: The red line shows the input, while the green line is the output. We can see that the AI decided to create a straight line upwards.

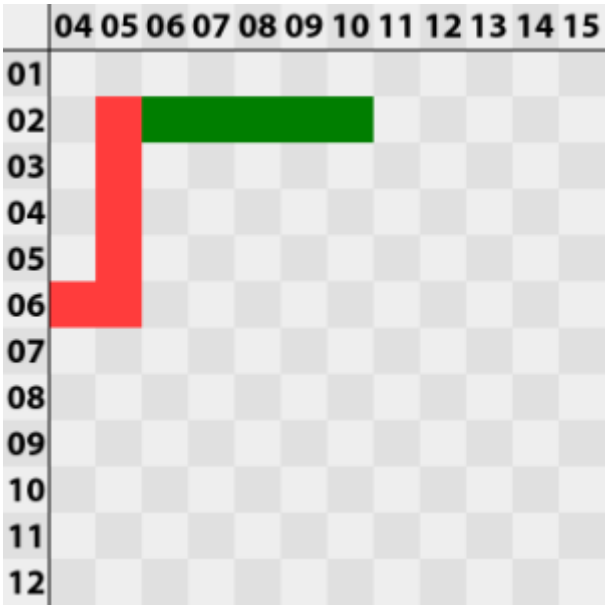


Figure 11.6: The next output.

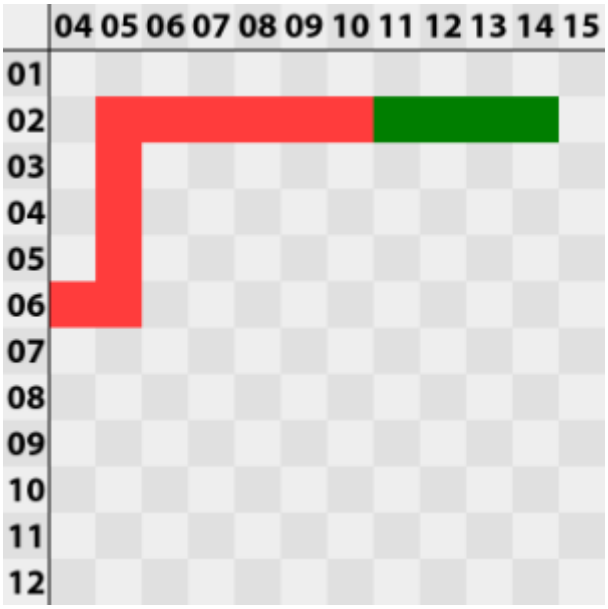


Figure 11.7: The AI generated two straight lines in a row.

The rest of the generation will continue in similar fashion as the tile-by-tile generation until the generation is completed. The time of completion depends on the implementation

Line by Line failed Implementation

We implemented this using a modified version of Markov Decision Process. This version can handle multiple "layers" of states, and returns multiple outputs.

Listing 11.1: Implementation of Line by line approach

```
enum direction
```

```
{
    GAP,
    UP,
    DOWN,
    RIGHT
}

//Output is represented by an int, which represents a direction, and
number of tiles
struct Output
{
    int direction;
    int lengthInTiles;
}

const int = MAX_LEVEL_LENGTH;
const int = MAX_PLATFORM_LENGTH; //technically not max, but won't
generate a new segment if the last segment reaches or surpasses this
number

2DVector tilePosition{0,0}; //The position where the next tile will be
generated

void generateTerrain()
{
    currentLength = 0;
    vector<Output> outputVector;
    while(currentLength > MAX_LEVEL_LENGTH)
    {
        outputVector.emplace_back(createSegment(outputVector));
    }
}

Output createSegment(vector<Output> outputVector)
{
    OutPut returnOutput = MDP.getOutput(outputVector);

    2DVector directionInVector;
    switch(returnOutput.direction)
    {
    case GAP:
        break;
    case UP:
        directionInVector.y+=1;
        break;
    case DOWN:
        directionInVector.y-=1;
        break;
    case RIGHT:
        directionInVector.x +=1;
        break;
    }
    if (returnOutput.direction != GAP)
    {
        for (auto i = 0; i < returnOutput.lengthInTiles; i++)
        {
            tilePosition += directionInVector;
```

```
            createTile(tilePosition);
        }
       int newOutput = MDP.getOutput(outputVector);
       //create new tile as long as a gap is not returned by the ANN
       if (newOutput.direction == GAP)
       {
           outputVector.emplace_back(newOutput);
       }
    }
    if (returnOutput.direction == GAP)
    {
          for (auto i = 0; i < returnOutput.lengthInTiles; i++)
          {
               tilePosition.x +=1;
          }
     }
}
```

We could not teach this algorithm how to learn to generate levels using the database, which is why this implementation was discarded.

Successful implementation of Line by Line Approach

This approach is built from the ground up to store and load from the remoteSQL Database, which is one of the primary reasons for its success. The level data is stored inside a struct within the object which handles the level loading.

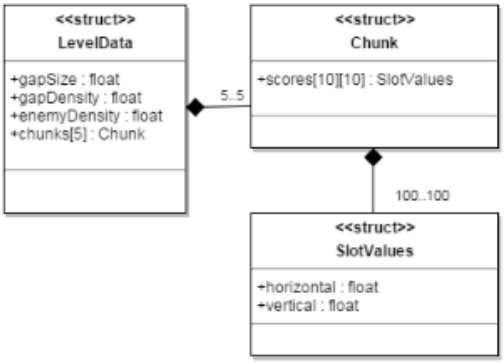


figure 11.8: Illustration of datastructure of LevelData

LevelData contains 5 chunks. Chunks are a way of separating a level into smaller pieces to more easily analyze what is going on behind the scenes. Each chunk is ten units wide and ten units high, or 100 units total. This means that the total level width is 50. We settled for shorter levels to get more precise feedback from users. The balance between precise feedback and too short level is a very thin line, but we seem to have found it. Some plans regarding feedback of specific chunks were also in place, but we decided against it for the sake of the user experience. One idea was to have the user pick out his favorite chunk and least favorite chunk, and apply stronger modifiers for those, but it turned out to be too impractical.

Each position in every chunk has a SlotValue object. SlotValue contains a horizontal and

vertical floating point value. These values represent how far in each direction tiles will be created once a tile reaches one of the positions, rounded down to first integer. The horizontal value will always finish generating its set of tiles before the vertical begins. The difference in height can never be greater than 9, and the highest point a tile can ever be generated at is 9, while to lowest is 0. This is to limit the generation and training of the AI. The AI is trained faster by limiting the state space. If the state space was near infinite, user feedback would have very little impact with this algorithm, as chances are that other users would never get to reach the tiles the first user gave feedback on. In addition to making the AI learn faster, it also makes it easier to read and track progress of the database when you only have a few thousand entries to keep track of rather than millions.

The algorithm which updates the database ensures that the values are never set to generate levels below 0 or above 9. The world generator is fully able to generate such levels, and this is intentional because it makes it easy for us to see if something has gone wrong somewhere during the training.

This table shows how the first 4 lines in a table are generated. Assume that generation starts where x=0 and y=5. We exclude gaps and enemies for now

Table 2

line number	1	2	3	4
position	x = 0,y = 5	x = 2,y = 3	x = 10, y = 0	x = 18, y = 3 (note: the x position here is moved to the second chunk)
Slotvalues	horizontal=2.1, vertical=-2	horizontal=8, vertical=-6.5.	horizontal=8, vertical=3.4	horizontal=4, vertical=0

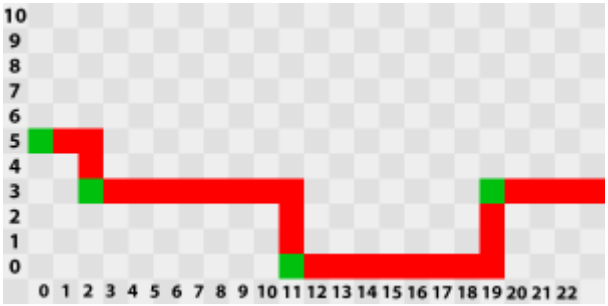


figure 11.9: figure illustrating table 2

In addition to just tiles lined up against each other, the generation also adds gaps and enemies. Gap size, gap density are loaded from the SQL database and used to decide how often and how wide the gaps should be. Gaps are limited in size to ensure that the player can leap over them. The gap size parameter is also more like a guideline than the

actual size. So there is still a random component to the gap size to give the levels a new feel each time. Gap density is also a parameter which acts more like a guideline than a static amount of tiles before the next gap. The chance of the first tile being a gap is 0%. After that the percentage increases. How much it increases depends on the gap density parameter. Some levels with have no gaps. The enemies density parameter behaves in a similar manner.

There is a difficulty balance in place to make sure that any given level has some obstacle, and on the other side it also makes sure no level is riddled with obstacles.

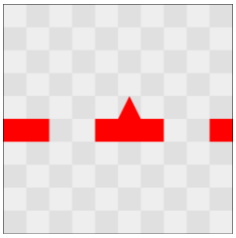


figure 11.10:a chunk from anywhere in the level. One can assume a fairly high gap density judging by the two gaps so close to eachother.

Storing and Loading from the database using Line by Line Approach

The database stores five different sets of data used for generating levels. These five sets are all stored in a single table, but in five separate columns. Those colums contain 1000 floating points each. There are 1000 floating points because there are 5 chunks in a level, each with 10 times 10 tiles, who each have two scores (a vertical and horizontal) as shown in Successful implementation of Line by Line Approach. 5 times 10 times 10 times two equals 1000. These values are stored in this manner for efficiency and simplicity. The values are loaded to a five temporary vectors, one for each column, and later stored properly in oldLevelData. Gap density, gap size and enemy density are loaded from the database in a similar manner. There are five sets of these values as well.

We postprocess on the local computer instead of storing the numbers in differently on the SQL server to make use of the additional processing power on the computer. There is still a noticeable network delay even when loading the numbers this way.

listing 11.2: loading data from the SQL database

```
std::vector<float> scores[5];
parentEngine->getDatabaseHandler()->retrieveData("SELECT Input1, Input2,
Input3, Input4, Input5 FROM LevelScores", columns, scores[0], scores[1],
scores[2], scores[3], scores[4]);
int chunkcounter = 0;
int i = 0;
for (auto eachScoreTable = 0; eachScoreTable < 5; eachScoreTable++)
{
 for (auto i = 0; i < 1000;)
 {
  for (auto j = 0; j < 5; j++)
  {
   for (auto y = 0; y < 10; y++)
   {
    for (auto x = 0; x < 10; x++)
    {
     oldLevelData[eachScoreTable].chunks[j].scores[y][x].horizontal =
scores[eachScoreTable][i++];
     oldLevelData[eachScoreTable].chunks[j].scores[y][x].vertical =
scores[eachScoreTable][i++];
    }
   }
  }
 }
}
```

Once all the data is loaded, a randomly selected column will be used to generate the level.

listing 11.3: level data is randomized

```
levelData = oldLevelData[rand() % 5];
```

The level is then loaded as described here.

The player will be prompted once he completes the level, where he is asked how much he liked or disliked the level. The program will then iterate through every selected starting tile figure 11.9, and find the current trend for the AI. If a player liked the level somewhat, then the AI will encourage the current trend somewhat by modifying the score towards the same direction the trend is going. If the player strongly disliked the level, the AI will discourage the current trend for the selected tiled by moving the score away from the trend.

Table 3:

Starting tile number		
position	x = 0,y = 5	x = 2,y = 3
selected Slotvalues	horizontal=2.1, vertical=-2	horizontal=8, vertical=-6.5.

other values1	horizontal=3.1, vertical=-3	horizontal=5, vertical= 3.2.
other values2	horizontal=4.1, vertical=-5	horizontal=3, vertical= 4.5.
other values3	horizontal=0.1, vertical=0	horizontal=2, vertical=-1.5.
other values4	horizontal=2.1, vertical=1	horizontal=6, vertical=-0.8.

We assume that the player liked the level, and we will take a quick look at the first two starting tiles to see what the current trend for these tiles is.

We start by finding all the average values, starting from the first horizontal to the last vertical we get: 2., -1.8, 4.8, -0.22

Now you compare the average with what the player liked. If the player liked a level that is above the average, the average will be increased. This is done by increasing one of the random values for the corresponding position in the database. These steps are repeated until all starting tiles are iterated through. The principle with random selecting is inspired by the random access pattern [19]. The idea is that will get a high variance in the levels due to the random nature of this pattern, while still being fair and moving towards a better generation AI over a long period and many iterations. One alternative is to always replace the oldest values, but we decided against that to save time on both implementation and the time it would take for the database to rearrange all tables by age after every feedback.

Noise Approach

When we got the idea of having a randomly generated world where we would use some sort of AI to place all the game objects, we had to consider of all the possibilities we could think of. One of these were using a noise function. The first thing we needed for the noise approach was a way to generate noise. We started researching all the different noise algorithm but ended up using Perlin noise, created by Ken Perlin, which was explained by Hugo Elias [1].

We started by implementing the version of Perlin noise Hugo Elias wrote, which is some sort of fractal noise and not actual Perlin noise [2][3], but it was simply too slow for our requirements. We explain why under Libnoise. This lead us back to searching for alternatives to implementing a proper noise function. We ended up using an external library to help us get the results we want, named Libnoise [4].

This library gave us more than we needed, as it got support for voronoi, checkerboard, spheres and more [5] but as we were struggling with performance, we thought a proper library would be much more optimized compared to what we could do, which is why we went that route. This turned out to be a good choice as we cut down loading times from 3 minutes down to 7 seconds at the same test. We used Visual Studios performance explorer to determine the differences in time between the library and our approach.

Libnoise only supports three dimensional Perlin noise, but it is easy to go down in dimensions compared to up so we just used one dimension for now, which is what we use for telling the world generator what height every object should generate at.

Perlin noise is really useful for us because we can easily get the next height without actually creating the object and then get its position, which cuts down memory requirements and load times as we know what happens next without having to actually generate a random number for the next object. Another benefit is that we can easily know what the next potential blocks would be and figuring out a pattern while generating instead of going over the world with multiple passes, which is another time saver when it comes to computer resources and memory usage as we only have to generate the world in one go and not having to go back and loop through all game objects to find the patterns.

Implementing Noise Approach

Using noise generators in games are quite common [6], especially for generating height maps for three dimensional worlds and we wanted to create a height map for a two dimensional world as that is what platformers are. We started implementing fractal noise from scratch and looked at how we wanted things to be in our world.

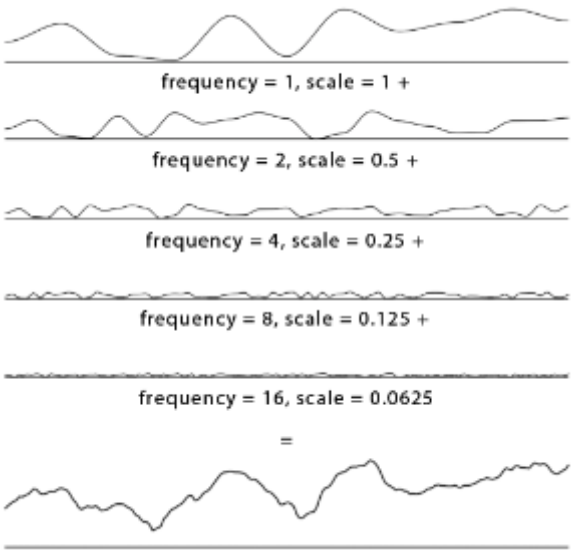


figure 11.11: example of output from fractal noise [7].

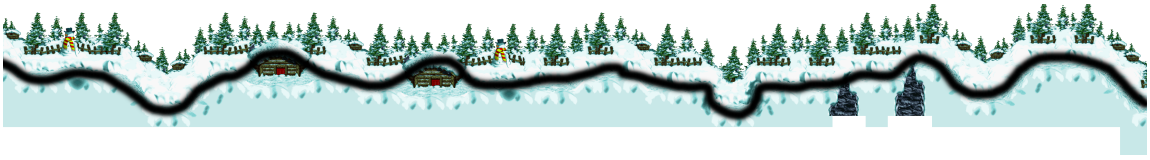


figure 11.12: example on how noise lines up with old platformer levels [8].

As we can see on figure 11.11, the various graphs fractal noise generates looks very

similar to levels in two dimensional platformer games, like the one displayed in figure 11.12, which is a picture from the game Donkey Kong Country from 1994. The black line represents the layout on top of an actual picture from the game. This is what this approach tries to replicate.

Libnoise got a lot of different variables we can tweak to theoretically find the perfect curve to match a professionally hand crafted level like the one on figure 11.12. We know this from looking at the example graph and the level and comparing them together, so we need to learn the computer how to produce similar levels. This is where the challenging part comes, as the various inputs to the noise function can produce levels you simply can not beat or levels that is completely flat, all of which we want to avoid. By tweaking the values by hand, we found something we were happy with so the learning algorithm know what to expect and where to start from.

Listing 11.4: The various inputs to perlin noise

```
noise::module::Perlin noise;
noise.SetSeed(static_cast<int>(time(NULL)));
noise.SetOctaveCount(6);
noise.SetFrequency(1.0);
noise.SetPersistence(1.0 / 2.0);
```

This are the values used to produce a somewhat interesting level, so the learning algorithm will have to change these values up and down to find the maximum and minimum values to generate interesting levels that is also doable. The problem with this is that we can not easily categorize levels from easy to hard, unless we generate every level possible and use User feedback to figure out what people thought were hard and easy.

To generate a basic level layout we use the following (simplified) block of code:

Listing 11.5: basic level layout

```
const unsigned int length = 100;
double height = 3.0;
for (unsigned int i = 0; i < length; i++)
{
glm::vec3 position{ i,
static_cast<int>((noise.GetValue(static_cast<double>(i / 4.0), 0.0, 0.0)
* height)), 0.0 };
this->actors.emplace_back(actorFactory->createActor(
dynamic_cast<ActorData*>(&PlatformData("res/actors/tileAtlases/castleFg_r
and.xml")),  //create actor
{ static_cast<int>(position.x), static_cast<int>((position.y * height),
position.z }, //set position
glm::vec2(0, 0)));
//texture to use
collisionDataVector.push_back(CollisionData{ {
static_cast<int>(layer2.x), static_cast<int>((layer2.y + (height)) *
1.5f) }, 1.f });
}
```

We place blocks between 0 and length on the x-axis and use the noise.GetValue function to figure out what height to place the blocks which is amplified by the height variable to give the level more variety in height. The static_cast<int> makes sure every block is placed exactly one unit in the height plane so it looks like figure 11.13, because that is the style of game we are trying to replicate.

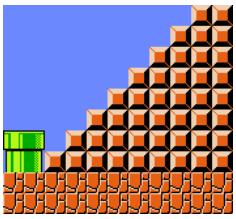


figure 11.13: image taken directly from the game Super Mario Bros. [9].

Learning With Noise

To create an algorithm that actually learns with this approach will be very hard because there is no pattern in how noise are generated through the different variables, so it would not know if the direction it was heading is good or not, which is why we have to make it look like it is learning. This can easily be achieved by creating a predefined set of rules to the world generation algorithm and use these in a smart way, which we discuss more in detail in the Obstacle Placement Using Noise section.

We looked at how Super Mario World, from 1990, was designed [10] and wanted to replicate the way the world is built up. They have designed the game to be gradually more challenging the further into the game you get so we need the learning algorithm to be kind to beginners and become gradually more challenging. It got a maximum of two

themes per level, with evenly spaced out safe spots where the player can take a break and get control over the situation. These themes can can be for example ground and rotating platforms. The entire level would then consist of only places to stand and platforms that rotate around a point to create elevators, the level designers then add enemies to places where they know the player will likely be after a certain challenge to kill them if they stop paying attention.

To implement this using algorithms is not as easy as you might think without User feedback as every player got a different skill level and a easy level for us might be hard for others. This approach would end up with an average difficult level all the time, so really good players would not be rewarded for their skills and beginners would probably not be able to beat the game. Something we could do to fix this would be to have a database for every player so every game would be different and tailored to your skill. The problem with this is that the learning algorithm needs thousands of inputs to generate something reasonable, which is simply too much for a single player to do.

Libnoise

The only reason to choose libnoise for this project is that it has been developed for years and is optimized for speed, which is exactly what games need. Our own approach worked, but it took up to three minutes to load a realistically sized level. This time was measured with Visual Studios own performance explorer. Having to wait three minutes in between each level is unacceptable. Another benefit with using a library that the project do not need but it was an interesting experiement, was all the various noise variants libnoise provided. This meant that we could test out variants we did not consider and potensially get better results, even though this is unlikely as the results do not produce curves that represents levels as we were looking for, but rather open caves or worms looking caves. Examples of this can be seen on figure 11.14.

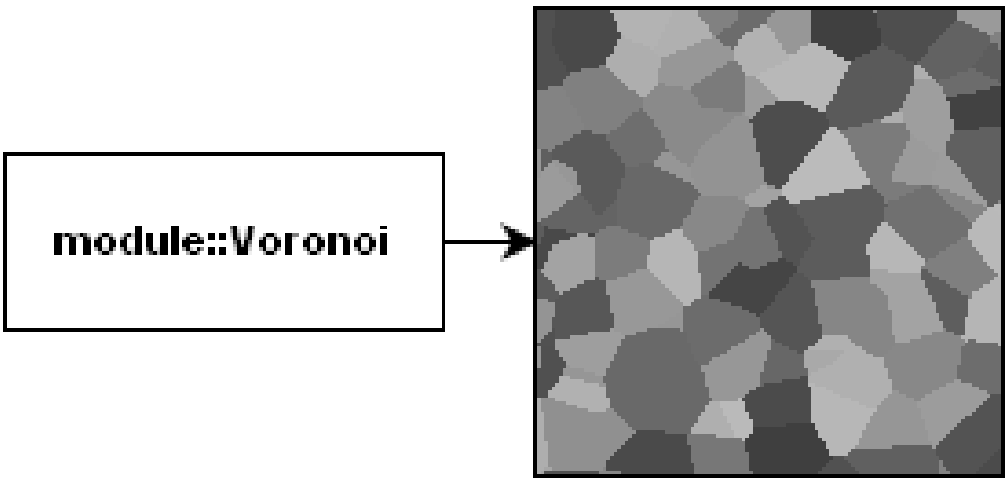


figure 11.14: voronoi pattern [18].

Obstacle Placement Using Noise

As we have discussed in Implementing Noise Approach, even hand crafted levels looks like noise, so the layout is not important if generated in a proper way. What matters are

where objects are placed. This is where the learning comes in, or what looks like learning. We created a few hand made algorithms to fill the level for us with all the various obstacles the project got support for. One of these are the spike filler algorithm.

On figure 11.15 we can see how the spikes are place in naturally generated pits. The yellow spikes are generated because the algorithm made a successful run, which means it found a suitable gap, placed two spikes and filled the surrounding with boxes, here visualized as red cubes. As figure 11.15 shows, the noise pattern creates a u-shape which matches the red boxes placed in the level.

figure 11.15: spike filler algorithm

Listing 11.6: Spike filler algorithm

```
int blocksToCheck = 5;
if (spike)
{
 glm::vec3 previousPosition = { i - 1,
static_cast<int>((noise.GetValue(static_cast<double>((i - 1) / 4.0), 0.0,
0.0) * height)), 0.0 };
 glm::vec3 nextPosition;
 if (previousPosition.y > position.y)
 {
  int spikesToPlace = 1;
  for (int h = 1; h < blocksToCheck; h++)
  {
   nextPosition = { i + h,
static_cast<int>((noise.GetValue(static_cast<double>((i + h) / 4.0), 0.0,
0.0) * height)), 0.0 };
   if (nextPosition.y == position.y) //equal heights
   {
    spikesToPlace++;
   }
   else //next block moved up or down
   {
    h = blocksToCheck;
   }
  }
  if (nextPosition.y > position.y) //next block moved up
  {
   for (int g = 0; g < spikesToPlace; g++) //fill the gap from original
position to last checked position
   {

this->actors.emplace_back(actorFactory->createActor(&ActorData("res/actor
s/movingSpike.xml"), { position.x + g, position.y - 0.5f, position.z }));
    for (int k = 1; k <= heightToFill + 1; k++) //loops through all
considered positions and fill them
    {

this->actors.emplace_back(actorFactory->createActor(dynamic_cast<ActorDat
a*>(&PlatformData("res/actors/tileAtlases/castleFg_rand.xml")), {
position.x + g, position.y - k, position.z }, glm::vec2(0, 0)));
     collisionDataVector.push_back(CollisionData{ {
static_cast<int>(position.x + g), static_cast<int>(position.y - k) }, 1.f
});
    }
   }
   i += (spikesToPlace-1);
  }
 }
}
```

This algorithm have a predefined parameter which can be changed depending on the difficulty of the level, for the explanations sake, the project uses the value 5. For every

block we consider placing into the world, we check if the previous block was above the next. We repeat this step as long as the previous block is still at the same level. When we have checked amount of blocks equal to blocksToCheck, we will check if the next block is above the current block we are currently at, if this is true, we will fill the gap with spikes. This will try to find u-shaped pits across the entire level and fill them with spikes as long as the gap is equal to or shorter than the blocksToCheck variable.

Using this kind of logic, we can define a lot of various obstacles and make the levels look smarter. The learning will only consider amount of objects to place, so an easy level might only have 3 pits with spikes even though the level got 6 suitable spots for them.

Tile by tile Approach

The tile-by-tile approach generates the terrain one tile at a time. The first tile will act as the starting position for the player, and from there on out an algorithm will decide the position of the next tile. One way to decide the next tile is to send the last few generated tiles as input to the agent, and get a position for a new tile as output. The output needs to have a random component to it, as to not generate the same levels for each user who uses this particular trained AI. Controlling the randomness is somewhat difficult because this approach only has four different out if you do not allow diagonal tile generation, and only three outputs in certain cases if you disallow generation backwards.

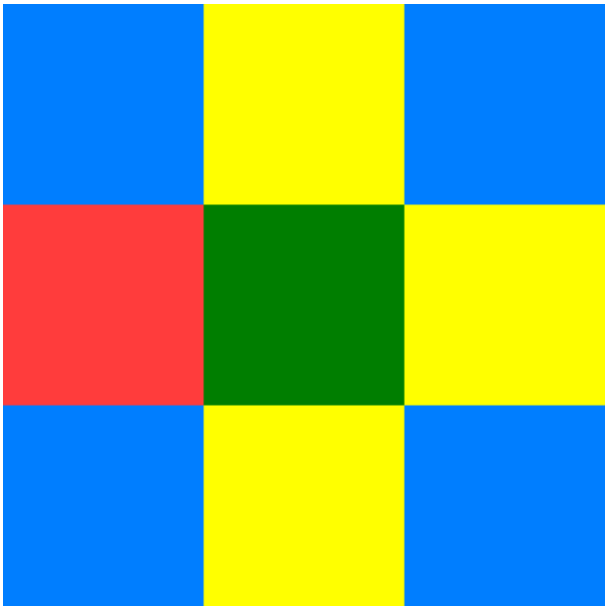


figure 11.16: outputs

The green square is the current tile. The red is the previous tile, so we can not generate the next tile in that position. The blue ones could be implemented by tweaking the algorithm, and the yellow ones are the primary available outputs. The fourth output is «ending the segment». Most platformers need «gaps» or «bottomless pits» .

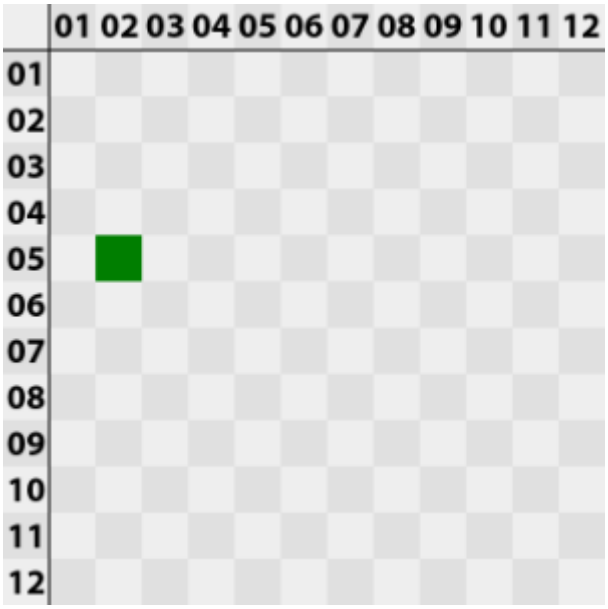


figure 11.17: The AI will take in «1 tile, position: 2,5» as input.

figure 11.18: The AI gave «Right» as output

The AI gave «Right» as output. We now send both tiles of the segment as input. It is important to send more than just the last output as input for the next tile, as the outputs are very limited, and would seem mostly random if you wanted varied generation.

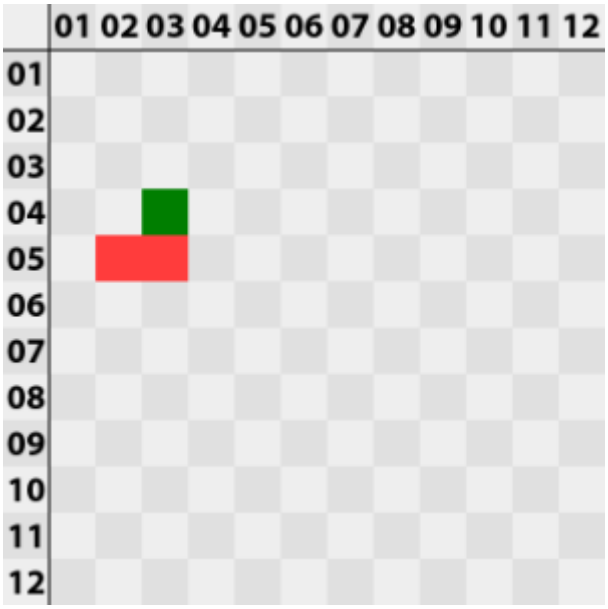


figure 11.19: This time the output was «Up»

This time the output was «Up». This time all three tiles are sent as input, and this cycle repeats itself until the AI returns «Gap» as output, at which point the generation will start over again at a new position somewhere to the right of the gap.

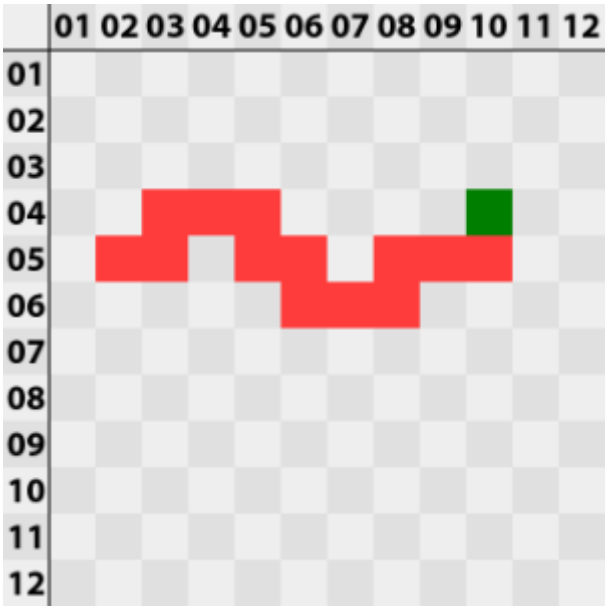


figure 11.20: A complete platform could look like this.

Pros:

The biggest advantage is the precise control over each tile you get. Every single tile is specifically tailored towards the rest of the inputs.

Cons:

Training the AI to generate realistic levels using this approach is difficult because platform levels consist of mostly straight lines. Creating straight lines with this approach would require the output to be mostly trained towards generating the same output as the last input. This generates situations where you could get an unusually long straight line. Otherwise the AI would be trained to generate something looking like figure 11.20.

Implementations:

We implemented this method by using Neural Networks(Artificial Neural Networks/ANN). An advantage and disadvantage of using ANN is that user feedback will affect every weigth in some way. This is advantageous when the user likes most patterns in a level, where all patterns will get a higher chance of occuring, and advantageous when he dislikes a level and all patterns are horrendous. Its disadvantageous when there is a perfect mix of bad and good patterns. All bad patterns will be graded as a "good" pattern if the player likes the level, and all good patterns will be labeled as "bad" if the level in general is bad. This could be avoided by asking very specific questions regarding the level, or even maybe ask the user to mark which parts of the level are bad, but this is impractical for both the user and our data analysis. We want to keep the feedback interface as user friendly as possible.

Listing 11.7: implementation of Tile by tile approach

```
2DVector currentPosition {0,0} //has two public variables x and y

...

enum direction
{
    GAP,
    UP,
    DOWN,
    RIGHT
}

//Output is represented by an int, which represents a direction
//The vector is empty on the first call. Neural networks require the same
amount of input each time. The neural network will fill empty inputs with
dummy input
void generateTerrain(vector<int> outputVector)
{
    int newOutput = ANN.getOutput(outputVector);
    //create new tile as long as a gap is not returned by the ANN
    switch(newOutput)
    {
    case GAP:
        break;
    case UP:
        currentPosition.y+=1;
        break;
    case DOWN:
        currentPosition.y-=1;
        break;
    case RIGHT:
        currentPosition.x +=1;
        break;
    }
    if (newOutput != GAP)
    {
        outputVector.emplace_back(newOutput);
        createTile(currentPosition);
        generateTerrain(outputVector);
    }
}
```

The above code has no randomness added to it, so it will always generate the same level until the network is trained slightly differently. ANN normally has no inapt way of adding randomness to its output. One could modify the behavior of the ANN to return a random output between some values, or add some form of randomness outside of the neural network. The weakness of the last method is that it will not be affected by the AI, so if the random implementation is sub-optimal, it has no way of improving itself.

World Collision

Here we cover a big stepping stone to smooth movement and collision. Tile based world generation comes with some problems which you evade with smooth landscapes.

Marching Squares Algorithm

The problem with tile collision was solved using the Marching Squares Algorithm [12]. The purpose for using the algorithm is to draw a long line along all the edges of a piece of landscape, instead of having Box2D create bodies from all actors.

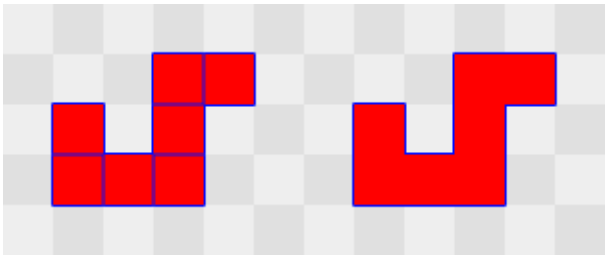


figure 11.21: platform before and after applying algorithm

The algorithm takes in a two dimensional array consisting of integers as input, where the integers are either 1 or 0. These numbers represent a complete platform in the level in addition to the empty spaces around this platform. Ones represent a collision block, and zeroes represent empty space. In practice this means one has to push a 1 to a vector or list every time a new tile which is meant to be used to the world is generated.

Listing 11.8: Adding a new solid tile to the vector for the algorithm

```
collisionDataVector.push_back(CollisionData{ { xPositionOfBlock,
yPositionOfBlock}, 1.f });
```

After the whole level is parsed this way, the PhysicsHandler class fills inn the empty spaces with zeroes. Normally one would have to use other values than only 0 and 1, but our world consists of only sharp edges and no curves or slopes, and the curves are either 90 degrees or 270 degrees, which these values portray perfectly when following the algorithm. Only using 0 and 1 allows us to skip a step of the algorithm, speeding up the process slightly.

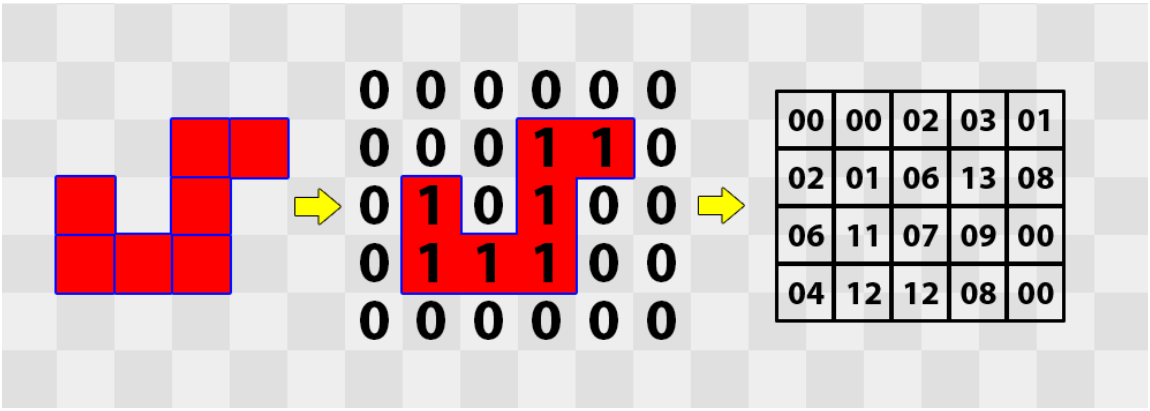


figure 11.22: what the user sees; what is sent to the physics handler; final numbers for look-up table

The details between the two last pictures is very technical and can be studied more deeply on the Marching Squares wikipedia page[12], but the general gist of it is that you take four numbers: the top left number, the one next to it, the one below that one, and finally the one to the right of the current number. You should end up on the number below where you started. This gives us 0000 in binary. Convert this number to decimal and you get the number in the top left square in the last picture. To get the next square you start from the second number on the top row and traverse the same path, giving you 0000 again, or 0 in the decimal. the third square is composed of 0010, which is 02 in decimal.

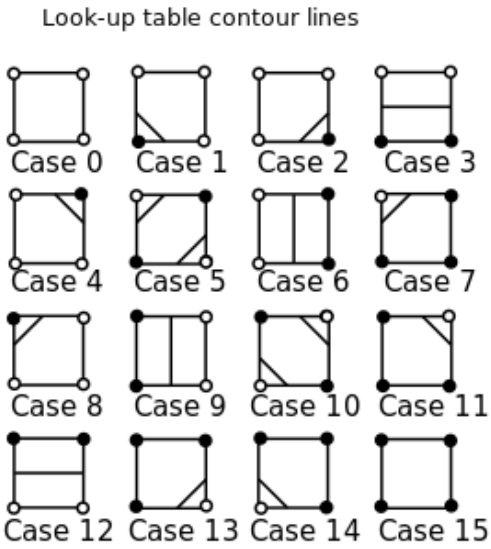


figure 11.23: cases

By replacing the numbers of the squares with the lines in the look-up table you will get something that looks like the platform above, except the all corners are cut. In The lines are drawn inside a switch statement by the physics handler, where all cases of cut corners are replaced by a 90 degree turn in the direction the corner is cut, giving us the correct line. Drawing starts from the first non-zero square starting from the top left and going

right. The switch marks each traversed node as visited, and calls itself recursively with the position of the next node. Marching Squares will stop generating lines as soon as a node is visited twice, which means the algorithm has created a complete line around a platform.

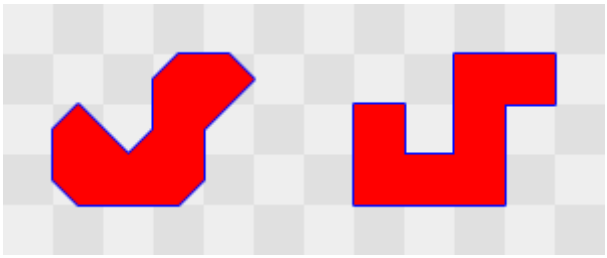


figure 11.24: comparison

The first shape above illustrates how the object would look like if the lines were drawn exactly from the algorithm. The second shape is just for comparison.

Listing 11.9: Example from Marching Squares algorithm

```
//case 2 and 13 are equivalent for our project as we only care about the
lines and not whether or not we are on the inside or outside of what's
physical
//only one of the if checks can be true
case 2:
case 13:
 if (direction.x < 0) //this check is true if we the previous tile was to
the right of this one, making sure we draw to the right
 {
  returnVector.back().x -= 0.5;   //we move towards the left
  newVector = returnVector.back(); //we start the next line from our
current position after moving half a square to the left
  newVector.y -= 0.5;     //we now move downwards
  returnVector.push_back(newVector);
  direction = { 0,-1 };     //we are going downwards. this is information
required by the next square
  if (!(newVector == originPoint))
  {
   i += 1;
  }
 }
 if (direction.y > 0) //this check is true if we the previous tile was
below this one, making sure we draw upwards
 {
  returnVector.back().y += 0.5;
  newVector = returnVector.back();
  newVector.x += 0.5;
  returnVector.push_back(newVector);
  direction = { 1,0 };
  if (!(newVector == originPoint))
  {
   j += 1;
  }
 }
 break;
```

The problem with tile collision

We use Box2D as our physics library. Box2D is a two dimensional physics library, so when a collision between two object occurs, the library has to decide how it wants to push the objects apart in the vertical and horizontal dimensions. In World Generation we made clear that we are going for a tile based world and how a tile based world is constructed. In Kremengine the world is made by placing square Actors next to each other with no space between them.

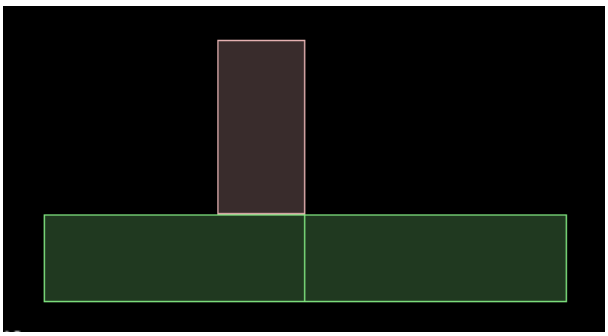


figure 11.25: two ground tiles from a separate game together with the player on top for illustration purposes [11]

When a collision between a player and the ground occurs, the ground will stay put because it has a static bodytype. Static bodies can not be moved physically under any circumstance, and can only change position by manually setting a new position. The player will constantly be pushed up by Box2D as long as it only collides with one side of a single object, but as soon as it collides with the top of one object and the side of another, there is a chance that the box will get stuck in a loop, depending on how deep the object is stuck in the sides of the second object.

figure 11.26: the block is being pushed up to the left, and left on the right [11]

The first object pushes the object up, which is perfectly fine, but the second object pushes the object in the opposite direction of where it is moving. This will make it look like the object is standing still when it is simultaneously moved up.

One cheap solution to this problem, which we used in Medieval Brawl, is to make the players and enemies circular. This does not solve the problem and causes some other minor issues regarding standing on ledges which one might want to avoid. A similar solution is to clip the ledges of a polygon. This is better than the circular solution but still has some of the same issues regarding ledges and you can still lose momentum when stuck between tiles since you are pushed lightly away from the direction you are moving.

figure 11.27: clipped edges of a square body makes movement better.[11]

Our choice

When we decided on AI for world generation as the topic for our thesis, we already had some ideas in mind on how we wanted to do it. We wanted the generated levels to be fairly generic to put the levels in as many realistic settings as possible. Many of the modern platform games have a wide variety of level «types». Some are mostly vertical, some are horizontal, some are climbing focused, some revolve around dodging obstacles on a vehicle. We wanted to focus on a single level type, and make that one as good as possible.

The type of level we decided on have both horizontal levels and horizontal scrolling with tile based platforming. These types of levels have enough complexity to make interesting levels, while still being simple enough to improve through a learning algorithm.

We are not concerned with the time it takes to generate a level as long as it is withing a reasonable time, only the level quality.

World Generation References

[1] Freespace.virgin.net. (no date). *Perlin Noise*. [online] Available at: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm [Accessed 14 May 2016].

[2] Stefan Gustavson. (2005). *Simplex noise demystified.* [online] Available at: http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf [Accessed 14 May 2016].

[3] Gamedev.stackexchange.com. (2011). *Understanding Perlin Noise.* [online] Gamedev.stackexchange.com. Available at: http://gamedev.stackexchange.com/questions/18330/understanding-perlin-noise/18331#18331 [Accessed 14 May 2016].

[4] Jason Bevins. (2007). *libnoise: a portable, open-source, coherent noise-generating library for C++*. [online] Available at: http://libnoise.sourceforge.net/ [Accessed 14 May 2016].

[5] Jason Bevins. (2007). *libnoise: Documentation.* [online] Available at: http://libnoise.sourceforge.net/docs/group__generatormodules.html [Accessed 14 May 2016].

[6] Wikipedia. (2016). *Procedural generation.* [online] Available at: https://en.wikipedia.org/w/index.php?title=Procedural_generation&oldid=716861707#Video_games [Accessed 17 May 2016].

[7] Scratchapixel.com. (2012). *Pattern Examples.* [online] Available at: http://www.scratchapixel.com/old/lessons/3d-advanced-lessons/noise-part-1/pattern-examples/ [Accessed 17 May 2016].

[8] Dkc-atlas.com. (2016). *DKC Atlas.* [online] Available at: http://www.dkc-atlas.com/ [Accessed 17 May 2016].

[9] Wikipedia. (2016). *Super Mario Bros.* [online] Available at: https://en.wikipedia.org/w/index.php?title=Super_Mario_Bros.&oldid=720413576 [Accessed 17 May 2016].

[10] Thegamedesignforum.com. (2016). *Reverse Design: Super Mario World.* [online] Available at: http://thegamedesignforum.com/features/RD_SMW_1.html [Accessed 17 May 2016].

[11] Iforce2d.net. (2013). *Ghost vertices - Box2D tutorials - iforce2d*. [online] Available at: http://www.iforce2d.net/b2dtut/ghost-vertices [Accessed 17 May 2016].

[12] Wikipedia. (2015). *Marching squares*. [online] Available at: https://en.wikipedia.org/wiki/Marching_squares [Accessed 17 May 2016].

[13] Wikipedia. (2016). *Donkey Kong (video game)*. [online] Available at: https://en.wikipedia.org/wiki/Donkey_Kong_(video_game) [Accessed 17 May 2016].

[14] Wikipedia. (2016). *Bubble Bobble*. [online] Available at: https://en.wikipedia.org/wiki/Bubble_Bobble [Accessed 17 May 2016].

[15] Wikipedia.(2016). *Super Mario Bros.*. [online] Available at: https://en.wikipedia.org/wiki/Super_Mario_Bros. [Accessed 17 May 2016].

[16] Shaun Inman. (2016). [online] Available at: http://blog.mimeoverse.com/post/577060703/following-yesterdays-analysis-of-super-mario [Accessed 17 May 2016].

[17] Wikipedia. (2016). *The Legend of Zelda*. [online] Available at: https://en.wikipedia.org/wiki/The_Legend_of_Zelda [Accessed 17 May 2016].

[18] Jason Bevins (2005). *libnoise: Documentation.* [online] Available at: http://libnoise.sourceforge.net/docs/classnoise_1_1module_1_1Voronoi.html [Accessed 18 May 2016].

[19] Wikipedia. (2016). *Random access.* [online] Available at: https://en.wikipedia.org/wiki/Random_access [Accessed 18 May 2016].

[20] Wikipedia. (2016). *Mega Man (video game).* [online] Available at: https://en.wikipedia.org/w/index.php?title=Mega_Man_(video_game)&oldid=713617125 [Accessed 18 May 2016].

User feedback and testing

Since we are making an algorithm that improves over time by people playing trough levels we are heavily dependent on testing the levels and algorithm and also people giving feedback to to improve the algorithm

User feedback

When creating an algorithm to generate 2D-platformer levels by using a learning AI there are different approaches you can use when it comes to the learning part of the AI. Since our goal is to make the algorithm learn and get better by having a player play trough the generated levels, we need to have some form of user feedback that the algorithm can use to learn how to make better levels. We looked into two types of user feedback for this task. We call them active and passive user feedback.

Passive user feedback

The passive user feedback is called passive because it is not actually collecting feedback directly from the player himself. Instead it is collecting data throughout the level when a player is playing trough a generated level. The program will collect different data and variables while the player is playing and store this to further improve the algorithm by using the data collected. The learning part of this is to use the data to generate levels based on how the player is playing. This can also be used in real time while player is playing to alter the generation ahead of the player. For example if the player is cruising trough the level and is having absolutely no problem defeating the challenges that the algorithm is giving, the algorithm can then see this with the data it has been collecting and generate a more difficult part ahead to make the level feel challenging. In the same way the algorithm can also make the levels feel easier if the player is having a really hard time. With this method we can make the player feel like the level has a perfect difficulty and when they improve the levels also improve their difficulty. With this approach the player might get the perfect difficulty curve and levels might be more enjoyable when the difficulty changes to the players skill.

Active user feedback

The active user feedback is called active because it is collecting feedback directly from the opinions of the player. We want to ask the player questions before and after a playthrough to gather the information we need for our algorithm. Before the playthrough we want to find out how the player feel about their skill level and the experience they have playing 2D-platformer games. These answers are stored and used for later by helping us see how experienced and skilled the player is and taking this into account when the player is answering the questions that comes after the playthrough. For example the player might find a level extremely difficult to complete but only because

the player has little or no experience playing 2D-platformer games. If this happens and we tell the algorithm that the level generated was extremely difficult, we end up "tricking" the algorithm into thinking that levels that experienced players actually find really easy is extremely hard and this will make the algorithm worse. The optimal algorithm will generate levels with different difficulty based on the experience of the player.

After the player has played through a generated level the user is again asked a few questions. The questions revolve around asking the player about the difficulty of the level, how fun the level was, rating the level and/or the overall design of the level. The answers from the player is collected and based on what the player has answered the algorithm gets updated. The algorithm takes data from the level that was played through and alter the algorithm based on if the level feedback was positive or negative. The algorithm can be saved in different ways: online, offline or per session. The per session approach will make the algorithm generate better levels for a player for just one session before it is reset, the offline approach will only develop the algorithm for one particular computer while the online approach will evolve the algorithm for everyone. The online approach also lets us see how the algorithm evolves over time and we can then actually find out if the algorithm is actually improving or not, since we can take a look at the algorithm and see how it has been evolving.

Our approach

For our algorithm we have decided to go with the active user feedback approach. The reason we did this is because we are interested in seeing how our generated levels would evolve when players had a active saying in which levels where good or not instead of having the program figure it out from data. We have decided to only as the player two questions after each level. The reason for this is that we want to collect data from every level but answering questions can be a boring task for a player and they might end up being annoyed or bored and leave us less feedback to work with. We also use questions that are easy and quick to answer to also make it more likely that players would put the effort inn to actually answer. We do not ask the player about the experience of the player and so on before they play because we focused on the algorithm creating just one level and not having more difficulties to choose from.

The questions we decided to ask the players is: "Was the level fun?" and "Please rate the level. (from 1 to 3 stars).

Was the level fun?

This question is what we consider to be the easiest and fastest way to figure out if the player found the generated level to be good. We think that if the player found the level to be fun the level must also have been at least somewhat decent, and since our game play is pretty limited we do not think that the game play alone can carry a boring level. The

question has some problems that we have to take into consideration though. The question only asks for the entire level, but the player might have enjoyed some parts of the levels and hated the rest, so when answering the question the player does not quite know what to answer. An solution to this that we considered was asking the player about different parts of the level, but this felt tedious for the player to answer and asking questions like: "Was the beginning fun?" felt strange and questions like: "What count as the beginning?" would come up. To make it so we only needed the one question we have made the levels short enough so that the player can feel like the whole level was either fun or not and that the player remembers the entire level.

Please rate the level

This question is also very easy and quick to answer after first answering the one above. The player can rate the level he just played and give it anywhere between one and three stars. The reason we went for a three star rating system is because we wanted three ratings that kinda meant bad, okay or good. If you have a rating system consisting of a lot of options people tend to pick answers on the end of the scale or the one in the very middle. This makes the other options kind of useless since they are not used enough and we also felt like we did not need them to make the algorithm better. The rating is then used to make more drastic changes to the algorithm based on the rating and the first question.

When the player answer the question the new data gets sent to the database. In the database we have the five last entries saved and the newest one replaces the oldest one. In the beginning we considered saving every entry but we figured that it would be too much data to store and we really did not need to have all the data but rather just the last few entries since the algorithm is evolving constantly. It would however be an interesting thing to have so you could look back at it later and see how the algorithm was at a certain point in time. We also considered saving every entry done and info about the session and player submitting the data but we did not want to deal with ethical issues like privacy and having to make an terms of service agreement to ask if we could use data collected from the player and we also felt like it did not add to much knowing these things.

Testing

Getting data from testing is a big part of our algorithm. We wanted to test the algorithm both internally in the team and publicly with other people. In the beginning the aim was to get people we know and also strangers to test the algorithm. The plan was to release the program online and have people test it there since the data gets saved to our online database. This became hard to do when our Installer did not want to work and we had to scratch the online testing. This did not limit our testing too much because we had already

talked about the risk of letting strangers testing and now that risk was gone. The risk with online testing is mostly about players answering incorrectly on our questions and "tricking" the algorithm to think bad levels are fun and vice versa. Other things to take into consideration is security regarding the database, for example SQL injection or some experienced developers would maybe be able to get the database info out of our code and tinkle with the data or ruin the database entirely. We are using prepared SQL statements so SQL injections should not be a problem. People tricking the algorithm should not be a problem since we think most people would have been honest and therefor the numbers of people being honest would out shine people trying to ruin the algorithm, but the security risk might have been a problem. Without the installer working we where still able to make some friends and family test our algorithm.

Internal testing

The internal testing was done throughout the entire bachelor period. We tested the algorithm a lot every time changes where made to see if it was producing levels like we wanted it too. After every change we also reset all the data in the database so that the algorithm would start from scratch in case something unexpected would happen after a change that we did not think about if the change was big enough. After a while when the algorithm worked mostly like we wanted it we did a new reset and started testing it with intention of seeing if the level would eventually become a good and fun one. After a reset the newly generated levels would be mostly a flat line like his:

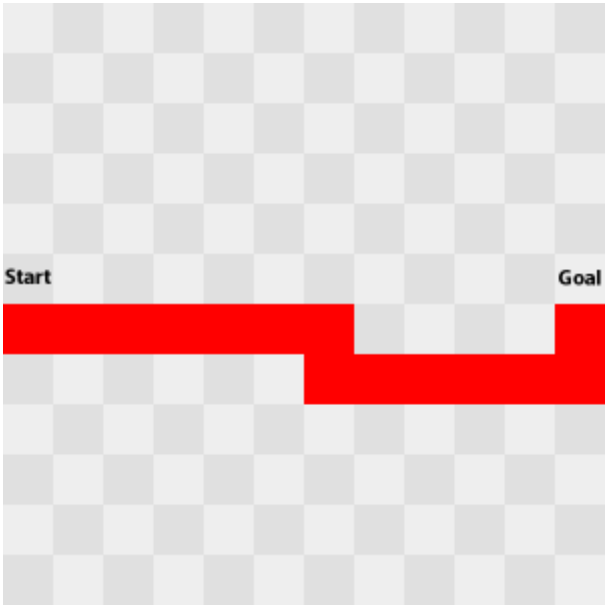


figure 12.1: flat level.

Then after testing the levels for some time it would start to look closer to how a fun and engaging 2D-platformer level looks like when it is professionally made for a game.

Those levels looked mostly like this:

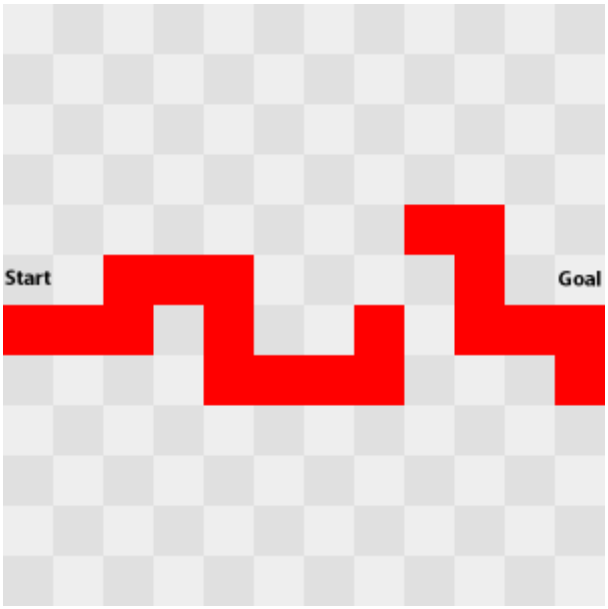


figure 12.2: level with variation.

After we reach the more playable and fun levels we decided to let other people have a go at the algorithm to see if it would improve further and create better levels or see other consequences of having people not involved in the project test the game.

Public testing

For public testing we chose to only invite friends and family because we know what type of players they are so we could get a better range of skill, from beginner to professional, this also meant that we could monitor the results to avoid people messing with the algorithm as we had limited data to start from. We settled on gathering data from ten people excluding the team. The algorithm requires more tests than ten people could provide, but even with the limited amount of people testing the algorithm, we still managed to get some interesting results. Compared to our Internal testing, we saw similar results in the layout of the level, but the algorithm evolved at a faster rate because of the larger pool of data we could gather. The team observed how different people played the game and how they answered the survey afterwards. Looking at the results we found out that we would need a lot more than ten people for future testing and to see if the algorithm evolved into what we expect, but we got a good first impression on how it worked in reality and we are happy with the results so far.

Deployment

Installer

We use Visual Studio installer[1] for our installer. To get it to work, you only need to install it and use Visual Studio to compile an installer. Everything is pretty much just drag and drop. You choose a folder and drag all the different resources needed into the project. You can also include all the merge modules[2] that you need to run the project.

Problems

There we a lot of problems concerning the installer. It will run perfectly fine on computers that are used by gamers and developers. This is due to the fact that they have most of the DLLs already installed on their computers. This was attempted to be solved using cygwin[3] using the "ldd [programname].exe" command and Dependency Walker[4]. Even using those two programs, we were unable to solve the issues of missing DLLs. But there was another problem there as well. Even after adding the DLLs that were needed, we still encountered problems. This pointed towards there being 64-bit DLLs being used when it was supposed to use 32-bit DLLs, But all the compiled DLLs were 32-bit. The problem might lie in one of the libraries that require a DLL that does not show up on programs that allow you to see the DLLs needed. It is highly probable that mysqlcppconn.dll is causing the problems due to the fact that it does not have proper support for compiling it for Visual Studio 2015.

Deployment References

[1] Microsoft.com (2015). *Microsoft Visual Studio 2015 Installer Projects.* [online] Available at: https://visualstudiogallery.msdn.microsoft.com/f1cc3f3e-c300-40a7-8797-c509fb8933b9 [Accessed 14 May 2016].

[2] Microsoft.com (2013). *About Merge Modules*. [online] Available at: https://msdn.microsoft.com/library/aa367434 [Accessed 14 May 2016].

[3] cygwin.com (no date). *Cygwin.* [online] Available at: https://www.cygwin.com/ [Accessed 14 May 2016].

[4] dependencywalker.com (no date). *Dependency Walker*. [online] Available at: http://www.dependencywalker.com/ [Accessed 14 May 2016].

Discussion

Group Work and Workload

We decided at the start of the bachelor period that no one in the team should be the group leader for the entire period. We ended up having no group leader at all. What we did instead was that we where going to vote whenever there was a disagreement and a decisions was needed to be made. In the event of a tie there should always be one person with a double vote so that we could push the disagreement in a direction and a final decision could be made. The person with the double vote would rotate every time the double vote was used. The first person to receive the double vote was Jonas and the double vote was never needed because most our disagreement ended with either everyone finally agreeing or a three to one vote. Early on in the bachelor period we had the requirement that each member of the team had to work a minimum of 40 hours, we where thinking that a full time job is approximately 40 hours a week and that we could do the same for our bachelor. We did not factor in people having other courses or that when you have a full time job you do not actually work the full eight hours everyday and you can have a few breaks in there but when we where tracking our time we only tracked actually working time. Therefor we changed the requirement down to 35 hours. This lead to the timed track actually being more inline with work actually being done and the work people did was a bit better since they did not need to push them self. Still even with 35 hours minimum people sometimes worked a bit more than that.

During the development each individual in the team got more and more locked into certain roles due to working on certain part of the engine a lot. This lead to it being easier to ask a person to do something in a specific part of the engine instead of doing it themselves, because it would either be faster or other people would not know how and would have to first learn how that part worked before they could implement what they wanted. This lead to some small problems with time being used waiting. Since we did not have a set time frame of when you had to do work on the bachelor project, a person might need something at nine in the morning, but the person that knew how to get that done might not be awake before later in the day. This was however not a huge problem since there was a lot to work on and you could mostly just do something else while waiting for the other part.

Further Development

Even though everyone on the team has learned a lot and find the concept really interesting, we as a team have decided that we do not want to work on the algorithm anymore after the bachelor project has concluded. The reason for this is that we are planning on creating an indie development game company and with that, our main focus will be creating games. We do not think that this algorithm will net us any money. Early on we where looking into if you could create an algorithm with a program that other

companies could use to generate levels for their 2D-games and then maybe sell this program. We quickly found out that the program would either be to limited for this or it would take us a lot of years to make it somewhat decent and we just do not think that this is worth our time as a company.

Cooperative Play

We were interested in adding a cooperative playmode from the beginning, and we were especially invested in figuring out how multiplayer effected the gameplay experience on the levels. Seeing how the feedback differs from single-player would be quite the learning experience. Some of the issues we foresaw were that co-op would make it harder to track AI improvements over time. SDL_Net, a network library we used during Medieval Brawl, was also replaced with RakNet early on, and we were excited to test it out. Early prototyping showed that RakNet was more user friendly than SDL_Net, and levels are actually playable co-op right now with RakNet. The database hurdle was just way too huge for it to be worth investing time in. The psycological effect it has on a person when he plays with someone else rather than alone is still something we would like to experiment with in the future.

Conclusion

When we started working we really wanted to create a perfect algorithm for generating 2D platformer levels. The more we worked on this the more we realized that this is not a feasible task, at least not without spending a big portion of your life dedicated to this task alone. What we ended up with is is an algorithm that is able to learn and generate 2D platformer levels that resembles hand crafted levels used in similar games. The algorithm is specific for 2D platformers of this type and is not easily modified to support other types of platformers.

We also made significant improvements to our own engine which we would like to use for future projects either for prototyping or creating simple games. We know that our engine is good for creating 2D platformers and until we get a lot of prefabricated assets for other game engines we think that our engine is the best for us for creating these types of games. Developing a bigger project where you have to use tools at a professional level, time tracking and management models as a group has made us learn a lot about how the industry work. This is a very good experience to have when we are going to start our own company after the bachelor period has concluded.

What we ended up with is a bachelor thesis that included something from every course included in the entire game programming degree.

Appendices

A. Project Plan

Kremen

Group Members:

Christer P. Somby

Henning E. Luick

Kristoffer Eidså

Jonas D. Reitan

PROJECT PLAN

1.1 Goal

1.1.1 Background

We made the start of a game engine during our game programming course during our 5th semester, and decided to continue working on that during our planning phase. We will be using our engine during this bachelor to make the level generation tool. We are all very interested in Artificial Intelligence, so we wanted to delve deeper into that topic during the thesis. We have some experience within the topic and wish to improve on that and hopefully discover something interesting about level design in the process.

Our team consists of Kristoffer Eidså, Henning E. Luick, Jonas Reitan and Christer P. Somby and we call ourselves Kremen.

1.1.2 Project goal

By the end of the project we will have learned more of the following subjects:

- Intelligent level generation
- Learning AI using established and new algorithms
- Gathering of user statistics for better learning of AI's.
- Medium scale networking and data collection

We wish to accomplish the following product goals by the end of the bachelor period:

- We will have implemented highly advanced AI
    - This will be accomplished by using artificial neural network or genetic algorithms or a combination of both.
- Have completely implemented networking using ZeroMQ

1.2 Scope

1.2.1 Project description

We are going to write an AI that can tell whether or not a platform level is good or not. The AI will do a test through the level to see if it's beatable, then it will present the level to the user. The user then plays the map, when the user is finished, he will rate the map. This will be extra data the AI receives to then be better at choosing levels that are good.

This AI will also use the users inputs or movements to teach itself how a player would solve a map and use this in the way it beats levels. (There are some risks of the levels not

being complicated enough after some time. So this needs to be considered)

The levels the AI will test are going to be randomly generated by some sort of AI that learns how to make platform levels. It might work together with the solving AI to learn how to make better maps.

1.2.2 Delimitation

We are not going to make an advanced 2d platformer but rather a tool used for level generation in 2d platform games. The generated levels will contain many common features in platform games, but we will not be able to include all features. New features can be added in the future, but the AI has to be recalibrated (by user tests) to include those in new generated levels.

The tool will be specifically tailored for our engine, but the algorithms we end up using should still be applicable to other tools.

If the make system that is planned for the project turns out to be too much work compared to what is initially expected, it will be dropped.

1.3 Projectorganization

1.3.1 Roles and Responsibilities

All team members will work on all parts of the project, and we have a shared responsibility to make sure the project turns out as good as possible.

Henning is responsible for the networking.
Christer is responsible for the make system.

Kristoffer and Jonas are responsible for the AI.

These responsibilities are by no means final and may change during the course of the bachelor. Any person that wishes to work in a specific field is allowed to do this.

1.4 Software Development

We have decided to choose RUP and Scrum as our software development methodologies.

1.4.1 Description of software model

We are going to use sprints from the scrum model. Every friday at 16:00 we are going to have a sprint meeting to discuss what we have accomplished that week and to find out what each person should work on for the coming week.

From the RUP model we are using the way they divide everything up in phases. We start off with planning and gradually switch over to development. Our project depend on user feedback so we want to do testing parallel to development.

1.5 Risks

1.5.1 Identify and analyze project risks

Time - We may use to much time on creating the program. If we use to much time on this we may have too little time collecting the data we need.

Too complex - We might get stuck on a particular part of the program that is too complex for our knowledge.

Group problems - Any problems with the group. I.E fighting over stuff, someone leaving.

Money - If for some reason the project will cost too much money for us to handle.

1.5.2 Plan for handling the biggest risks

Time: we hope that our sprint meetings will help us plan good enough so we don’t encounter a problem where time is an issue.

Too complex: If we find something that seems hard to solve we will have a discussion about if it’s worth trying or not and either try or scrap it based on this decision

Group problems: solved by following the rules that we all signed.

Money: Solved by following the rules and we also plan on not using very much money at all..

1.6 Development Process

1.6.1 Development Process

We are using the atlassian set of tools, Jira, Confluence and Bitbucket. These will manage our time spent, issues and such.

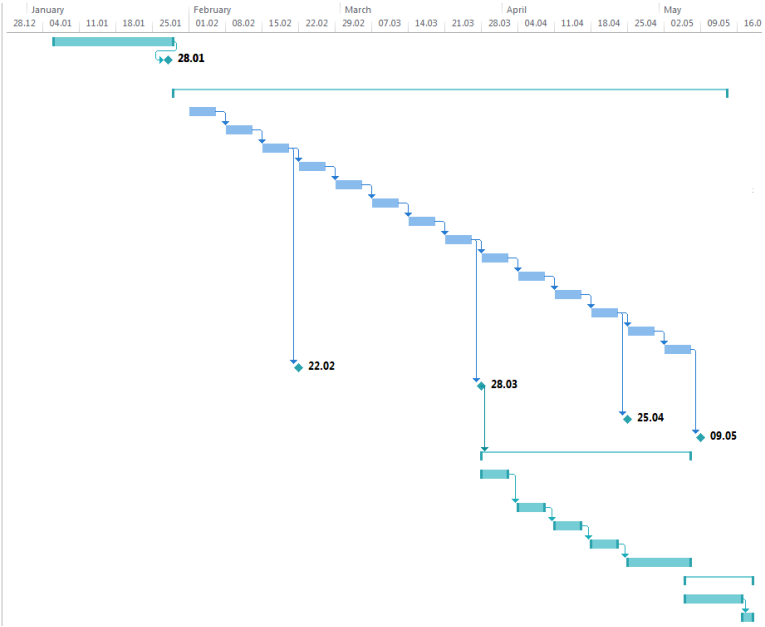
A daily scrum will be held every day at 09:00 to discuss what we have done, what we will do, if we have any issues and add or remove tasks/issues.

A sprint meeting will be held every friday at 16:00 where we will do a review of what was done during the week and add bigger tasks and issues and discuss these. Estimated duration of these meetings is 1 hour.

A work log will be kept where group members will write what the did during the day and what they are planning on doing the next day.

1.6.2 Gantt-diagram

Task Name	Duration	Start	Finish
Planning	17 days	Wed 06.01.16	Thu 28.01.16
Delivered Project Plan	0 days	Thu 28.01.16	Thu 28.01.16
Development	76 days	Fri 29.01.16	Fri 13.05.16
Sprint 1	5 days	Mon 01.02.16	Fri 05.02.16
Sprint 2	5 days	Mon 08.02.16	Fri 12.02.16
Sprint 3	5 days	Mon 15.02.16	Fri 19.02.16
Sprint 4	5 days	Mon 22.02.16	Fri 26.02.16
Sprint 5	5 days	Mon 29.02.16	Fri 04.03.16
Sprint 6	5 days	Mon 07.03.16	Fri 11.03.16
Sprint 7	5 days	Mon 14.03.16	Fri 18.03.16
Sprint 8	5 days	Mon 21.03.16	Fri 25.03.16
Sprint 9	5 days	Mon 28.03.16	Fri 01.04.16
Sprint 10	5 days	Mon 04.04.16	Fri 08.04.16
Sprint 11	5 days	Mon 11.04.16	Fri 15.04.16
Sprint 12	5 days	Mon 18.04.16	Fri 22.04.16
Sprint 13	5 days	Mon 25.04.16	Fri 29.04.16
Sprint 14	5 days	Mon 02.05.16	Fri 06.05.16
First prototype	0 days	Mon 22.02.16	Mon 22.02.16
Minimum Viable Product	0 days	Mon 28.03.16	Mon 28.03.16
Beta Version	0 days	Mon 25.04.16	Mon 25.04.16
Finished Product	0 days	Mon 09.05.16	Mon 09.05.16
Testing	30 days	Mon 28.03.16	Fri 06.05.16
Gathering initial user data	5 days	Mon 28.03.16	Fri 01.04.16
Fixing issues	5 days	Mon 04.04.16	Fri 08.04.16
Gather user data	5 days	Mon 11.04.16	Fri 15.04.16
Analyze user data	5 days	Mon 18.04.16	Fri 22.04.16
Gather user data	10 days	Mon 25.04.16	Fri 06.05.16
Documentation	9 days	Fri 06.05.16	Wed 18.05.16
Writing report	7 days	Fri 06.05.16	Mon 16.05.16
Review report	2 days	Tue 17.05.16	Wed 18.05.16



1.6.3 Comments

Our development phase consists of 14 one-week sprints. Each sprint ends with a meeting to discuss the next sprint. The product backlog will be updated and additional tasks will be added if needed.

The development period is split into four milestones. The first one is a prototype to show the potential use of the project. The second one is the MVP(Minimum Viable Product), which is the first version where we start larger scale user data collection. The Beta Version is a more finished product which will have a mostly fleshed out user data collection and will also mostly be finished. And the last one is the finished product.

When we reach a milestone, we will see whether we are on track to finishing the project. If we notice that we are not going to be able to reach what we set out to do, we will reduce the project parameters. We will also add or remove planned or useless features.

Testing will be done during the development process, because we need user data to see whether or not the AI that is being developed actually learns from the data collected. The first phase of testing will be the initial gathering of data. This data might be useless to us

later, but we need to gather it and analyze it to see if it is of use, and then we will modify the data collection if needed. If we notice that we are actually able to collect data earlier, the MVP milestone will be pushed forward.

Rules:

1. **In the event of disagreements, we will vote on the proposals. In the event of a tie, one person will be given a double vote. If a decision is decided by a double vote, the double vote is then rotated to another person.**

2. **If a group member does not complete the work he has promised, but has done his best and asked for assistance with the task he was given and it turned out to be too much, no penalty will be given.**
   If a group member does not complete the work he has promised and has given the task much less time than expected, a warning will be given. In the event of multiple warnings, rule 6 will be applied.

3. **Exceptions for rule 2 could be: Death in the family, illness, no access to computer and other unpredictable events that the other members of the group agree are valid.**

4. **For anyone taking the Professional Programming (IMT3602) during the same semester as the bachelor, it is expected that they work at least 40 hour a week. Group members are expected to pick up new tasks if their task is completed with less than the designated time.**
   In the case of them not taking Professional Programming, it is expected that the work at least 35 hours a week.

5. **In the case a member has breached rule 2 or 4 a total of 3 times, the member will be dismissed from the group. Members that don't wish to be a part of the bachelor will be dismissed from the group.**

6. **In the case of dismissal, the member in question will be stripped of all rights to the documents (project plan, bachelor document, design document, etc), repositories, asset with limited licenses. Dismissed members will not be compensated for any eventual expenses.**

7. **All expenses are divided equally among all members.**

8. **All member can sign documents on behalf of the group.**

9. **All members can call for a meeting. The person that calls for the meetings are**

responsible for the meeting agenda.

10. **During meetings, members are expected to devote all their attention to the meeting.**

11. **We have no project leader.**

Signed:

_____	_____
Jonas Reitan	Henning Luick
_____	_____
Christer Somby	Kristoffer Eidså

B. Prosjektavtale

NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

KREMEN (SE VEDLEGG 1)

(oppdragsgiver), og

Christer P. Somby, Henning E. Lucke,
Kristoffer Eidså, Jonas D. Reitan

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 28.01.16 til 18.05.16 .

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

   Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

   Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.

11. Når NTNU/AIMT også opptrer som oppdragsgiver, trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Mariusz Nowostawski

Oppdragsgivers kontaktperson (navn): Jonas D. Reitan

Student(er) (signatur): Jonas Reitan dato 28.01.16

Ola J. Sæle dato 28.01.16

Kristoffer Eidal dato 28.01.16

Henning E. Lueck dato 28.01.16

Oppdragsgiver (signatur): Jonas Reitan dato 28.01.16

Signert avtale leveres digitalt i Fronter(IMT3912)
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): _____ dato _____

Attachment 1
Prosjektavtale

Kremen

Names of group members:
Christer P. Somby
Henning E. Luick
Kristoffer Eidså
Jonas Reitan

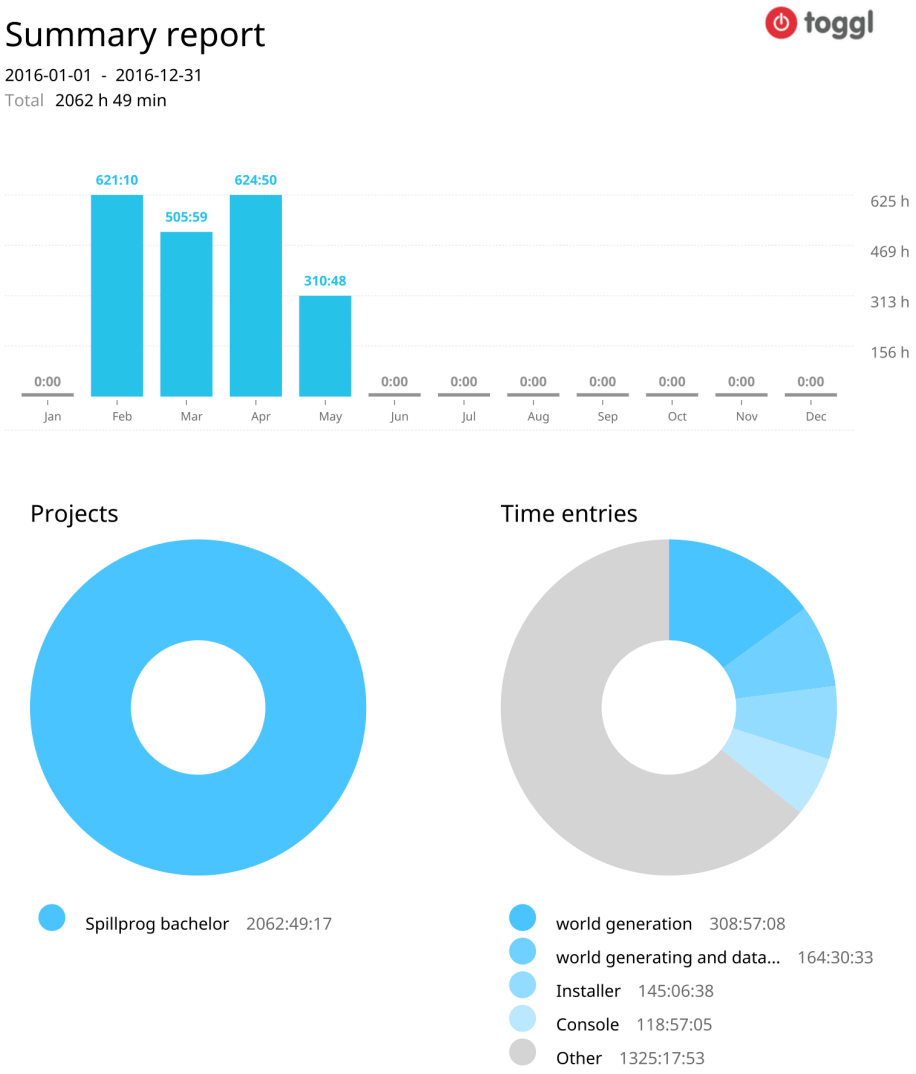
Each group member has shared ownership as shown below

Name:	Share:
Christer P. Somby	25%
Henning E. Luick	25%
Kristoffer Eidså	25%
Jonas Reitan	25%

1. In the case of dismissal, the member in question will be stripped of all rights to the documents (project plan, bachelor document, design document, etc), repositories, asset with limited licenses. Dismissed members will not be compensated for any eventual expenses.

2. In the event of disagreements, we will vote on the proposals. In the event of a tie, one person will be given a double vote. If a decision is decided by a double vote, the double vote is then rotated to another person.

C. Toggl


Summary report
toggl
2016-01-01 - 2016-12-31
Total 2062 h 49 min
621:10
624:50
505:59
310:48
0:00
625 h
469 h
313 h
156 h
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Projects
Spillprog bachelor 2062:49:17
Time entries
world generation 308:57:08
world generating and data... 164:30:33
Installer 145:06:38
Console 118:57:05
Other 1325:17:53

Projects / Time entries	Duration
Spillprog bachelor	**2062:49:17**
-	0:59:37
A* and world, discussing with Henning	4:16:08
A* and world, discussing with Henning, research	2:26:12
Added/learned libnoise	3:41:32
Added sql and boost	0:16:34
Added support for billboarding	0:51:31
added timedblock to the world generation, but nothing is spawned yet	0:37:52
adding 45 degree slopes	3:06:01
adding cube	1:41:12
adding custom component types	1:14:46
adding enemies and fixing/improving marching squares	1:58:49
adding enemy	1:18:03
adding fun enemies	2:01:48
adding pyramid	2:21:25
A* in World Implementation and theorycrafting	2:23:00
attempt at level generation using ANN	2:02:29
Attempting to install VM	3:25:44
Bachelor	4:32:04
back to more ANN stuff to get something general up and running first	1:38:45
box2d research for "gravity zones" and apparently over complicating stuff	1:59:32
bug fixing and improvements	2:39:10
confluence documentation	6:56:00
confluence, research and stuff	5:07:42
Confluence Settings	5:37:52
confluence writing	3:30:00
confluence writing about development process, noise, generation etc.	10:25:00
confluence writing and discussions	6:25:36
Console	118:57:05
Console / Cmake	5:46:08
consulting	8:55:15
creating a somewhat working level again	4:02:16
creating enemies	1:55:51
database	1:27:56
database integration	2:36:31
Database stuff	29:45:42
database, ui and world generation	6:06:40
Debugging and fixing new factory	2:49:55

2

debugging database	0:51:40
Decided not to use zeromq. helped some people, did some research on more suited libraries	3:51:46
designing chunk ideas, world generation and consulting	3:48:24
did some ui stuff and world gen	5:03:51
Discussed stuff as a group. World generation, bugfix, tweaks.	5:00:00
discussing AI in gamelab	4:14:25
discussing confluence, bachelor and art stuff, changed some stuff, wrote some stuff	5:23:50
Discussing different noise algorithms and finding bottlenecks	10:07:00
Discussing generation techniques	2:54:15
Discussing spikes, UVs and mesh with Henning, tested cave algorithm	4:14:26
Discussing w/ Henning	0:18:08
documentation	16:18:03
Documentation	24:18:16
documentation /enemy stuff	5:57:35
documenting	0:37:42
document writing	18:25:49
enemy stuff	7:41:57
Engine actor improvements	8:12:39
Experimenting with make	0:22:55
figuring out how the custom state system works and looked up box2d stuff	1:26:42
figuring out mysql examples	1:08:00
Fill gap algorithm in world generation, tested spike/moving platform stuff, tried adding damage on spikes, removed some unused crap	5:44:00
finished up some stuff, did some jira thingy and started adding a spring game mechanic	3:24:10
finishing the first version of database handler	0:52:01
Fixed bug where goal would not spawn, compile time benchmarking, testing multithreading, testing various ideas(need feedback?)	6:04:59
Fixed some bugs(Actually compiles now), added goal to world generation and fixed some errors in various actor xmls	1:47:07
Fixed some bugs in world generator, read about a* edge placement, discussed some stuff with Henning, network debugging	7:34:13
fixing memory bug	3:17:00
fixing network code	5:04:38
fixing proper cube mesh	1:49:14
fixing some animation and compile performance by rearranging includes	2:34:10
fixing stupid cpp files with template	3:46:29
Flashing stuff and figuring out stuff, bugfixing, shader stuff	4:22:29
friday meeting	4:27:08

fryday meeting	2:09:21
General object storage	4:48:03
generation algorithm improvements	1:58:39
generation improvements	0:15:03
group meeting at my place	3:00:00
hardcore merge	2:39:53
Helping eidså	0:10:30
Helping Eidså	5:14:28
Helping Jonas	0:10:47
Hopefully finishing touches on componentcreation	0:57:22
implementing ANN	1:02:14
implementing block and fixing contact listener	4:02:44
implementing block that tricks ur tiny human mind	3:21:36
Implementing database handler	11:42:29
implementing spring	2:05:53
Implementing user feedback gui	22:46:25
implementing zeromq in engine	1:48:10
improving gui	60:51:02
improving level generation	14:36:03
Installer	145:06:38
Installer, dependency	5:10:32
Installer (DLLs)	5:39:24
Installer testing	5:10:49
Lambda experimentation	1:02:32
Lan Console work	1:00:00
learning algorithms	4:01:31
level design discussing and how to do it	1:43:00
level design for world generation and looked at box2d documentation to solve my problems	3:54:14
Level Design Learning and looking at platforming A*	3:21:19
level design research and looking at various themes we might want to implement	1:42:35
level generation	9:29:12
Level Generation	60:55:56
Looked at optimization algorithms (heap, simplifying paths, etc.), a* improvements (weights) and other things that might help us	5:48:40
Looked more into fixing database bug	2:03:26
made new art for confluence	1:14:39
Made world generator a bit smarter, still much to do. Read up about algorithms to solve some problems.	6:57:31
Make stuff	14:21:43

making userfeedback work with database	16:47:29
MDP plan b	0:26:29
meeting	1:43:37
Meeting	41:43:35
meeting at jonas	12:30:20
meeting at mustad	1:28:00
Meeting at mustad	1:27:07
Meeting at Mustad	1:01:45
meeting at school	5:20:00
Meeting at school. World generation discussion and ideas	7:03:00
Meeting - Business	1:13:00
meeting, discussing	6:30:30
Meeting Jonas	9:30:00
meeting, more confluence stuff	4:09:00
Meeting Mustad	1:28:00
meeting with ådne	0:50:01
meeting with mariusz	2:31:20
Meeting with Mariusz	2:25:21
Meeting w/tweaking of gui, world and objects	1:29:42
Member function Pointer	7:38:57
Mem_fn storage	3:06:20
Merge to master, fixing conflicts, support eidså, christer mvp	1:16:28
More A*, grid, pushed some stuff, still some stuff to do	0:46:00
more enemies	1:45:29
More general object storage	0:27:00
more generation stuff. looking into MDP	1:37:50
more marching squares improvements	3:05:07
MorningMeeting	0:58:09
morning meeting	3:21:07
Morning meeting	0:41:00
mostly documentation stuff	4:08:57
Moving database and stuff	5:14:39
mustad meeting	1:28:29
Mysql	4:42:11
Network Testing	1:12:00
Pathfinding	22:49:58
Performance testing with henning	0:38:00
Planned out object generation and looked at ways to implement it	4:56:15
planning level generation	3:21:10

5

played with coin at school	5:20:27
pressing f5 on confluence	1:29:16
pushed some broken stuff.	0:08:59
Raknet implementation	2:41:50
raknet work	3:01:33
read about reverse design on Mario to learn gradually increasing difficulty in level design	3:23:20
read about stl, mlc, some tweaks to stuff	5:18:01
reading about level design	1:30:31
Reading about Machine learning	2:24:45
Reading about MDP	0:13:10
Reading AI stuff	3:46:23
reading and testing world generation stuff	5:02:13
Reading bachelor reports	2:05:38
Reading Curry	0:38:01
Reading GA	1:21:03
Reading game loop models	6:52:05
reading/looking at crossplatform compiling, world generation	4:34:07
Reading make stuff	2:37:05
Reading member function pointers	2:40:55
Reading on software models	0:37:11
Reading on Vulcan API	5:18:54
Read Variadic Templating	5:48:50
reevaluating tile-by-tile method	2:24:26
reimplementing actor loading	3:46:34
removing messaging between components	4:49:32
renaming the project	0:38:23
replace includes with forward declarations	2:07:50
replacing more sdl net code	1:29:46
Replacing sdl net with zeromq	2:05:38
Research	20:42:28
Research	12:38:49
research and note taking	4:55:07
researched good level design, world generation, group discussion on skype	7:29:00
researched NEAT (cgNEAT). World/level generation.	5:50:35
Researched stuff regarding writing a bachelor thesis, discussed/meeting, tried to fix a few bugs, looked at a* stuff	7:26:16
Research/Implementation	12:58:01
researching and implementing FSM for actors	17:19:07

running diagnostic tests	0:30:19
School	5:20:00
School meeting	3:36:00
School thing	4:37:00
scrum meeting	1:22:07
separating generation into sections and adding tinkering with AI	1:25:33
spike physics	1:04:42
Spring Meeting	2:04:11
Sprint meeting	7:30:56
Storing objects	0:28:12
Support	1:27:01
Support	2:02:48
tested some more world generation stuff	1:46:21
Tested some stuff for Henning	0:39:55
Testing ZeroMQ	1:35:13
thursday meeting	6:30:00
tried to fix database not working in debug. Gave up and did a workaround that works fine.	1:09:16
trying out mysql stuff	5:08:46
Trying to solve a weird bug in grid representation of world, need rework to solve issues with world being in negative y, tested userfeedback	4:38:15
trying to sort out a few bugs, but the compiler don't agree with me	4:13:43
trying to understand database and stuff	5:29:46
Try to fix an error with make	8:32:46
tutorial from henning	0:41:34
tutorial from Henning	0:41:40
Tweaked world generation	0:08:34
Tweaking A* to work with physics, grid representation of world	4:20:10
weekly meeting	1:57:15
went to school	17:32:16
whiteboard sketching and testing stuff	2:18:56
Working in airport	1:19:00
Working in the air	1:25:00
Working on bus	1:04:00
world gen and hitting head against a wall	7:18:00
world generating and database stuff	164:30:33
World Generation	29:52:52
world generation	308:57:08
World generation	65:03:17
world generation	7:18:41

7

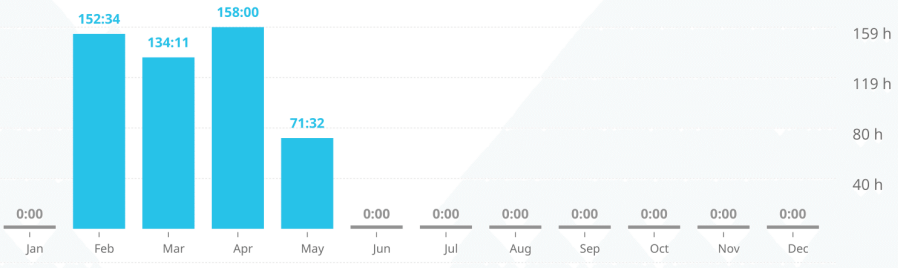
world generation and meeting with Mariusz	4:11:32
world generation and research	5:06:39
world generation and some research	9:01:28
world generation, new project test(didn't work), merge with master	2:49:57
world generation, reading/researching stuff	6:21:45
world generation testing	1:14:05
World Generation testing to see what might work or not	5:08:12
World generation testing, tweak the grid for A* and object spawning algorithm	5:15:04
world generation theories, research and planning	5:15:07
world generation with henning	3:10:42
world generation with jonas	2:43:47
world generation with layers and experimenting with stuff	4:20:42
world generation with more database integration	4:39:04
world generator	7:49:52
world generator stuff, rotating balls	1:51:04
world/level generation	14:19:59
world/level generation/research and testing some stuff	5:52:39
World Mechanics	2:39:27
world stuff	5:06:00
Writing about database, Henning crashed confluence so writing in notepad or something	1:18:24
writing about gui, user feedback and databases	3:10:50
writing about userfeedback and tweaking user feedback stuff	5:10:26
Writing and fixing stuff	14:43:08
writing client code	3:28:25
Writing in my document, worked with some box2d stuff that didn't work for my purpose	3:15:13
writing server code	3:19:25
Writing stuff about database	1:28:12
Wrote about noise algorithm and the challenges of finding the correct one	3:16:10
Wrote about noise at confluence, tested multi layered world generation	2:01:00
ZeroMQ	3:02:55

Created with toggl.com

Summary report

2016-01-01 - 2016-12-31
Total 516 h 18 min

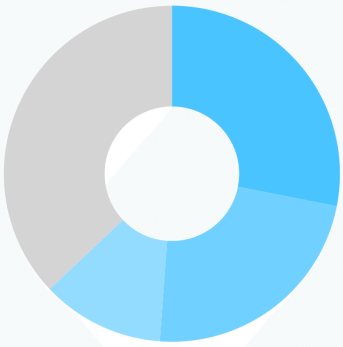
Christerpsomby selected as users



Projects

Spillprog bachelor 516:18:37

Time entries



Installer 145:06:38
Console 118:57:05
Level Generation 60:55:56
Other 191:18:58

Projects / Time entries	Duration
Spillprog bachelor	**516:18:37**
Attempting to install VM	3:25:44
Bachelor	4:32:04
Confluence Settings	5:37:52
Console	118:57:05
Console / Cmake	5:46:08
Documentation	24:18:16
Experimenting with make	0:22:55
General object storage	4:48:03
Helping eidså	0:10:30
Helping Eidså	5:14:28
Helping Jonas	0:10:47
Installer	145:06:38
Installer, dependency	5:10:32
Installer (DLLs)	5:39:24
Installer testing	5:10:49
Lambda experimentation	1:02:32
Lan Console work	1:00:00
Level Generation	60:55:56
Make stuff	14:21:43
Meeting	7:08:14
Meeting - Business	1:13:00
Meeting Jonas	9:30:00
Meeting Mustad	1:28:00
Member function Pointer	7:38:57
Mem_fn storage	3:06:20
More general object storage	0:27:00
Mysql	4:42:11
Performance testing with henning	0:38:00
Reading about Machine learning	2:24:45
Reading about MDP	0:13:10
Reading AI stuff	3:46:23
Reading bachelor reports	2:05:38
Reading Curry	0:38:01
Reading GA	1:21:03
Reading game loop models	6:52:05
Reading make stuff	2:37:05
Reading member function pointers	2:40:55

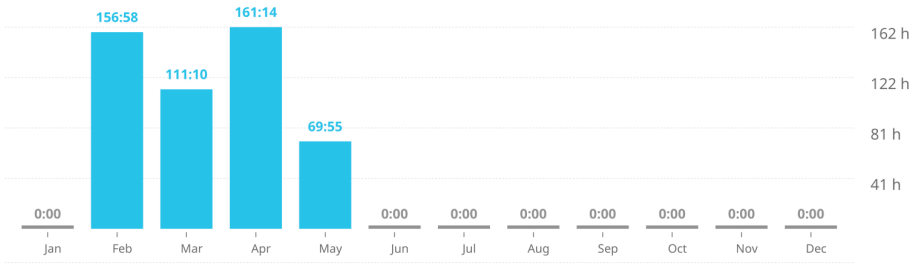
Reading on software models	0:37:11
Reading on Vulcan API	5:18:54
Read Variadic Templating	5:48:50
School	5:20:00
School meeting	3:36:00
School thing	4:37:00
Spring Meeting	2:04:11
Sprint meeting	5:03:46
Storing objects	0:28:12
Try to fix an error with make	8:32:46
tutorial from henning	0:41:34
Working in airport	1:19:00
Working in the air	1:25:00
Working on bus	1:04:00

Created with toggl.com

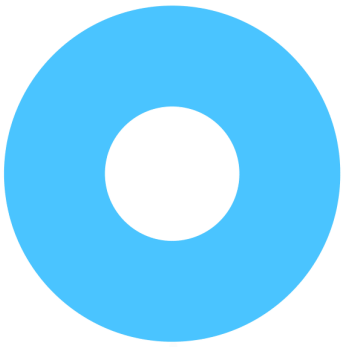
Summary report

2016-01-01 - 2016-12-31
Total 499 h 18 min

Henningel selected as users



Projects



Spillprog bachelor 499:18:25

Time entries

world generation 220:30:38
Other 278:47:47

Projects / Time entries	Duration
Spillprog bachelor	**499:18:25**
-	0:59:37
adding 45 degree slopes	3:06:01
adding cube	1:41:12
adding custom component types	1:14:46
adding enemies and fixing/improving marching squares	1:58:49
adding enemy	1:18:03
adding fun enemies	2:01:48
adding pyramid	2:21:25
attempt at level generation using ANN	2:02:29
back to more ANN stuff to get something general up and running first	1:38:45
bug fixing and improvements	2:39:10
consulting	8:08:15
creating a somewhat working level again	4:02:16
creating enemies	1:55:51
database	1:27:56
database integration	2:36:31
Debugging and fixing new factory	2:49:55
Decided not to use zeromq. helped some people, did some research on more suited libraries	3:51:46
discussing AI in gamelab	4:14:25
documentation	16:18:03
documentation /enemy stuff	5:57:35
documenting	0:37:42
document writing	18:25:49
enemy stuff	7:41:57
Engine actor improvements	8:12:39
fixing memory bug	3:17:00
fixing network code	5:04:38
fixing proper cube mesh	1:49:14
fixing some animation and compile performance by rearranging includes	2:34:10
fixing stupid cpp files with template	3:46:29
friday meeting	4:27:08
fryday meeting	2:09:21
generation algorithm improvements	1:58:39
generation improvements	0:15:03
hardcore merge	2:39:53
Hopefully finishing touches on componentcreation	0:57:22
implementing ANN	1:02:14

implementing zeromq in engine	1:48:10
improving level generation	14:36:03
learning algorithms	4:01:31
level generation	9:29:12
MDP plan b	0:26:29
meeting	1:27:19
Meeting	2:28:37
meeting at jonas	3:00:01
meeting at mustad	1:28:00
meeting with ådne	0:50:01
meeting with mariusz	2:31:20
more enemies	1:45:29
more generation stuff. looking into MDP	1:37:50
more marching squares improvements	3:05:07
MorningMeeting	0:58:09
morning meeting	3:21:07
mostly documentation stuff	4:08:57
planning level generation	3:21:10
played with coin at school	5:20:27
Raknet implementation	2:41:50
raknet work	3:01:33
reevaluating tile-by-tile method	2:24:26
reimplementing actor loading	3:46:34
removing messaging between components	4:49:32
renaming the project	0:38:23
replace includes with forward declarations	2:07:50
replacing more sdl net code	1:29:46
Replacing sdl net with zeromq	2:05:38
researching and implementing FSM for actors	17:19:07
running diagnostic tests	0:30:19
scrum meeting	1:22:07
separating generation into sections and adding tinkering with AI	1:25:33
spike physics	1:04:42
Testing ZeroMQ	1:35:13
thursday meeting	6:30:00
weekly meeting	1:57:15
went to school	3:35:28
world generation	220:30:38
world generation with jonas	2:43:47

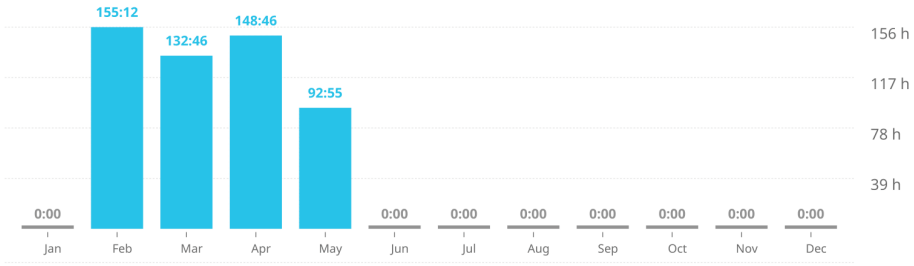
world generation with more database integration	4:39:04
writing client code	3:28:25
writing server code	3:19:25
ZeroMQ	3:02:55

Created with toggl.com

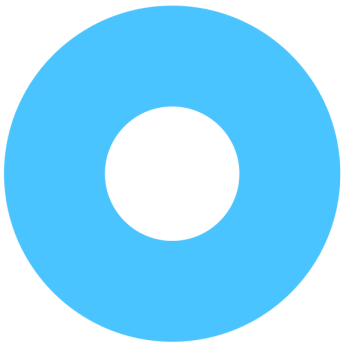
Summary report

2016-01-01 - 2016-12-31
Total 529 h 41 min

Jonas Reitan selected as users

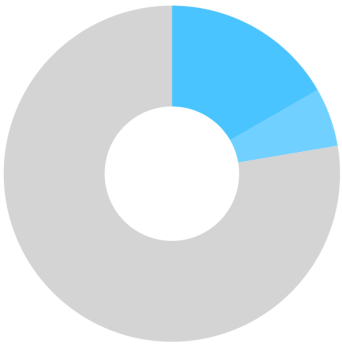


Projects



Spillprog bachelor 529:41:21

Time entries



world generation 88:26:30
World Generation 29:52:52
Other 411:21:59

Projects / Time entries	Duration
Spillprog bachelor	**529:41:21**
A* and world, discussing with Henning	4:16:08
A* and world, discussing with Henning, research	2:26:12
Added/learned libnoise	3:41:32
Added sql and boost	0:16:34
Added support for billboarding	0:51:31
added timedblock to the world generation, but nothing is spawned yet	0:37:52
A* in World Implementation and theorycrafting	2:23:00
box2d research for "gravity zones" and apparently over complicating stuff	1:59:32
confluence documentation	6:56:00
confluence, research and stuff	5:07:42
confluence writing	3:30:00
confluence writing about development process, noise, generation etc.	10:25:00
confluence writing and discussions	6:25:36
consulting	0:47:00
designing chunk ideas, world generation and consulting	3:48:24
Discussed stuff as a group. World generation, bugfix, tweaks.	5:00:00
discussing confluence, bachelor and art stuff, changed some stuff, wrote some stuff	5:23:50
Discussing different noise algorithms and finding bottlenecks	10:07:00
Discussing generation techniques	2:54:15
Discussing spikes, UVs and mesh with Henning, tested cave algorithm	4:14:26
Discussing w/ Henning	0:18:08
Fill gap algorithm in world generation, tested spike/moving platform stuff, tried adding damage on spikes, removed some unused crap	5:44:00
Fixed bug where goal would not spawn, compile time benchmarking, testing multithreading, testing various ideas(need feedback?)	6:04:59
Fixed some bugs(Actually compiles now), added goal to world generation and fixed some errors in various actor xmls	1:47:07
Fixed some bugs in world generator, read about a* edge placement, discussed some stuff with Henning, network debugging	7:34:13
Flashing stuff and figuring out stuff, bugfixing, shader stuff	4:22:29
group meeting at my place	3:00:00
level design discussing and how to do it	1:43:00
level design for world generation and looked at box2d documentation to solve my problems	3:54:14
Level Design Learning and looking at platforming A*	3:21:19
level design research and looking at various themes we might want to implement	1:42:35
Looked at optimization algorithms (heap, simplifying paths, etc.), a* improvements (weights) and other things that might help us	5:48:40

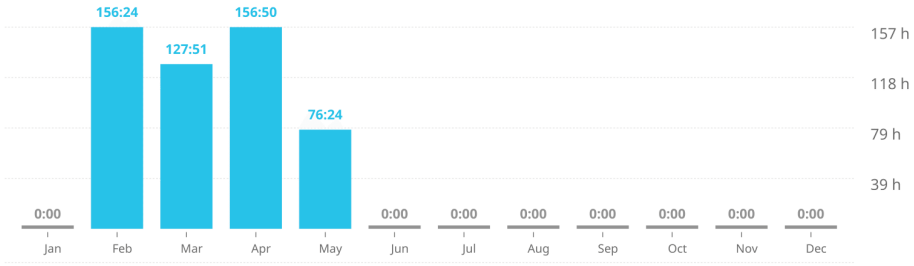
made new art for confluence	1:14:39
Made world generator a bit smarter, still much to do. Read up about algorithms to solve some problems.	6:57:31
meeting	0:16:18
Meeting	14:32:40
Meeting at Mustad	1:01:45
meeting at school	5:20:00
Meeting at school. World generation discussion and ideas	7:03:00
meeting, discussing	6:30:30
meeting, more confluence stuff	4:09:00
Meeting w/tweaking of gui, world and objects	1:29:42
Merge to master, fixing conflicts, support eidså, christer mvp	1:16:28
More A*, grid, pushed some stuff, still some stuff to do	0:46:00
Morning meeting	0:41:00
mustad meeting	1:28:29
Network Testing	1:12:00
Pathfinding	22:49:58
Planned out object generation and looked at ways to implement it	4:56:15
pushed some broken stuff.	0:08:59
read about reverse design on Mario to learn gradually increasing difficulty in level design	3:23:20
read about stl, mlc, some tweaks to stuff	5:18:01
reading about level design	1:30:31
reading and testing world generation stuff	5:02:13
reading/looking at crossplatform compiling, world generation	4:34:07
Research	20:42:28
research and note taking	4:55:07
researched good level design, world generation, group discussion on skype	7:29:00
researched NEAT (cgNEAT). World/level generation.	5:50:35
Researched stuff regarding writing a bachelor thesis, discussed/meeting, tried to fix a few bugs, looked at a* stuff	7:26:16
Research/Implementation	12:58:01
Support	1:27:01
Support	2:02:48
tested some more world generation stuff	1:46:21
Tested some stuff for Henning	0:39:55
Trying to solve a weird bug in grid representation of world, need rework to solve issues with world being in negative y, tested userfeedback	4:38:15

trying to sort out a few bugs, but the compiler don't agree with me	4:13:43
Tweaked world generation	0:08:34
Tweaking A* to work with physics, grid representation of world	4:20:10
whiteboard sketching and testing stuff	2:18:56
world generation	88:26:30
World Generation	29:52:52
world generation	7:18:41
world generation and meeting with Mariusz	4:11:32
world generation and research	5:06:39
world generation and some research	9:01:28
world generation, new project test(didn't work), merge with master	2:49:57
world generation, reading/researching stuff	6:21:45
world generation testing	1:14:05
World Generation testing to see what might work or not	5:08:12
World generation testing, tweak the grid for A* and object spawning algorithm	5:15:04
world generation theories, research and planning	5:15:07
world generation with henning	3:10:42
world generation with layers and experimenting with stuff	4:20:42
world generator	7:49:52
world generator stuff, rotating balls	1:51:04
world/level generation	14:19:59
world/level generation/research and testing some stuff	5:52:39
World Mechanics	2:39:27
world stuff	5:06:00
Wrote about noise algorithm and the challenges of finding the correct one	3:16:10
Wrote about noise at confluence, tested multi layered world generation	2:01:00

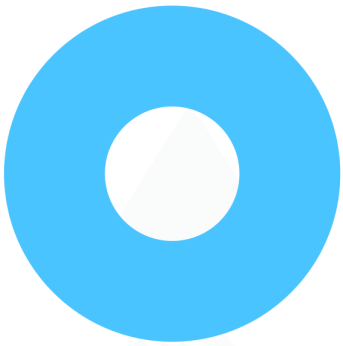
Summary report

2016-01-01 - 2016-12-31
Total 517 h 30 min

Kristoffer Eidså selected as users

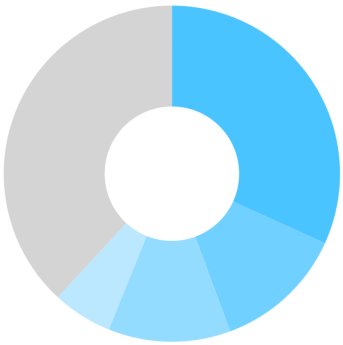


Projects



Spillprog bachelor 517:30:54

Time entries



world generating and data... 164:30:33
World generation 65:03:17
improving gui 60:51:02
Database stuff 29:45:42
Other 197:20:20

Projects / Time entries	Duration
Spillprog bachelor	**517:30:54**
Database stuff	29:45:42
database, ui and world generation	6:06:40
debugging database	0:51:40
did some ui stuff and world gen	5:03:51
figuring out how the custom state system works and looked up box2d stuff	1:26:42
figuring out mysql examples	1:08:00
finished up some stuff, did some jira thingy and started adding a spring game mechanic	3:24:10
finishing the first version of database handler	0:52:01
implementing block and fixing contact listener	4:02:44
implementing block that tricks ur tiny human mind	3:21:36
Implementing database handler	11:42:29
implementing spring	2:05:53
Implementing user feedback gui	22:46:25
improving gui	60:51:02
Looked more into fixing database bug	2:03:26
making userfeedback work with database	16:47:29
Meeting	17:34:04
meeting at jonas	9:30:19
Meeting at mustad	1:27:07
Meeting with Mariusz	2:25:21
Moving database and stuff	5:14:39
pressing f5 on confluence	1:29:16
Research	12:38:49
Sprint meeting	2:27:10
tried to fix database not working in debug. Gave up and did a workaround that works fine.	1:09:16
trying out mysql stuff	5:08:46
trying to understand database and stuff	5:29:46
tutorial from Henning	0:41:40
went to school	13:56:48
world gen and hitting head against a wall	7:18:00
world generating and database stuff	164:30:33
World generation	65:03:17
Writing about database, Henning crashed confluence so writing in notepad or something	1:18:24
writing about gui, user feedback and databases	3:10:50
writing about userfeedback and tweaking user feedback stuff	5:10:26
Writing and fixing stuff	14:43:08
Writing in my document, worked with some box2d stuff that didn't work for my purpose	3:15:13

Writing stuff about database 1:28:12

Created with toggl.com

Detailed report

toggl

2016-01-01 - 2016-12-31

Total 2065 h 13 min

Date	Description	Duration	User
02-01	**Bachelor** Spillprog bachelor	**4:32:04** 09:00-13:32	Christerpsomby
02-01	**Meeting** Spillprog bachelor	**0:52:00** 09:00-09:52	Kristoffer Eidså
02-01	**MorningMeeting** Spillprog bachelor - [Weekly]	**0:50:00** 09:00-09:50	Henningel
02-01	**Meeting** Spillprog bachelor	**0:27:26** 09:15-09:42	Jonas Reitan
02-01	**Meeting** Spillprog bachelor	**0:01:09** 09:40-09:41	Jonas Reitan
02-01	**Pathfinding** Spillprog bachelor	**6:26:54** 09:41-16:08	Jonas Reitan
02-01	**Research** Spillprog bachelor	**3:04:21** 09:53-12:58	Kristoffer Eidså
02-01	**Research** Spillprog bachelor	**1:12:40** 13:07-14:19	Kristoffer Eidså
02-01	**ZeroMQ** Spillprog bachelor	**1:19:17** 21:31-22:50	Henningel
02-01	**ZeroMQ** Spillprog bachelor	**0:36:09** 23:14-23:50	Henningel
02-02	**ZeroMQ** Spillprog bachelor	**0:25:36** 00:49-01:15	Henningel
02-02	**ZeroMQ** Spillprog bachelor	**0:41:53** 01:40-02:22	Henningel
02-02	**Meeting** Spillprog bachelor	**0:10:36** 09:00-09:11	Kristoffer Eidså
02-02	**MorningMeeting** Spillprog bachelor	**0:08:09** 09:03-09:11	Henningel
02-02	**Research** Spillprog bachelor	**4:41:09** 09:11-13:52	Kristoffer Eidså
02-02	**Testing ZeroMQ** Spillprog bachelor	**0:43:07** 09:12-09:55	Henningel
02-02	**Testing ZeroMQ** Spillprog bachelor	**0:52:06** 10:16-11:08	Henningel
02-02	**Pathfinding** Spillprog bachelor	**2:39:05** 11:00-13:39	Jonas Reitan
02-02	**Meeting** Spillprog bachelor	**1:27:00** 15:00-16:27	Jonas Reitan

02-02	**Reading make stuff**	**2:37:05**	Christerpsomby
	Spillprog bachelor	15:06-17:43	
02-02	**Meeting**	**0:13:43**	Kristoffer Eidså
	Spillprog bachelor	15:14-15:27	
02-02	**Pathfinding**	**1:54:00**	Jonas Reitan
	Spillprog bachelor	16:27-18:21	
02-02	**Engine actor improvements**	**1:33:39**	Henningel
	Spillprog bachelor	23:44-01:18	
02-03	**Meeting**	**0:25:06**	Kristoffer Eidså
	Spillprog bachelor	09:00-09:25	
02-03	**Pathfinding**	**6:49:16**	Jonas Reitan
	Spillprog bachelor	09:11-16:01	
02-03	**trying out mysql stuff**	**3:06:33**	Kristoffer Eidså
	Spillprog bachelor	09:25-12:32	
02-03	**Engine actor improvements**	**1:55:34**	Henningel
	Spillprog bachelor	09:44-11:39	
02-03	**trying out mysql stuff**	**2:02:13**	Kristoffer Eidså
	Spillprog bachelor	14:39-16:41	
02-03	**Make stuff**	**6:51:33**	Christerpsomby
	Spillprog bachelor	15:07-21:58	
02-03	**Engine actor improvements**	**2:01:59**	Henningel
	Spillprog bachelor	17:32-19:34	
02-03	**Implementing database handler**	**3:13:42**	Kristoffer Eidså
	Spillprog bachelor	19:00-22:14	
02-03	**Helping eidså**	**0:10:30**	Christerpsomby
	Spillprog bachelor	22:03-22:14	
02-04	**Engine actor improvements**	**2:41:27**	Henningel
	Spillprog bachelor	03:35-06:17	
02-04	**Implementing database handler**	**8:28:47**	Kristoffer Eidså
	Spillprog bachelor	09:00-17:29	
02-04	**reimplementing actor loading**	**3:46:34**	Henningel
	Spillprog bachelor	10:01-13:47	
02-04	**Mysql**	**4:42:11**	Christerpsomby
	Spillprog bachelor	13:06-17:48	
02-04	**Pathfinding**	**5:00:43**	Jonas Reitan
	Spillprog bachelor	13:28-18:29	
02-04	**Debugging and fixing new factory**	**2:49:55**	Henningel
	Spillprog bachelor	14:15-17:04	
02-04	**Experimenting with make**	**0:22:55**	Christerpsomby
	Spillprog bachelor	18:28-18:51	
02-04	**Hopefully finishing touches on componentcreation**	**0:57:22**	Henningel
	Spillprog bachelor	18:37-19:34	

02-05	**removing messaging between components**	**4:49:32**	Henningel
	Spillprog bachelor	10:17-15:06	
02-05	**Research/Implementation**	**5:36:15**	Jonas Reitan
	Spillprog bachelor	12:56-18:32	
02-05	**figuring out mysql examples**	**1:08:00**	Kristoffer Eidså
	Spillprog bachelor	12:58-14:06	
02-05	**finishing the first version of database handler**	**0:52:01**	Kristoffer Eidså
	Spillprog bachelor	14:07-14:59	
02-05	**Try to fix an error with make**	**2:28:43**	Christerpsomby
	Spillprog bachelor	14:15-16:44	
02-05	**Try to fix an error with make**	**4:31:50**	Christerpsomby
	Spillprog bachelor	16:50-21:21	
02-05	**Meeting**	**2:29:15**	Jonas Reitan
	Spillprog bachelor	18:50-21:19	
02-05	**Meeting**	**2:28:37**	Henningel
	Spillprog bachelor	18:50-21:18	
02-05	**Sprint meeting**	**2:27:10**	Kristoffer Eidså
	Spillprog bachelor	18:51-21:18	
02-05	**Research/Implementation**	**1:07:00**	Jonas Reitan
	Spillprog bachelor	21:19-22:26	
02-05	**Try to fix an error with make**	**1:32:13**	Christerpsomby
	Spillprog bachelor	23:21-00:53	
02-06	**Make stuff**	**1:45:43**	Christerpsomby
	Spillprog bachelor	20:51-22:37	
02-06	**Research**	**2:02:29**	Kristoffer Eidså
	Spillprog bachelor	23:48-01:50	
02-07	**fixing stupid cpp files with template**	**3:46:29**	Henningel
	Spillprog bachelor	00:30-04:16	
02-07	**Make stuff**	**2:20:46**	Christerpsomby
	Spillprog bachelor	01:49-04:09	
02-07	**Research**	**1:07:30**	Kristoffer Eidså
	Spillprog bachelor	02:59-04:07	
02-07	**Research/Implementation**	**2:14:21**	Jonas Reitan
	Spillprog bachelor	03:53-06:07	
02-07	**creating a somewhat working level again**	**4:02:16**	Henningel
	Spillprog bachelor	07:39-11:42	
02-07	**Research/Implementation**	**3:00:13**	Jonas Reitan
	Spillprog bachelor	07:45-10:45	
02-07	**Support**	**0:57:00**	Jonas Reitan
	Spillprog bachelor	10:45-11:42	
02-07	**Research**	**0:30:40**	Kristoffer Eidså
	Spillprog bachelor	15:00-15:31	

02-07	**improving gui**	**1:29:30**	Kristoffer Eidså
	Spillprog bachelor	15:31-17:01	
02-07	**Make stuff**	**3:23:41**	Christerpsomby
	Spillprog bachelor	16:03-19:27	
02-07	**implementing zeromq in engine**	**1:48:10**	Henningel
	Spillprog bachelor	18:50-20:39	
02-07	**tutorial from Henning**	**0:41:40**	Kristoffer Eidså
	Spillprog bachelor	19:27-20:08	
02-07	**tutorial from henning**	**0:41:34**	Christerpsomby
	Spillprog bachelor	19:27-20:08	
02-07	**Console**	**3:52:21**	Christerpsomby
	Spillprog bachelor	20:08-00:01	
02-07	**improving gui**	**2:11:07**	Kristoffer Eidså
	Spillprog bachelor	20:08-22:20	
02-07	**Replacing sdl net with zeromq**	**2:05:38**	Henningel
	Spillprog bachelor	21:44-23:50	
02-08	**improving gui**	**4:16:40**	Kristoffer Eidså
	Spillprog bachelor	12:00-16:17	
02-08	**World Generation**	**1:24:43**	Jonas Reitan
	Spillprog bachelor	14:15-15:40	
02-08	**World Generation**	**1:27:05**	Jonas Reitan
	Spillprog bachelor	15:56-17:23	
02-08	**World Generation**	**3:04:15**	Jonas Reitan
	Spillprog bachelor	17:38-20:42	
02-08	**Decided not to use zeromq. helped some people, did some research on more suited libraries**	**3:51:46**	Henningel
	Spillprog bachelor	18:01-21:53	
02-08	**improving gui**	**1:55:06**	Kristoffer Eidså
	Spillprog bachelor	20:36-22:31	
02-08	**improving gui**	**0:14:17**	Kristoffer Eidså
	Spillprog bachelor	23:04-23:19	
02-08	**World Generation**	**0:48:26**	Jonas Reitan
	Spillprog bachelor	23:31-00:19	
02-09	**World Generation**	**0:27:40**	Jonas Reitan
	Spillprog bachelor	00:28-00:56	
02-09	**Console**	**1:19:19**	Christerpsomby
	Spillprog bachelor	03:44-05:03	
02-09	**Meeting**	**0:16:00**	Christerpsomby
	Spillprog bachelor	09:00-09:16	
02-09	**Meeting**	**0:16:26**	Kristoffer Eidså
	Spillprog bachelor	09:00-09:16	
02-09	**morning meeting**	**0:15:53**	Henningel
	Spillprog bachelor	09:01-09:17	

02-09	**Console**	**2:35:00**	Christerpsomby
	Spillprog bachelor	09:17-11:52	
02-09	**improving gui**	**2:43:02**	Kristoffer Eidså
	Spillprog bachelor	09:17-12:00	
02-09	**Raknet implementation**	**2:41:50**	Henningel
	Spillprog bachelor	09:45-12:27	
02-09	**raknet work**	**0:29:58**	Henningel
	Spillprog bachelor	13:37-14:06	
02-09	**improving gui**	**0:05:16**	Kristoffer Eidså
	Spillprog bachelor	15:15-15:21	
02-09	**Research**	**1:44:39**	Jonas Reitan
	Spillprog bachelor	15:33-17:17	
02-09	**improving gui**	**2:37:53**	Kristoffer Eidså
	Spillprog bachelor	16:56-19:34	
02-09	**World Generation**	**2:56:33**	Jonas Reitan
	Spillprog bachelor	19:15-22:11	
02-09	**Console**	**1:15:46**	Christerpsomby
	Spillprog bachelor	21:07-22:23	
02-09	**Research**	**0:10:36**	Jonas Reitan
	Spillprog bachelor	22:11-22:22	
02-10	**raknet work**	**2:31:35**	Henningel
	Spillprog bachelor	00:00-02:31	
02-10	**Console**	**1:31:21**	Christerpsomby
	Spillprog bachelor	01:17-02:49	
02-10	**replacing more sdl net code**	**1:29:46**	Henningel
	Spillprog bachelor	03:26-04:56	
02-10	**Console**	**2:27:35**	Christerpsomby
	Spillprog bachelor	04:48-07:16	
02-10	**improving gui**	**2:00:03**	Kristoffer Eidså
	Spillprog bachelor	12:00-14:00	
02-10	**writing server code**	**3:19:25**	Henningel
	Spillprog bachelor	12:40-15:59	
02-10	**World Generation**	**3:48:15**	Jonas Reitan
	Spillprog bachelor	13:13-17:01	
02-10	**improving gui**	**3:30:43**	Kristoffer Eidså
	Spillprog bachelor	14:30-18:01	
02-10	**consulting**	**0:35:45**	Henningel
	Spillprog bachelor	16:25-17:00	
02-10	**Research**	**1:25:15**	Jonas Reitan
	Spillprog bachelor	17:03-18:28	
02-10	**Console**	**2:49:25**	Christerpsomby
	Spillprog bachelor	19:20-22:09	
02-11	**writing client code**	**3:28:25**	Henningel
	Spillprog bachelor	01:11-04:39	

02-11	**Research**	**2:23:18**	Jonas Reitan
	Spillprog bachelor	03:06-05:29	
02-11	**Console**	**1:50:42**	Christerpsomby
	Spillprog bachelor	04:27-06:17	
02-11	**Console**	**3:34:32**	Christerpsomby
	Spillprog bachelor	06:19-09:54	
02-11	**fixing network code**	**0:51:57**	Henningel
	Spillprog bachelor	06:46-07:38	
02-11	**World Generation**	**3:02:55**	Jonas Reitan
	Spillprog bachelor	08:47-11:50	
02-11	**morning meeting**	**0:31:28**	Henningel
	Spillprog bachelor	09:02-09:33	
02-11	**improving gui**	**10:34:14**	Kristoffer Eidså
	Spillprog bachelor	09:30-20:04	
02-11	**fixing network code**	**4:12:41**	Henningel
	Spillprog bachelor	09:33-13:46	
02-11	**implementing ANN**	**1:02:14**	Henningel
	Spillprog bachelor	16:11-17:14	
02-11	**World Generation**	**1:17:00**	Jonas Reitan
	Spillprog bachelor	17:00-18:17	
02-11	**planning level generation**	**0:21:40**	Henningel
	Spillprog bachelor	17:14-17:35	
02-11	**consulting**	**2:08:15**	Henningel
	Spillprog bachelor	17:36-19:44	
02-11	**Research**	**1:09:57**	Jonas Reitan
	Spillprog bachelor	18:18-19:28	
02-11	**Console**	**3:38:20**	Christerpsomby
	Spillprog bachelor	20:33-00:11	
02-12	**Console / Cmake**	**3:35:08**	Christerpsomby
	Spillprog bachelor	01:57-05:32	
02-12	**Console / Cmake**	**2:11:00**	Christerpsomby
	Spillprog bachelor	09:54-12:05	
02-12	**improving gui**	**4:54:17**	Kristoffer Eidså
	Spillprog bachelor	11:00-15:55	
02-12	**World Generation**	**6:33:00**	Jonas Reitan
	Spillprog bachelor	15:02-21:35	
02-12	**Meeting**	**2:31:33**	Christerpsomby
	Spillprog bachelor	16:00-18:31	
02-12	**Meeting**	**2:09:47**	Kristoffer Eidså
	Spillprog bachelor	16:00-18:10	
02-12	**fryday meeting**	**2:09:21**	Henningel
	Spillprog bachelor	16:00-18:10	
02-12	**consulting**	**1:09:30**	Henningel
	Spillprog bachelor	18:38-19:47	

02-12	**Research**	**2:38:31**	Jonas Reitan
	Spillprog bachelor	21:52-00:31	
02-13	**Research**	**1:01:24**	Jonas Reitan
	Spillprog bachelor	04:23-05:25	
02-13	**planning level generation**	**2:59:30**	Henningel
	Spillprog bachelor	05:07-08:06	
02-13	**Research**	**1:13:00**	Jonas Reitan
	Spillprog bachelor	05:26-06:39	
02-13	**Support**	**1:27:01**	Jonas Reitan
	Spillprog bachelor	06:40-08:07	
02-13	**Network Testing**	**1:12:00**	Jonas Reitan
	Spillprog bachelor	08:08-09:20	
02-13	**Console**	**2:14:00**	Christerpsomby
	Spillprog bachelor	11:16-13:30	
02-13	**attempt at level generation using ANN**	**1:21:00**	Henningel
	Spillprog bachelor	19:07-20:28	
02-13	**Console**	**0:16:26**	Christerpsomby
	Spillprog bachelor	21:45-22:01	
02-14	**attempt at level generation using ANN**	**0:41:29**	Henningel
	Spillprog bachelor	00:50-01:31	
02-14	**Console**	**5:09:02**	Christerpsomby
	Spillprog bachelor	02:05-07:14	
02-14	**level generation**	**2:20:00**	Henningel
	Spillprog bachelor	12:43-15:03	
02-14	**improving gui**	**2:33:00**	Kristoffer Eidså
	Spillprog bachelor	14:59-17:32	
02-14	**consulting**	**0:14:46**	Henningel
	Spillprog bachelor	15:44-15:59	
02-14	**level generation**	**1:16:52**	Henningel
	Spillprog bachelor	16:31-17:48	
02-14	**Support**	**1:05:48**	Jonas Reitan
	Spillprog bachelor	16:59-18:05	
02-14	**improving gui**	**1:16:23**	Kristoffer Eidså
	Spillprog bachelor	17:32-18:49	
02-14	**Console**	**2:49:57**	Christerpsomby
	Spillprog bachelor	18:33-21:23	
02-14	**improving gui**	**0:57:59**	Kristoffer Eidså
	Spillprog bachelor	20:53-21:51	
02-15	**Console**	**1:07:15**	Christerpsomby
	Spillprog bachelor	07:53-09:00	
02-15	**Meeting**	**0:11:00**	Jonas Reitan
	Spillprog bachelor	09:00-09:11	
02-15	**Meeting**	**0:11:23**	Christerpsomby
	Spillprog bachelor	09:00-09:11	

02-15	**improving gui**	**6:01:17**	Kristoffer Eidså
	Spillprog bachelor	12:00-18:01	
02-15	**Console**	**2:02:00**	Christerpsomby
	Spillprog bachelor	12:15-14:17	
02-15	**Research**	**6:13:33**	Jonas Reitan
	Spillprog bachelor	16:00-22:13	
02-16	**Research/Implementation**	**1:00:12**	Jonas Reitan
	Spillprog bachelor	03:28-04:28	
02-16	**Console**	**5:01:25**	Christerpsomby
	Spillprog bachelor	04:42-09:43	
02-16	**improving gui**	**3:49:28**	Kristoffer Eidså
	Spillprog bachelor	09:30-13:19	
02-16	**level generation**	**1:48:06**	Henningel
	Spillprog bachelor	11:35-13:23	
02-16	**Meeting**	**2:05:51**	Jonas Reitan
	Spillprog bachelor	13:01-15:07	
02-16	**Meeting**	**1:24:00**	Christerpsomby
	Spillprog bachelor	13:16-14:40	
02-16	**meeting with mariusz**	**1:16:20**	Henningel
	Spillprog bachelor	13:23-14:40	
02-16	**Meeting with Mariusz**	**1:10:11**	Kristoffer Eidså
	Spillprog bachelor	13:30-14:41	
02-16	**level generation**	**1:52:37**	Henningel
	Spillprog bachelor	15:21-17:13	
02-16	**improving gui**	**0:28:15**	Kristoffer Eidså
	Spillprog bachelor	15:22-15:51	
02-16	**adding enemy**	**0:07:43**	Henningel
	Spillprog bachelor	17:50-17:58	
02-16	**improving gui**	**0:33:55**	Kristoffer Eidså
	Spillprog bachelor	17:56-18:30	
02-16	**World Mechanics**	**1:33:26**	Jonas Reitan
	Spillprog bachelor	19:25-20:58	
02-17	**Console**	**0:41:44**	Christerpsomby
	Spillprog bachelor	07:19-08:00	
02-17	**Meeting**	**0:03:00**	Christerpsomby
	Spillprog bachelor	09:00-09:03	
02-17	**Reading GA**	**1:21:03**	Christerpsomby
	Spillprog bachelor	09:59-11:20	
02-17	**improving gui**	**4:45:53**	Kristoffer Eidså
	Spillprog bachelor	11:00-15:46	
02-17	**Helping Eidså**	**0:34:40**	Christerpsomby
	Spillprog bachelor	12:42-13:16	
02-17	**World Generation**	**5:03:00**	Jonas Reitan
	Spillprog bachelor	12:57-18:00	

02-17	**improving gui**	**1:20:22**	Kristoffer Eidså
	Spillprog bachelor	15:52-17:13	
02-17	**World Mechanics**	**1:06:01**	Jonas Reitan
	Spillprog bachelor	19:58-21:04	
02-18	**Research**	**0:34:15**	Jonas Reitan
	Spillprog bachelor	01:47-02:21	
02-18	**Reading Curry**	**0:38:01**	Christerpsomby
	Spillprog bachelor	04:15-04:53	
02-18	**Reading member function pointers**	**2:40:55**	Christerpsomby
	Spillprog bachelor	04:53-07:34	
02-18	**learning algorithms**	**4:01:31**	Henningel
	Spillprog bachelor	06:45-10:47	
02-18	**Member function Pointer**	**0:50:23**	Christerpsomby
	Spillprog bachelor	07:34-08:24	
02-18	**Meeting**	**0:50:18**	Christerpsomby
	Spillprog bachelor	09:00-09:50	
02-18	**Meeting**	**1:46:45**	Kristoffer Eidså
	Spillprog bachelor	09:00-10:47	
02-18	**improving gui**	**2:32:22**	Kristoffer Eidså
	Spillprog bachelor	11:09-13:41	
02-18	**generation algorithm improvements**	**1:58:39**	Henningel
	Spillprog bachelor	12:11-14:10	
02-18	**Discussing different noise algorithms and finding bottlenecks**	**10:07:00**	Jonas Reitan
	Spillprog bachelor	13:15-23:22	
02-18	**Implementing user feedback gui**	**0:07:53**	Kristoffer Eidså
	Spillprog bachelor	13:41-13:49	
02-18	**Implementing user feedback gui**	**2:33:19**	Kristoffer Eidså
	Spillprog bachelor	15:57-18:30	
02-18	**more generation stuff. looking into MDP**	**1:37:50**	Henningel
	Spillprog bachelor	17:01-18:39	
02-18	**MDP plan b**	**0:26:29**	Henningel
	Spillprog bachelor	18:48-19:14	
02-19	**Console**	**3:36:28**	Christerpsomby
	Spillprog bachelor	07:47-11:23	
02-19	**back to more ANN stuff to get something general up and running first**	**1:38:45**	Henningel
	Spillprog bachelor	08:06-09:45	
02-19	**generation improvements**	**0:15:03**	Henningel
	Spillprog bachelor	10:01-10:16	
02-19	**Implementing user feedback gui**	**3:30:34**	Kristoffer Eidså
	Spillprog bachelor	12:29-16:00	
02-19	**Research**	**2:08:00**	Jonas Reitan
	Spillprog bachelor	13:17-15:25	

02-19	**improving level generation**	**2:05:33**	Henningel
	Spillprog bachelor	14:01-16:06	
02-19	**Storing objects**	**0:28:12**	Christerpsomby
	Spillprog bachelor	14:39-15:08	
02-19	**Sprint meeting**	**2:03:45**	Christerpsomby
	Spillprog bachelor	16:00-18:03	
02-19	**Meeting**	**2:03:40**	Kristoffer Eidså
	Spillprog bachelor	16:00-18:04	
02-19	**Meeting**	**2:03:27**	Jonas Reitan
	Spillprog bachelor	16:00-18:04	
02-19	**weekly meeting**	**1:57:15**	Henningel
	Spillprog bachelor	16:07-18:04	
02-19	**improving level generation**	**0:49:01**	Henningel
	Spillprog bachelor	18:04-18:53	
02-19	**Member function Pointer**	**0:02:30**	Christerpsomby
	Spillprog bachelor	18:43-18:46	
02-19	**Member function Pointer**	**2:34:00**	Christerpsomby
	Spillprog bachelor	19:59-22:33	
02-20	**General object storage**	**1:26:03**	Christerpsomby
	Spillprog bachelor	08:40-10:06	
02-20	**improving level generation**	**3:12:27**	Henningel
	Spillprog bachelor	10:17-13:30	
02-20	**General object storage**	**0:50:12**	Christerpsomby
	Spillprog bachelor	10:34-11:25	
02-20	**General object storage**	**0:10:10**	Christerpsomby
	Spillprog bachelor	13:16-13:26	
02-20	**General object storage**	**1:18:15**	Christerpsomby
	Spillprog bachelor	13:52-15:11	
02-20	**reevaluating tile-by-tile method**	**2:24:26**	Henningel
	Spillprog bachelor	14:24-16:48	
02-20	**improving level generation**	**0:36:31**	Henningel
	Spillprog bachelor	18:23-18:59	
02-20	**More general object storage**	**0:27:00**	Christerpsomby
	Spillprog bachelor	20:16-20:43	
02-21	**General object storage**	**1:03:23**	Christerpsomby
	Spillprog bachelor	12:26-13:30	
02-21	**improving level generation**	**2:58:15**	Henningel
	Spillprog bachelor	12:40-15:38	
02-21	**Discussing generation techniques**	**2:54:15**	Jonas Reitan
	Spillprog bachelor	13:00-15:55	
02-21	**Implementing user feedback gui**	**1:17:05**	Kristoffer Eidså
	Spillprog bachelor	14:56-16:13	

02-21	**Console**	**1:44:16**	Christerpsomby
	Spillprog bachelor	15:01-16:46	
02-21	**improving level generation**	**4:54:16**	Henningel
	Spillprog bachelor	17:42-22:36	
02-21	**Member function Pointer**	**2:27:04**	Christerpsomby
	Spillprog bachelor	18:20-20:47	
02-21	**Implementing user feedback gui**	**3:00:45**	Kristoffer Eidså
	Spillprog bachelor	18:22-21:23	
02-21	**(no description)**	**0:51:55**	Henningel
	Spillprog bachelor	23:04-23:56	
02-22	**Meeting**	**0:09:11**	Christerpsomby
	Spillprog bachelor	09:00-09:09	
02-22	**Meeting**	**0:10:23**	Kristoffer Eidså
	Spillprog bachelor	09:00-09:11	
02-22	**morning meeting**	**0:07:38**	Henningel
	Spillprog bachelor	09:01-09:09	
02-22	**Implementing user feedback gui**	**1:16:23**	Kristoffer Eidså
	Spillprog bachelor	09:11-10:27	
02-22	**Added/learned libnoise**	**3:41:32**	Jonas Reitan
	Spillprog bachelor	13:06-16:47	
02-22	**Implementing user feedback gui**	**0:58:37**	Kristoffer Eidså
	Spillprog bachelor	15:00-15:59	
02-22	**separating generation into sections and adding tinkering with AI**	**1:25:33**	Henningel
	Spillprog bachelor	16:35-18:00	
02-22	**Implementing user feedback gui**	**1:47:15**	Kristoffer Eidså
	Spillprog bachelor	19:05-20:52	
02-22	**Merge to master, fixing conflicts, support eidså, christer mvp**	**1:16:28**	Jonas Reitan
	Spillprog bachelor	19:40-20:56	
02-22	**Implementing user feedback gui**	**0:48:26**	Kristoffer Eidså
	Spillprog bachelor	21:50-22:39	
02-22	**Console**	**2:04:00**	Christerpsomby
	Spillprog bachelor	21:54-23:58	
02-23	**trying to sort out a few bugs, but the compiler don't agree with me**	**4:13:43**	Jonas Reitan
	Spillprog bachelor	04:40-08:54	
02-23	**Implementing user feedback gui**	**2:29:56**	Kristoffer Eidså
	Spillprog bachelor	06:30-09:00	
02-23	**Meeting**	**0:39:44**	Jonas Reitan
	Spillprog bachelor	08:54-09:33	
02-23	**Meeting**	**0:27:27**	Christerpsomby
	Spillprog bachelor	09:00-09:27	
02-23	**Meeting**	**0:30:12**	Kristoffer Eidså
	Spillprog bachelor	09:00-09:30	

02-23	**morning meeting**	**0:31:54**	Henningel
	Spillprog bachelor	09:02-09:34	
02-23	**Tweaked world generation**	**0:08:34**	Jonas Reitan
	Spillprog bachelor	09:33-09:42	
02-23	**world generation**	**0:23:07**	Henningel
	Spillprog bachelor	09:34-09:57	
02-23	**level generation**	**1:40:47**	Henningel
	Spillprog bachelor	14:55-16:36	
02-23	**Implementing user feedback gui**	**0:32:25**	Kristoffer Eidså
	Spillprog bachelor	15:57-16:30	
02-23	**Implementing user feedback gui**	**1:29:34**	Kristoffer Eidså
	Spillprog bachelor	17:51-19:21	
02-23	**level generation**	**0:30:50**	Henningel
	Spillprog bachelor	17:53-18:24	
02-23	**adding fun enemies**	**2:01:48**	Henningel
	Spillprog bachelor	18:24-20:26	
02-23	**Member function Pointer**	**1:45:00**	Christerpsomby
	Spillprog bachelor	19:20-21:05	
02-23	**meeting**	**1:27:19**	Henningel
	Spillprog bachelor	23:33-01:00	
02-23	**Meeting**	**1:29:22**	Kristoffer Eidså
	Spillprog bachelor	23:34-01:03	
02-23	**Meeting w/tweaking of gui, world and objects**	**1:29:42**	Jonas Reitan
	Spillprog bachelor	23:34-01:04	
02-24	**more enemies**	**1:37:04**	Henningel
	Spillprog bachelor	08:26-10:03	
02-24	**Meeting**	**1:00:16**	Christerpsomby
	Spillprog bachelor	09:00-10:00	
02-24	**Meeting**	**1:03:35**	Jonas Reitan
	Spillprog bachelor	09:00-10:04	
02-24	**more enemies**	**0:08:25**	Henningel
	Spillprog bachelor	10:40-10:48	
02-24	**Meeting - Business**	**1:13:00**	Christerpsomby
	Spillprog bachelor	10:47-12:00	
02-24	**Meeting**	**1:10:54**	Kristoffer Eidså
	Spillprog bachelor	10:49-12:00	
02-24	**meeting with ådne**	**0:50:01**	Henningel
	Spillprog bachelor	11:00-11:50	
02-24	**Meeting at Mustad**	**1:01:45**	Jonas Reitan
	Spillprog bachelor	11:00-12:02	
02-24	**world generation, new project test(didn't work), merge with master**	**2:49:57**	Jonas Reitan
	Spillprog bachelor	13:10-16:00	

02-24	**adding enemy**	**1:10:20**	Henningel
	Spillprog bachelor	14:48-15:58	
02-24	**Implementing user feedback gui**	**0:52:22**	Kristoffer Eidså
	Spillprog bachelor	17:00-17:52	
02-24	**Console**	**4:25:00**	Christerpsomby
	Spillprog bachelor	17:30-21:54	
02-24	**Implementing user feedback gui**	**2:01:51**	Kristoffer Eidså
	Spillprog bachelor	19:54-21:55	
02-25	**enemy stuff**	**1:00:09**	Henningel
	Spillprog bachelor	00:09-01:09	
02-25	**adding pyramid**	**2:21:25**	Henningel
	Spillprog bachelor	01:23-03:45	
02-25	**Discussing spikes, UVs and mesh with Henning, tested cave algorithm**	**4:14:26**	Jonas Reitan
	Spillprog bachelor	02:17-06:32	
02-25	**spike physics**	**1:04:42**	Henningel
	Spillprog bachelor	04:38-05:43	
02-25	**Meeting**	**0:13:08**	Jonas Reitan
	Spillprog bachelor	09:00-09:13	
02-25	**morning meeting**	**0:12:48**	Henningel
	Spillprog bachelor	09:00-09:13	
02-25	**adding enemies and fixing/improving marching squares**	**1:58:49**	Henningel
	Spillprog bachelor	09:14-11:12	
02-25	**Discussing w/ Henning**	**0:18:08**	Jonas Reitan
	Spillprog bachelor	09:14-09:32	
02-25	**making userfeedback work with database**	**1:12:53**	Kristoffer Eidså
	Spillprog bachelor	11:02-12:14	
02-25	**more marching squares improvements**	**1:16:21**	Henningel
	Spillprog bachelor	11:53-13:10	
02-25	**making userfeedback work with database**	**1:47:16**	Kristoffer Eidså
	Spillprog bachelor	12:16-14:03	
02-25	**Helping Eidså**	**0:32:30**	Christerpsomby
	Spillprog bachelor	12:20-12:52	
02-25	**more marching squares improvements**	**0:08:20**	Henningel
	Spillprog bachelor	14:26-14:34	
02-25	**more marching squares improvements**	**1:40:26**	Henningel
	Spillprog bachelor	14:49-16:30	
02-25	**making userfeedback work with database**	**2:02:32**	Kristoffer Eidså
	Spillprog bachelor	16:26-18:29	
02-25	**Console**	**4:03:46**	Christerpsomby
	Spillprog bachelor	20:50-00:54	
02-26	**morning meeting**	**0:17:05**	Henningel
	Spillprog bachelor	09:00-09:17	

02-26	**Meeting**	**0:18:09**	Kristoffer Eidså
	Spillprog bachelor	09:00-09:19	
02-26	**making userfeedback work with database**	**3:25:29**	Kristoffer Eidså
	Spillprog bachelor	09:19-12:44	
02-26	**Reading on software models**	**0:37:11**	Christerpsomby
	Spillprog bachelor	12:08-12:46	
02-26	**adding 45 degree slopes**	**2:58:23**	Henningel
	Spillprog bachelor	12:48-15:46	
02-26	**making userfeedback work with database**	**0:34:51**	Kristoffer Eidså
	Spillprog bachelor	13:52-14:27	
02-26	**Reading game loop models**	**1:30:48**	Christerpsomby
	Spillprog bachelor	13:53-15:23	
02-26	**adding 45 degree slopes**	**0:07:38**	Henningel
	Spillprog bachelor	15:49-15:57	
02-26	**Sprint meeting**	**1:29:54**	Christerpsomby
	Spillprog bachelor	15:55-17:25	
02-26	**friday meeting**	**1:25:38**	Henningel
	Spillprog bachelor	16:00-17:26	
02-26	**Meeting**	**1:25:18**	Jonas Reitan
	Spillprog bachelor	16:00-17:26	
02-26	**Meeting**	**1:25:04**	Kristoffer Eidså
	Spillprog bachelor	16:00-17:26	
02-26	**A* and world, discussing with Henning**	**4:16:08**	Jonas Reitan
	Spillprog bachelor	17:27-21:43	
02-26	**Reading game loop models**	**3:30:25**	Christerpsomby
	Spillprog bachelor	19:31-23:01	
02-26	**making userfeedback work with database**	**0:18:42**	Kristoffer Eidså
	Spillprog bachelor	19:33-19:52	
02-26	**consulting**	**1:19:02**	Henningel
	Spillprog bachelor	20:22-21:41	
02-26	**A* and world, discussing with Henning, research**	**2:26:12**	Jonas Reitan
	Spillprog bachelor	21:43-00:09	
02-26	**making userfeedback work with database**	**3:41:28**	Kristoffer Eidså
	Spillprog bachelor	22:00-01:41	
02-26	**Helping Eidså**	**0:35:11**	Christerpsomby
	Spillprog bachelor	23:01-23:36	
02-26	**Helping Eidså**	**1:56:25**	Christerpsomby
	Spillprog bachelor	23:45-01:41	
02-27	**Level Design Learning and looking at platforming A***	**3:21:19**	Jonas Reitan
	Spillprog bachelor	04:02-07:23	
02-27	**adding custom component types**	**1:14:46**	Henningel
	Spillprog bachelor	09:18-10:33	

02-27	**running diagnostic tests**	**0:30:19**	Henningel
	Spillprog bachelor	10:33-11:04	
02-27	**replace includes with forward declarations**	**2:07:50**	Henningel
	Spillprog bachelor	11:04-13:12	
02-27	**A* in World Implementation and theorycrafting**	**2:23:00**	Jonas Reitan
	Spillprog bachelor	13:32-15:55	
02-27	**making userfeedback work with database**	**1:37:00**	Kristoffer Eidså
	Spillprog bachelor	15:35-17:12	
02-27	**hardcore merge**	**1:16:45**	Henningel
	Spillprog bachelor	16:01-17:18	
02-27	**Read Variadic Templating**	**4:57:49**	Christerpsomby
	Spillprog bachelor	17:42-22:39	
02-28	**Reading on Vulcan API**	**5:18:54**	Christerpsomby
	Spillprog bachelor	14:41-20:00	
02-28	**Database stuff**	**3:07:09**	Kristoffer Eidså
	Spillprog bachelor	19:32-22:39	
02-28	**hardcore merge**	**1:23:08**	Henningel
	Spillprog bachelor	21:45-23:08	
02-28	**fixing memory bug**	**0:53:50**	Henningel
	Spillprog bachelor	23:08-00:02	
02-29	**Meeting**	**0:16:49**	Jonas Reitan
	Spillprog bachelor	08:54-09:11	
02-29	**morning meeting**	**0:09:20**	Henningel
	Spillprog bachelor	09:00-09:09	
02-29	**Flashing stuff and figuring out stuff, bugfixing, shader stuff**	**4:22:29**	Jonas Reitan
	Spillprog bachelor	09:11-13:34	
02-29	**fixing memory bug**	**2:23:10**	Henningel
	Spillprog bachelor	09:26-11:49	
02-29	**Database stuff**	**4:01:19**	Kristoffer Eidså
	Spillprog bachelor	13:58-17:59	
02-29	**Helping Eidså**	**1:21:42**	Christerpsomby
	Spillprog bachelor	15:00-16:21	
02-29	**world generation**	**0:29:33**	Henningel
	Spillprog bachelor	16:25-16:55	
02-29	**world generation**	**0:17:13**	Henningel
	Spillprog bachelor	18:54-19:11	
02-29	**researching and implementing FSM for actors**	**2:57:53**	Henningel
	Spillprog bachelor	19:11-22:09	
02-29	**making userfeedback work with database**	**2:07:18**	Kristoffer Eidså
	Spillprog bachelor	20:06-22:13	
02-29	**researching and implementing FSM for actors**	**0:07:44**	Henningel
	Spillprog bachelor	23:53-00:01	

03-01	**Tweaking A* to work with physics, grid representation of world**	**2:42:26**	Jonas Reitan
	Spillprog bachelor	09:11-11:53	
03-01	**researching and implementing FSM for actors**	**2:04:09**	Henningel
	Spillprog bachelor	11:13-13:17	
03-01	**Database stuff**	**1:54:30**	Kristoffer Eidså
	Spillprog bachelor	11:32-13:26	
03-01	**Tweaking A* to work with physics, grid representation of world**	**1:37:44**	Jonas Reitan
	Spillprog bachelor	12:09-13:46	
03-01	**Database stuff**	**3:02:38**	Kristoffer Eidså
	Spillprog bachelor	14:17-17:19	
03-01	**researching and implementing FSM for actors**	**1:36:33**	Henningel
	Spillprog bachelor	14:30-16:07	
03-01	**Added sql and boost**	**0:16:34**	Jonas Reitan
	Spillprog bachelor	14:53-15:09	
03-01	**Database stuff**	**1:09:14**	Kristoffer Eidså
	Spillprog bachelor	18:12-19:21	
03-01	**Reading game loop models**	**1:50:52**	Christerpsomby
	Spillprog bachelor	18:24-20:14	
03-01	**More A*, grid, pushed some stuff, still some stuff to do**	**0:46:00**	Jonas Reitan
	Spillprog bachelor	18:42-19:28	
03-01	**Read Variadic Templating**	**0:51:01**	Christerpsomby
	Spillprog bachelor	21:44-22:35	
03-01	**researching and implementing FSM for actors**	**1:23:11**	Henningel
	Spillprog bachelor	23:19-00:42	
03-02	**morning meeting**	**0:17:04**	Henningel
	Spillprog bachelor	09:00-09:17	
03-02	**Meeting**	**0:16:46**	Kristoffer Eidså
	Spillprog bachelor	09:00-09:17	
03-02	**Meeting**	**0:02:00**	Jonas Reitan
	Spillprog bachelor	09:15-09:17	
03-02	**Trying to solve a weird bug in grid representation of world, need rework to solve issues with world being in negative y, tested userfeedback**	**4:38:15**	Jonas Reitan
	Spillprog bachelor	09:17-13:55	
03-02	**Database stuff**	**1:29:25**	Kristoffer Eidså
	Spillprog bachelor	09:17-10:47	
03-02	**researching and implementing FSM for actors**	**0:18:04**	Henningel
	Spillprog bachelor	10:23-10:41	
03-02	**Database stuff**	**2:29:02**	Kristoffer Eidså
	Spillprog bachelor	11:14-13:43	
03-02	**Console**	**4:27:02**	Christerpsomby
	Spillprog bachelor	13:21-17:48	

03-02	**Database stuff**	**1:25:37**	Kristoffer Eidså
	Spillprog bachelor	15:50-17:16	
03-02	**researching and implementing FSM for actors**	**3:37:20**	Henningel
	Spillprog bachelor	18:38-22:15	
03-02	**Console**	**2:07:00**	Christerpsomby
	Spillprog bachelor	19:53-22:00	
03-03	**Meeting**	**0:05:59**	Jonas Reitan
	Spillprog bachelor	09:00-09:05	
03-03	**morning meeting**	**0:05:24**	Henningel
	Spillprog bachelor	09:00-09:06	
03-03	**Looked at optimization algorithms (heap, simplifying paths, etc.), a* improvements (weights) and other things that might help us**	**5:48:40**	Jonas Reitan
	Spillprog bachelor	09:18-15:07	
03-03	**Database stuff**	**3:03:42**	Kristoffer Eidså
	Spillprog bachelor	10:30-13:34	
03-03	**Console**	**7:06:32**	Christerpsomby
	Spillprog bachelor	11:05-18:12	
03-03	**researching and implementing FSM for actors**	**1:27:41**	Henningel
	Spillprog bachelor	11:08-12:36	
03-03	**Database stuff**	**1:00:36**	Kristoffer Eidså
	Spillprog bachelor	13:56-14:57	
03-03	**researching and implementing FSM for actors**	**1:15:33**	Henningel
	Spillprog bachelor	16:56-18:11	
03-03	**Database stuff**	**1:26:13**	Kristoffer Eidså
	Spillprog bachelor	17:03-18:29	
03-03	**Database stuff**	**0:36:48**	Kristoffer Eidså
	Spillprog bachelor	21:53-22:29	
03-03	**Fixed some bugs(Actually compiles now), added goal to world generation and fixed some errors in various actor xmls**	**1:47:07**	Jonas Reitan
	Spillprog bachelor	22:23-00:10	
03-03	**Helping Jonas**	**0:10:47**	Christerpsomby
	Spillprog bachelor	22:46-22:56	
03-03	**consulting**	**0:56:35**	Henningel
	Spillprog bachelor	22:52-23:48	
03-03	**researching and implementing FSM for actors**	**0:59:32**	Henningel
	Spillprog bachelor	23:48-00:48	
03-04	**researching and implementing FSM for actors**	**1:31:27**	Henningel
	Spillprog bachelor	01:19-02:51	
03-04	**Morning meeting**	**0:19:00**	Jonas Reitan
	Spillprog bachelor	09:00-09:19	
03-04	**morning meeting**	**0:17:35**	Henningel
	Spillprog bachelor	09:00-09:18	
03-04	**Fixed bug where goal would not spawn, compile time benchmarking, testing multithreading, testing various ideas(need feedback?)**	**6:04:59**	Jonas Reitan
	Spillprog bachelor	09:20-15:25	

03-04	**Database stuff** Spillprog bachelor	**4:59:29** 11:00-16:00	Kristoffer Eidså
03-04	**fixing some animation and compile performance by rearranging includes** Spillprog bachelor	**2:34:10** 11:22-13:56	Henningel
03-04	**Performance testing with henning** Spillprog bachelor	**0:38:00** 13:02-13:40	Christerpsomby
03-04	**Console** Spillprog bachelor	**2:21:35** 13:40-16:02	Christerpsomby
03-04	**world generation** Spillprog bachelor	**0:45:16** 13:56-14:41	Henningel
03-04	**Meeting** Spillprog bachelor	**0:30:21** 16:00-16:30	Kristoffer Eidså
03-04	**Meeting** Spillprog bachelor	**0:28:29** 16:00-16:29	Jonas Reitan
03-04	**scrum meeting** Spillprog bachelor	**0:28:02** 16:00-16:28	Henningel
03-04	**Sprint meeting** Spillprog bachelor	**0:36:00** 16:02-16:38	Christerpsomby
03-04	**Console** Spillprog bachelor	**4:36:59** 17:22-21:59	Christerpsomby
03-05	**Console** Spillprog bachelor	**0:06:53** 14:34-14:41	Christerpsomby
03-05	**Fill gap algorithm in world generation, tested spike/moving platform stuff, tried adding damage on spikes, removed some unused crap** Spillprog bachelor	**5:44:00** 22:05-03:49	Jonas Reitan
03-05	**creating enemies** Spillprog bachelor	**1:55:51** 22:15-00:11	Henningel
03-05	**Console** Spillprog bachelor	**0:41:34** 22:48-23:30	Christerpsomby
03-06	**Console** Spillprog bachelor	**3:20:23** 02:17-05:38	Christerpsomby
03-06	**bug fixing and improvements** Spillprog bachelor	**2:39:10** 11:46-14:26	Henningel
03-06	**finished up some stuff, did some jira thingy and started adding a spring game mechanic** Spillprog bachelor	**3:24:10** 13:29-16:53	Kristoffer Eidså
03-06	**Console** Spillprog bachelor	**3:43:32** 14:31-18:14	Christerpsomby
03-06	**world generation** Spillprog bachelor	**1:46:39** 15:40-17:26	Henningel
03-06	**implementing spring** Spillprog bachelor	**1:36:48** 19:40-21:17	Kristoffer Eidså
03-06	**world generation** Spillprog bachelor	**3:12:34** 20:14-23:27	Henningel

03-06	**Console** Spillprog bachelor	**1:04:40** 20:14-21:19	Christerpsomby
03-06	**implementing spring** Spillprog bachelor	**0:29:05** 21:17-21:46	Kristoffer Eidså
03-07	**Console** Spillprog bachelor	**6:29:52** 03:13-09:42	Christerpsomby
03-07	**Meeting** Spillprog bachelor	**0:08:31** 09:00-09:08	Kristoffer Eidså
03-07	**morning meeting** Spillprog bachelor	**0:08:36** 09:00-09:09	Henningel
03-07	**Morning meeting** Spillprog bachelor	**0:08:18** 09:00-09:09	Jonas Reitan
03-07	**Made world generator a bit smarter, still much to do. Read up about algorithms to solve some problems.** Spillprog bachelor	**6:57:31** 09:09-16:06	Jonas Reitan
03-07	**implementing block that tricks ur tiny human mind** Spillprog bachelor	**3:21:36** 09:09-12:30	Kristoffer Eidså
03-07	**world generation** Spillprog bachelor	**1:58:33** 11:48-13:47	Henningel
03-07	**Lambda experimentation** Spillprog bachelor	**1:02:32** 13:23-14:26	Christerpsomby
03-07	**implementing block and fixing contact listener** Spillprog bachelor	**4:02:44** 16:30-20:32	Kristoffer Eidså
03-07	**renaming the project** Spillprog bachelor	**0:38:23** 17:00-17:38	Henningel
03-07	**Tested some stuff for Henning** Spillprog bachelor	**0:39:55** 17:06-17:45	Jonas Reitan
03-07	**consulting** Spillprog bachelor	**1:44:22** 19:07-20:51	Henningel
03-07	**debugging database** Spillprog bachelor	**0:51:40** 20:32-21:23	Kristoffer Eidså
03-08	**Mem_fn storage** Spillprog bachelor	**3:06:20** 00:23-03:29	Christerpsomby
03-08	**Console** Spillprog bachelor	**0:36:04** 07:01-07:38	Christerpsomby
03-08	**Console** Spillprog bachelor	**0:32:05** 08:28-09:00	Christerpsomby
03-08	**Meeting** Spillprog bachelor	**0:15:06** 09:00-09:15	Christerpsomby
03-08	**morning meeting** Spillprog bachelor	**0:13:11** 09:02-09:15	Henningel
03-08	**Fixed some bugs in world generator, read about a* edge placement, discussed some stuff with Henning, network debugging** Spillprog bachelor	**7:34:13** 09:05-16:40	Jonas Reitan

03-08	**Looked more into fixing database bug** Spillprog bachelor	**2:03:26** 09:22-11:26	Kristoffer Eidså
03-08	**world generation** Spillprog bachelor	**2:02:45** 10:29-12:32	Henningel
03-08	**pressing f5 on confluence** Spillprog bachelor	**1:29:16** 11:26-12:55	Kristoffer Eidså
03-08	**tried to fix database not working in debug. Gave up and did a workaround that works fine.** Spillprog bachelor	**1:09:16** 15:26-16:36	Kristoffer Eidså
03-08	**Helping Eidså** Spillprog bachelor	**0:14:00** 16:18-16:32	Christerpsomby
03-08	**figuring out how the custom state system works and looked up box2d stuff** Spillprog bachelor	**1:26:42** 16:36-18:02	Kristoffer Eidså
03-08	**Writing about database, Henning crashed confluence so writing in notepad or something** Spillprog bachelor	**1:18:24** 18:02-19:21	Kristoffer Eidså
03-08	**world generation** Spillprog bachelor	**1:21:55** 22:39-00:01	Henningel
03-08	**Console** Spillprog bachelor	**1:04:00** 23:33-00:37	Christerpsomby
03-09	**Console** Spillprog bachelor	**2:34:47** 01:23-03:57	Christerpsomby
03-09	**morning meeting** Spillprog bachelor	**0:05:02** 09:00-09:05	Henningel
03-09	**Morning meeting** Spillprog bachelor	**0:04:53** 09:00-09:05	Jonas Reitan
03-09	**Meeting** Spillprog bachelor	**0:04:06** 09:00-09:04	Kristoffer Eidså
03-09	**Writing in my document, worked with some box2d stuff that didn't work for my purpose** Spillprog bachelor	**3:15:13** 09:04-12:19	Kristoffer Eidså
03-09	**Researched stuff regarding writing a bachelor thesis, discussed/meeting, tried to fix a few bugs, looked at a* stuff** Spillprog bachelor	**7:26:16** 09:05-16:31	Jonas Reitan
03-09	**world generation** Spillprog bachelor	**2:12:34** 10:25-12:37	Henningel
03-09	**writing about gui, user feedback and databases** Spillprog bachelor	**3:10:50** 12:20-15:31	Kristoffer Eidså
03-09	**document writing** Spillprog bachelor	**2:00:49** 14:04-16:05	Henningel
03-09	**Console** Spillprog bachelor	**3:11:37** 16:04-19:16	Christerpsomby
03-09	**Console** Spillprog bachelor	**0:46:41** 21:14-22:00	Christerpsomby

03-10	**Added support for billboarding** Spillprog bachelor	**0:51:31** 04:01-04:52	Jonas Reitan
03-10	**writing about userfeedback and tweaking user feedback stuff** Spillprog bachelor	**5:10:26** 10:00-15:10	Kristoffer Eidså
03-10	**document writing** Spillprog bachelor	**2:54:51** 12:56-15:51	Henningel
03-10	**Console** Spillprog bachelor	**0:26:13** 15:33-15:59	Christerpsomby
03-10	**Sprint meeting** Spillprog bachelor	**0:54:07** 15:59-16:53	Christerpsomby
03-10	**Meeting** Spillprog bachelor	**0:54:18** 15:59-16:54	Kristoffer Eidså
03-10	**Meeting** Spillprog bachelor	**0:54:29** 16:00-16:54	Jonas Reitan
03-10	**scrum meeting** Spillprog bachelor	**0:54:05** 16:00-16:54	Henningel
03-10	**document writing** Spillprog bachelor	**0:46:32** 17:49-18:36	Henningel
03-10	**Console** Spillprog bachelor	**1:49:00** 18:15-20:04	Christerpsomby
03-10	**Reading AI stuff** Spillprog bachelor	**2:05:13** 20:56-23:01	Christerpsomby
03-10	**document writing** Spillprog bachelor	**3:58:10** 21:00-00:58	Henningel
03-10	**Wrote about noise algorithm and the challenges of finding the correct one** Spillprog bachelor	**3:16:10** 23:12-02:28	Jonas Reitan
03-11	**Console** Spillprog bachelor	**1:04:09** 04:24-05:28	Christerpsomby
03-11	**document writing** Spillprog bachelor	**2:34:50** 09:42-12:17	Henningel
03-11	**Writing stuff about database** Spillprog bachelor	**1:28:12** 10:41-12:09	Kristoffer Eidså
03-11	**World generation** Spillprog bachelor	**2:29:50** 12:09-14:39	Kristoffer Eidså
03-11	**document writing** Spillprog bachelor	**2:16:23** 12:44-15:00	Henningel
03-11	**Console** Spillprog bachelor	**0:53:00** 14:07-15:00	Christerpsomby
03-11	**Console** Spillprog bachelor	**0:15:34** 18:49-19:05	Christerpsomby
03-12	**document writing** Spillprog bachelor	**0:32:58** 10:38-11:10	Henningel

03-12	**world generation**	**1:17:43**	Henningel
	Spillprog bachelor	11:11-12:28	
03-12	**world generation**	**0:05:04**	Henningel
	Spillprog bachelor	14:26-14:31	
03-12	**Lan Console work**	**1:00:00**	Christerpsomby
	Spillprog bachelor	16:25-17:25	
03-12	**world generation**	**0:28:10**	Henningel
	Spillprog bachelor	19:42-20:10	
03-12	**consulting**	**0:47:00**	Jonas Reitan
	Spillprog bachelor	20:54-21:41	
03-12	**world generation**	**0:47:16**	Henningel
	Spillprog bachelor	20:54-21:41	
03-13	**Console**	**1:20:07**	Christerpsomby
	Spillprog bachelor	03:05-04:25	
03-13	**Wrote about noise at confluence, tested multi layered world generation**	**2:01:00**	Jonas Reitan
	Spillprog bachelor	03:13-05:14	
03-13	**world generation with layers and experimenting with stuff**	**4:20:42**	Jonas Reitan
	Spillprog bachelor	12:09-16:30	
03-13	**Reading AI stuff**	**1:41:10**	Christerpsomby
	Spillprog bachelor	14:03-15:44	
03-13	**World generation**	**2:37:42**	Kristoffer Eidså
	Spillprog bachelor	15:44-18:22	
03-13	**Console**	**0:12:33**	Christerpsomby
	Spillprog bachelor	15:45-15:57	
03-13	**Console**	**0:53:23**	Christerpsomby
	Spillprog bachelor	16:44-17:37	
03-13	**world generation**	**3:11:20**	Henningel
	Spillprog bachelor	17:33-20:44	
03-13	**Reading about Machine learning**	**2:24:45**	Christerpsomby
	Spillprog bachelor	17:38-20:02	
03-13	**Reading about MDP**	**0:13:10**	Christerpsomby
	Spillprog bachelor	20:02-20:16	
03-13	**world generation**	**2:55:13**	Henningel
	Spillprog bachelor	21:05-00:01	
03-14	**World generation**	**5:03:33**	Kristoffer Eidså
	Spillprog bachelor	12:09-17:12	
03-14	**Console**	**1:22:55**	Christerpsomby
	Spillprog bachelor	13:13-14:36	
03-14	**World Generation testing to see what might work or not**	**5:08:12**	Jonas Reitan
	Spillprog bachelor	14:35-19:44	
03-14	**Console**	**1:18:11**	Christerpsomby
	Spillprog bachelor	15:41-16:59	

03-15	**School thing**	**4:37:00**	Christerpsomby
	Spillprog bachelor	09:00-13:37	
03-15	**went to school**	**5:00:00**	Kristoffer Eidså
	Spillprog bachelor	09:00-14:00	
03-15	**Discussed stuff as a group. World generation, bugfix, tweaks.**	**5:00:00**	Jonas Reitan
	Spillprog bachelor	09:00-14:00	
03-15	**discussing AI in gamelab**	**4:14:25**	Henningel
	Spillprog bachelor	09:30-13:45	
03-16	**Working on bus**	**1:04:00**	Christerpsomby
	Spillprog bachelor	04:30-05:34	
03-16	**Working in airport**	**1:19:00**	Christerpsomby
	Spillprog bachelor	06:01-07:20	
03-16	**Working in the air**	**1:25:00**	Christerpsomby
	Spillprog bachelor	08:15-09:40	
03-16	**Morning meeting**	**0:08:49**	Jonas Reitan
	Spillprog bachelor	09:00-09:08	
03-16	**morning meeting**	**0:08:09**	Henningel
	Spillprog bachelor	09:00-09:08	
03-16	**did some ui stuff and world gen**	**5:03:51**	Kristoffer Eidså
	Spillprog bachelor	12:00-17:04	
03-16	**world generation**	**2:13:47**	Henningel
	Spillprog bachelor	17:05-19:18	
03-16	**Planned out object generation and looked at ways to implement it**	**4:56:15**	Jonas Reitan
	Spillprog bachelor	20:21-01:17	
03-17	**Level Generation**	**3:59:03**	Christerpsomby
	Spillprog bachelor	08:06-12:05	
03-17	**(no description)**	**0:07:42**	Henningel
	Spillprog bachelor	09:01-09:08	
03-17	**World generation**	**2:17:19**	Kristoffer Eidså
	Spillprog bachelor	09:33-11:50	
03-17	**world generation**	**1:04:57**	Henningel
	Spillprog bachelor	10:58-12:03	
03-17	**World generation**	**3:08:51**	Kristoffer Eidså
	Spillprog bachelor	13:00-16:09	
03-17	**Level Generation**	**2:58:55**	Christerpsomby
	Spillprog bachelor	14:33-17:32	
03-17	**world generation**	**0:51:22**	Henningel
	Spillprog bachelor	15:41-16:33	
03-17	**world generation**	**0:36:01**	Henningel
	Spillprog bachelor	17:04-17:40	
03-17	**world generation**	**1:10:06**	Henningel
	Spillprog bachelor	19:40-20:50	

03-17	**World generation testing, tweak the grid for A* and object spawning algorithm**	**5:15:04**	Jonas Reitan
	Spillprog bachelor	20:11-01:26	
03-17	**world generation**	**0:26:04**	Henningel
	Spillprog bachelor	21:10-21:37	
03-18	**world generation**	**1:54:26**	Henningel
	Spillprog bachelor	10:05-12:00	
03-18	**world generation**	**1:28:10**	Henningel
	Spillprog bachelor	12:50-14:19	
03-18	**Level Generation**	**1:16:15**	Christerpsomby
	Spillprog bachelor	14:06-15:22	
03-18	**world generation**	**2:59:21**	Henningel
	Spillprog bachelor	17:00-20:00	
03-18	**Level Generation**	**3:16:54**	Christerpsomby
	Spillprog bachelor	23:35-02:51	
03-19	**Level Generation**	**0:57:06**	Christerpsomby
	Spillprog bachelor	04:33-05:30	
03-19	**world gen and hitting head against a wall**	**7:18:00**	Kristoffer Eidså
	Spillprog bachelor	09:34-16:51	
03-19	**read about stl, mlc, some tweaks to stuff**	**5:18:01**	Jonas Reitan
	Spillprog bachelor	11:02-16:20	
03-19	**world generation**	**4:34:18**	Henningel
	Spillprog bachelor	13:58-18:32	
03-19	**Level Generation**	**1:26:19**	Christerpsomby
	Spillprog bachelor	14:04-15:31	
03-19	**Level Generation**	**4:02:47**	Christerpsomby
	Spillprog bachelor	16:31-20:33	
03-19	**tested some more world generation stuff**	**1:46:21**	Jonas Reitan
	Spillprog bachelor	17:36-19:22	
03-19	**world generation**	**2:49:21**	Henningel
	Spillprog bachelor	19:15-22:05	
03-20	**Level Generation**	**1:48:59**	Christerpsomby
	Spillprog bachelor	02:26-04:15	
03-20	**world generation**	**3:24:28**	Henningel
	Spillprog bachelor	10:33-13:58	
03-20	**researched good level design, world generation, group discussion on skype**	**7:29:00**	Jonas Reitan
	Spillprog bachelor	11:30-18:59	
03-20	**World generation**	**2:10:00**	Kristoffer Eidså
	Spillprog bachelor	11:42-13:52	
03-20	**Level Generation**	**1:20:50**	Christerpsomby
	Spillprog bachelor	14:03-15:23	
03-20	**Level Generation**	**0:27:38**	Christerpsomby
	Spillprog bachelor	16:09-16:37	

03-21	**database, ui and world generation**	**6:06:40**	Kristoffer Eidså
	Spillprog bachelor	08:27-14:33	
03-22	**World generation**	**3:33:34**	Kristoffer Eidså
	Spillprog bachelor	10:03-13:36	
03-22	**added timedblock to the world generation, but nothing is spawned yet**	**0:37:52**	Jonas Reitan
	Spillprog bachelor	14:48-15:25	
03-22	**world generation**	**2:18:34**	Henningel
	Spillprog bachelor	20:52-23:11	
03-23	**world generating and database stuff**	**4:53:25**	Kristoffer Eidså
	Spillprog bachelor	10:12-15:05	
03-23	**database**	**1:27:56**	Henningel
	Spillprog bachelor	13:38-15:06	
03-23	**world generation theories, research and planning**	**5:15:07**	Jonas Reitan
	Spillprog bachelor	20:34-01:49	
03-24	**Level Generation**	**3:08:38**	Christerpsomby
	Spillprog bachelor	04:20-07:28	
03-24	**trying to understand database and stuff**	**3:29:22**	Kristoffer Eidså
	Spillprog bachelor	10:21-13:50	
03-24	**trying to understand database and stuff**	**2:00:24**	Kristoffer Eidså
	Spillprog bachelor	14:15-16:15	
03-25	**Console**	**4:21:04**	Christerpsomby
	Spillprog bachelor	01:22-05:43	
03-25	**world generation**	**3:09:51**	Henningel
	Spillprog bachelor	11:41-14:51	
03-25	**level design research and looking at various themes we might want to implement**	**1:42:35**	Jonas Reitan
	Spillprog bachelor	15:43-17:25	
03-25	**Level Generation**	**2:05:22**	Christerpsomby
	Spillprog bachelor	17:33-19:39	
03-25	**level design discussing and how to do it**	**1:43:00**	Jonas Reitan
	Spillprog bachelor	18:21-20:04	
03-25	**world generation**	**0:08:38**	Henningel
	Spillprog bachelor	21:16-21:25	
03-26	**Level Generation**	**2:22:26**	Christerpsomby
	Spillprog bachelor	00:36-02:59	
03-26	**world generation**	**0:43:06**	Henningel
	Spillprog bachelor	01:16-01:59	
03-26	**Level Generation**	**1:06:56**	Christerpsomby
	Spillprog bachelor	05:26-06:32	
03-26	**world generation**	**1:06:34**	Henningel
	Spillprog bachelor	11:09-12:15	
03-26	**world generation**	**0:29:48**	Henningel
	Spillprog bachelor	17:22-17:52	

03-26	**Level Generation**	**1:59:14**	Christerpsomby
	Spillprog bachelor	17:38-19:37	
03-27	**box2d research for "gravity zones" and apparently over complicating stuff**	**1:59:32**	Jonas Reitan
	Spillprog bachelor	01:01-04:00	
03-27	**Level Generation**	**1:02:47**	Christerpsomby
	Spillprog bachelor	01:36-03:38	
03-27	**level design for world generation and looked at box2d documentation to solve my problems**	**3:54:14**	Jonas Reitan
	Spillprog bachelor	11:03-14:57	
03-27	**world generation**	**1:32:58**	Henningel
	Spillprog bachelor	14:06-15:39	
03-27	**Level Generation**	**2:32:38**	Christerpsomby
	Spillprog bachelor	16:31-19:03	
03-27	**world generation**	**1:25:12**	Henningel
	Spillprog bachelor	18:26-19:51	
03-27	**world generation**	**1:31:42**	Henningel
	Spillprog bachelor	20:49-22:21	
03-27	**world generation**	**1:13:20**	Henningel
	Spillprog bachelor	22:47-00:00	
03-28	**Level Generation**	**1:54:51**	Christerpsomby
	Spillprog bachelor	02:58-04:53	
03-29	**Level Generation**	**2:10:24**	Christerpsomby
	Spillprog bachelor	00:43-02:53	
03-29	**Level Generation**	**2:45:00**	Christerpsomby
	Spillprog bachelor	13:59-16:44	
03-29	**world/level generation**	**4:54:56**	Jonas Reitan
	Spillprog bachelor	22:13-03:07	
03-30	**Level Generation**	**1:30:00**	Christerpsomby
	Spillprog bachelor	08:00-09:30	
03-30	**Level Generation**	**1:30:00**	Christerpsomby
	Spillprog bachelor	10:00-11:30	
03-30	**World generation**	**3:54:49**	Kristoffer Eidså
	Spillprog bachelor	10:01-13:55	
03-30	**World generation**	**2:08:41**	Kristoffer Eidså
	Spillprog bachelor	15:15-17:24	
03-30	**researched NEAT (cgNEAT). World/level generation.**	**5:50:35**	Jonas Reitan
	Spillprog bachelor	18:16-00:07	
03-31	**World generation**	**7:46:10**	Kristoffer Eidså
	Spillprog bachelor	09:42-17:29	
03-31	**Level Generation**	**4:12:48**	Christerpsomby
	Spillprog bachelor	10:02-14:15	
03-31	**world generation**	**1:45:02**	Henningel
	Spillprog bachelor	10:05-11:50	

03-31	**world/level generation**	**2:49:00**	Jonas Reitan
	Spillprog bachelor	13:14-16:03	
03-31	**world generation**	**1:54:47**	Henningel
	Spillprog bachelor	16:27-18:21	
03-31	**world/level generation**	**3:35:00**	Jonas Reitan
	Spillprog bachelor	17:01-20:36	
04-01	**world/level generation/research and testing some stuff**	**5:52:39**	Jonas Reitan
	Spillprog bachelor	04:13-10:05	
04-01	**Level Generation**	**0:55:37**	Christerpsomby
	Spillprog bachelor	12:05-13:01	
04-01	**World generation**	**2:25:20**	Kristoffer Eidså
	Spillprog bachelor	12:10-14:35	
04-01	**world generation**	**1:13:56**	Henningel
	Spillprog bachelor	14:52-16:06	
04-01	**Meeting**	**0:38:01**	Jonas Reitan
	Spillprog bachelor	16:00-16:38	
04-01	**Spring Meeting**	**0:38:01**	Christerpsomby
	Spillprog bachelor	16:00-16:38	
04-01	**Meeting**	**0:37:16**	Kristoffer Eidså
	Spillprog bachelor	16:00-16:37	
04-01	**friday meeting**	**0:40:57**	Henningel
	Spillprog bachelor	16:01-16:42	
04-01	**world generation**	**2:56:03**	Henningel
	Spillprog bachelor	17:39-20:35	
04-01	**world generating and database stuff**	**3:04:01**	Kristoffer Eidså
	Spillprog bachelor	17:49-20:53	
04-01	**Attempting to install VM**	**3:25:44**	Christerpsomby
	Spillprog bachelor	18:02-21:27	
04-01	**world generation**	**0:35:05**	Henningel
	Spillprog bachelor	21:22-21:57	
04-01	**world generation**	**3:57:23**	Henningel
	Spillprog bachelor	23:06-03:04	
04-02	**read about reverse design on Mario to learn gradually increasing difficulty in level design**	**3:23:20**	Jonas Reitan
	Spillprog bachelor	03:47-07:10	
04-02	**world generation**	**5:56:55**	Henningel
	Spillprog bachelor	09:16-15:12	
04-02	**World generation**	**1:35:40**	Kristoffer Eidså
	Spillprog bachelor	11:07-12:42	
04-02	**World generation**	**3:42:15**	Kristoffer Eidså
	Spillprog bachelor	12:42-16:24	
04-02	**world generation**	**1:54:13**	Henningel
	Spillprog bachelor	15:33-17:28	

04-02	**Level Generation**	**6:52:53**	Christerpsomby
	Spillprog bachelor	16:35-23:27	
04-02	**world/level generation**	**3:01:03**	Jonas Reitan
	Spillprog bachelor	18:30-21:31	
04-02	**Installer**	**0:10:00**	Christerpsomby
	Spillprog bachelor	23:28-23:38	
04-02	**Installer**	**3:57:44**	Christerpsomby
	Spillprog bachelor	23:58-03:56	
04-03	**World generation**	**4:59:16**	Kristoffer Eidså
	Spillprog bachelor	12:02-17:01	
04-03	**world generation**	**6:30:58**	Henningel
	Spillprog bachelor	13:16-19:47	
04-03	**world generation testing**	**1:14:05**	Jonas Reitan
	Spillprog bachelor	16:03-17:17	
04-03	**Installer testing**	**5:10:49**	Christerpsomby
	Spillprog bachelor	17:34-22:45	
04-03	**designing chunk ideas, world generation and consulting**	**3:48:24**	Jonas Reitan
	Spillprog bachelor	18:35-22:23	
04-03	**World generation**	**5:03:50**	Kristoffer Eidså
	Spillprog bachelor	18:52-23:56	
04-03	**world generation**	**1:46:54**	Henningel
	Spillprog bachelor	21:33-23:19	
04-03	**world generation**	**6:10:52**	Henningel
	Spillprog bachelor	23:52-06:03	
04-04	**World generation**	**3:00:02**	Kristoffer Eidså
	Spillprog bachelor	11:48-14:48	
04-04	**World generation**	**2:24:46**	Kristoffer Eidså
	Spillprog bachelor	18:19-20:44	
04-05	**Installer**	**2:04:26**	Christerpsomby
	Spillprog bachelor	00:04-02:08	
04-05	**Meeting at school. World generation discussion and ideas**	**7:03:00**	Jonas Reitan
	Spillprog bachelor	09:00-16:03	
04-05	**went to school**	**3:36:34**	Kristoffer Eidså
	Spillprog bachelor	09:00-12:36	
04-05	**School meeting**	**3:36:00**	Christerpsomby
	Spillprog bachelor	09:00-12:36	
04-05	**went to school**	**3:35:28**	Henningel
	Spillprog bachelor	09:00-12:36	
04-05	**World generation**	**1:38:14**	Kristoffer Eidså
	Spillprog bachelor	16:32-18:10	
04-06	**Installer**	**4:08:50**	Christerpsomby
	Spillprog bachelor	02:30-06:38	

04-06	**World generation**	**3:02:31**	Kristoffer Eidså
	Spillprog bachelor	12:03-15:06	
04-06	**world generation**	**6:58:42**	Jonas Reitan
	Spillprog bachelor	12:29-19:27	
04-06	**World generation**	**2:00:54**	Kristoffer Eidså
	Spillprog bachelor	19:33-21:33	
04-06	**world generation**	**2:19:09**	Henningel
	Spillprog bachelor	19:37-21:56	
04-07	**Installer**	**5:08:10**	Christerpsomby
	Spillprog bachelor	01:36-06:44	
04-07	**world generating and database stuff**	**3:10:38**	Kristoffer Eidså
	Spillprog bachelor	12:01-15:11	
04-07	**world generation**	**4:52:42**	Jonas Reitan
	Spillprog bachelor	14:17-19:09	
04-07	**world generating and database stuff**	**2:00:36**	Kristoffer Eidså
	Spillprog bachelor	21:33-23:34	
04-08	**Level Generation**	**3:11:36**	Christerpsomby
	Spillprog bachelor	02:29-05:41	
04-08	**reading about level design**	**1:30:31**	Jonas Reitan
	Spillprog bachelor	07:03-08:33	
04-08	**world generation**	**3:31:42**	Henningel
	Spillprog bachelor	11:04-14:36	
04-08	**world generating and database stuff**	**3:29:46**	Kristoffer Eidså
	Spillprog bachelor	12:17-15:47	
04-08	**Meeting**	**0:46:47**	Kristoffer Eidså
	Spillprog bachelor	16:00-16:47	
04-08	**Spring Meeting**	**0:42:00**	Christerpsomby
	Spillprog bachelor	16:06-16:48	
04-08	**friday meeting**	**0:42:07**	Henningel
	Spillprog bachelor	16:06-16:48	
04-08	**reading and testing world generation stuff**	**5:02:13**	Jonas Reitan
	Spillprog bachelor	18:34-23:36	
04-08	**Installer**	**2:50:32**	Christerpsomby
	Spillprog bachelor	18:46-21:36	
04-08	**world generation**	**0:37:08**	Henningel
	Spillprog bachelor	19:02-19:39	
04-08	**world generation**	**0:49:57**	Henningel
	Spillprog bachelor	20:44-21:34	
04-08	**world generating and database stuff**	**0:47:29**	Kristoffer Eidså
	Spillprog bachelor	20:51-21:39	
04-09	**Installer**	**2:47:36**	Christerpsomby
	Spillprog bachelor	11:41-14:29	

04-09	**world generation**	**3:00:27**	Henningel
	Spillprog bachelor	13:16-16:17	
04-09	**world generating and database stuff**	**3:09:08**	Kristoffer Eidså
	Spillprog bachelor	14:11-17:20	
04-09	**Installer**	**2:53:22**	Christerpsomby
	Spillprog bachelor	15:03-17:57	
04-09	**reading/looking at crossplatform compiling, world generation**	**4:34:07**	Jonas Reitan
	Spillprog bachelor	16:33-21:07	
04-09	**world generation**	**0:54:34**	Henningel
	Spillprog bachelor	17:05-18:00	
04-09	**Installer**	**2:50:57**	Christerpsomby
	Spillprog bachelor	20:24-23:14	
04-09	**world generation**	**4:04:19**	Henningel
	Spillprog bachelor	20:56-01:00	
04-09	**world generating and database stuff**	**2:02:33**	Kristoffer Eidså
	Spillprog bachelor	21:01-23:04	
04-10	**whiteboard sketching and testing stuff**	**2:18:56**	Jonas Reitan
	Spillprog bachelor	07:13-09:31	
04-10	**Installer**	**4:48:28**	Christerpsomby
	Spillprog bachelor	08:40-13:29	
04-10	**world generation**	**5:03:03**	Henningel
	Spillprog bachelor	11:55-16:58	
04-10	**world generating and database stuff**	**2:19:48**	Kristoffer Eidså
	Spillprog bachelor	14:53-17:13	
04-10	**world generating and database stuff**	**1:31:28**	Kristoffer Eidså
	Spillprog bachelor	17:56-19:28	
04-10	**world generation**	**4:21:24**	Henningel
	Spillprog bachelor	18:20-22:42	
04-10	**world generation**	**2:41:42**	Jonas Reitan
	Spillprog bachelor	21:04-23:46	
04-10	**world generation**	**5:54:01**	Henningel
	Spillprog bachelor	23:21-05:15	
04-11	**world generating and database stuff**	**5:01:35**	Kristoffer Eidså
	Spillprog bachelor	11:36-16:38	
04-11	**world generation**	**1:08:10**	Henningel
	Spillprog bachelor	18:50-19:59	
04-11	**world generation**	**0:43:04**	Henningel
	Spillprog bachelor	20:58-21:41	
04-12	**meeting, discussing**	**6:30:30**	Jonas Reitan
	Spillprog bachelor	08:57-15:27	
04-12	**Meeting Jonas**	**6:30:00**	Christerpsomby
	Spillprog bachelor	09:00-15:30	

04-12	**meeting at jonas**	**6:30:00**	Kristoffer Eidså
	Spillprog bachelor	09:00-15:30	
04-12	**thursday meeting**	**6:30:00**	Henningel
	Spillprog bachelor	09:00-15:30	
04-13	**world generation**	**5:20:00**	Jonas Reitan
	Spillprog bachelor	08:34-13:54	
04-13	**Installer**	**3:11:54**	Christerpsomby
	Spillprog bachelor	09:13-12:24	
04-13	**world generating and database stuff**	**2:48:27**	Kristoffer Eidså
	Spillprog bachelor	11:06-13:54	
04-13	**world generating and database stuff**	**2:11:49**	Kristoffer Eidså
	Spillprog bachelor	14:36-16:48	
04-14	**world generation**	**4:08:00**	Jonas Reitan
	Spillprog bachelor	08:22-12:30	
04-14	**Installer (DLLs)**	**5:39:24**	Christerpsomby
	Spillprog bachelor	08:36-14:15	
04-14	**world generating and database stuff**	**2:00:25**	Kristoffer Eidså
	Spillprog bachelor	11:01-13:02	
04-14	**world generation**	**0:42:43**	Jonas Reitan
	Spillprog bachelor	12:42-13:24	
04-14	**world generating and database stuff**	**1:19:26**	Kristoffer Eidså
	Spillprog bachelor	14:05-15:24	
04-14	**world generating and database stuff**	**1:44:32**	Kristoffer Eidså
	Spillprog bachelor	20:06-21:50	
04-15	**world generating and database stuff**	**1:29:15**	Kristoffer Eidså
	Spillprog bachelor	07:32-09:02	
04-15	**Installer**	**7:11:10**	Christerpsomby
	Spillprog bachelor	08:35-15:47	
04-15	**world generation and meeting with Mariusz**	**4:11:32**	Jonas Reitan
	Spillprog bachelor	10:34-14:45	
04-15	**world generating and database stuff**	**1:39:20**	Kristoffer Eidså
	Spillprog bachelor	11:19-12:58	
04-15	**Meeting with Mariusz**	**1:15:10**	Kristoffer Eidså
	Spillprog bachelor	13:30-14:45	
04-15	**meeting with mariusz**	**1:15:00**	Henningel
	Spillprog bachelor	13:30-14:45	
04-15	**world generation**	**2:05:47**	Henningel
	Spillprog bachelor	15:50-17:56	
04-15	**Installer**	**2:25:21**	Christerpsomby
	Spillprog bachelor	19:49-22:14	
04-15	**world generation**	**4:51:38**	Henningel
	Spillprog bachelor	20:37-01:29	

04-16	**world generator**	**7:49:52**	Jonas Reitan
	Spillprog bachelor	07:28-15:17	
04-16	**Installer**	**2:25:40**	Christerpsomby
	Spillprog bachelor	09:35-12:00	
04-16	**world generating and database stuff**	**3:08:01**	Kristoffer Eidså
	Spillprog bachelor	10:01-13:09	
04-16	**world generation**	**3:59:43**	Henningel
	Spillprog bachelor	15:08-19:08	
04-16	**Installer**	**0:11:22**	Christerpsomby
	Spillprog bachelor	20:00-20:11	
04-16	**Installer**	**2:57:36**	Christerpsomby
	Spillprog bachelor	20:20-23:18	
04-16	**world generation**	**1:17:10**	Henningel
	Spillprog bachelor	23:05-00:23	
04-16	**world generating and database stuff**	**1:01:04**	Kristoffer Eidså
	Spillprog bachelor	23:12-00:13	
04-17	**Installer**	**3:07:33**	Christerpsomby
	Spillprog bachelor	08:32-11:40	
04-17	**world generation, reading/researching stuff**	**6:21:45**	Jonas Reitan
	Spillprog bachelor	10:03-16:25	
04-17	**world generation**	**3:39:10**	Henningel
	Spillprog bachelor	12:02-15:41	
04-17	**world generating and database stuff**	**4:51:07**	Kristoffer Eidså
	Spillprog bachelor	12:05-16:56	
04-17	**Installer**	**1:03:59**	Christerpsomby
	Spillprog bachelor	12:26-13:30	
04-17	**Installer**	**0:28:28**	Christerpsomby
	Spillprog bachelor	13:37-14:05	
04-17	**world generation**	**9:28:11**	Henningel
	Spillprog bachelor	20:12-05:40	
04-18	**world generation**	**6:13:40**	Jonas Reitan
	Spillprog bachelor	07:33-13:46	
04-18	**world generating and database stuff**	**5:30:50**	Kristoffer Eidså
	Spillprog bachelor	11:21-16:52	
04-19	**world generation**	**5:18:50**	Jonas Reitan
	Spillprog bachelor	09:00-14:18	
04-19	**Installer, dependency**	**5:10:32**	Christerpsomby
	Spillprog bachelor	09:35-14:45	
04-19	**world generating and database stuff**	**5:08:09**	Kristoffer Eidså
	Spillprog bachelor	12:47-17:55	
04-20	**world generation**	**3:54:41**	Jonas Reitan
	Spillprog bachelor	08:14-12:08	

04-20	**Installer**	**2:51:32**	Christerpsomby
	Spillprog bachelor	10:00-12:52	
04-20	**world generating and database stuff**	**5:15:51**	Kristoffer Eidså
	Spillprog bachelor	11:46-17:02	
04-20	**world generation**	**0:02:09**	Henningel
	Spillprog bachelor	13:25-13:27	
04-20	**world generation**	**4:48:13**	Henningel
	Spillprog bachelor	17:03-21:52	
04-20	**Installer**	**1:54:09**	Christerpsomby
	Spillprog bachelor	17:04-18:58	
04-21	**world generation**	**1:35:00**	Henningel
	Spillprog bachelor	01:17-02:52	
04-21	**world generation**	**4:10:18**	Jonas Reitan
	Spillprog bachelor	08:05-12:15	
04-21	**world generating and database stuff**	**5:08:25**	Kristoffer Eidså
	Spillprog bachelor	11:03-16:11	
04-21	**Installer**	**3:33:12**	Christerpsomby
	Spillprog bachelor	11:50-15:24	
04-21	**world generation**	**1:52:39**	Henningel
	Spillprog bachelor	16:40-18:32	
04-21	**Installer**	**1:36:09**	Christerpsomby
	Spillprog bachelor	20:07-21:43	
04-21	**world generation**	**3:12:29**	Henningel
	Spillprog bachelor	23:30-02:42	
04-22	**Installer**	**2:16:48**	Christerpsomby
	Spillprog bachelor	10:49-13:05	
04-22	**world generating and database stuff**	**4:49:39**	Kristoffer Eidså
	Spillprog bachelor	11:03-15:53	
04-22	**world generation**	**1:39:42**	Henningel
	Spillprog bachelor	11:46-13:26	
04-22	**world generation**	**1:04:33**	Henningel
	Spillprog bachelor	14:58-16:03	
04-22	**Meeting**	**0:30:45**	Kristoffer Eidså
	Spillprog bachelor	16:00-16:31	
04-22	**Spring Meeting**	**0:31:53**	Christerpsomby
	Spillprog bachelor	16:02-16:33	
04-22	**friday meeting**	**0:30:46**	Henningel
	Spillprog bachelor	16:03-16:33	
04-22	**Installer**	**0:09:41**	Christerpsomby
	Spillprog bachelor	16:33-16:43	
04-22	**world generation**	**1:35:57**	Henningel
	Spillprog bachelor	16:45-18:21	
04-22	**world generation**	**4:32:26**	Jonas Reitan
	Spillprog bachelor	17:42-22:15	

04-22	**Installer**	**0:14:00**	Christerpsomby
	Spillprog bachelor	18:03-18:17	
04-22	**Installer**	**0:11:53**	Christerpsomby
	Spillprog bachelor	19:32-19:44	
04-22	**Installer**	**3:58:15**	Christerpsomby
	Spillprog bachelor	20:17-00:15	
04-23	**Installer**	**4:15:01**	Christerpsomby
	Spillprog bachelor	12:20-16:35	
04-23	**world generating and database stuff**	**4:42:25**	Kristoffer Eidså
	Spillprog bachelor	12:23-17:06	
04-23	**world generation**	**4:05:55**	Henningel
	Spillprog bachelor	13:36-17:42	
04-23	**world generation**	**6:10:25**	Jonas Reitan
	Spillprog bachelor	13:43-19:53	
04-23	**Installer**	**2:10:37**	Christerpsomby
	Spillprog bachelor	21:12-23:22	
04-24	**Installer**	**1:43:30**	Christerpsomby
	Spillprog bachelor	02:45-04:29	
04-24	**world generation**	**3:24:00**	Jonas Reitan
	Spillprog bachelor	08:58-12:22	
04-24	**world generation**	**4:44:56**	Henningel
	Spillprog bachelor	11:27-16:12	
04-24	**world generating and database stuff**	**3:59:22**	Kristoffer Eidså
	Spillprog bachelor	12:07-16:06	
04-24	**Installer**	**4:42:54**	Christerpsomby
	Spillprog bachelor	14:54-19:36	
04-24	**world generation**	**3:52:53**	Henningel
	Spillprog bachelor	17:08-21:01	
04-24	**world generation**	**1:16:04**	Jonas Reitan
	Spillprog bachelor	18:31-19:47	
04-24	**world generation**	**5:50:57**	Henningel
	Spillprog bachelor	21:30-03:21	
04-25	**world generating and database stuff**	**5:04:50**	Kristoffer Eidså
	Spillprog bachelor	13:42-18:47	
04-25	**Installer**	**3:23:05**	Christerpsomby
	Spillprog bachelor	15:16-18:39	
04-25	**database integration**	**2:36:31**	Henningel
	Spillprog bachelor	16:23-18:59	
04-26	**group meeting at my place**	**3:00:00**	Jonas Reitan
	Spillprog bachelor	10:00-13:00	
04-26	**Meeting Jonas**	**3:00:00**	Christerpsomby
	Spillprog bachelor	10:00-13:00	
04-26	**meeting at jonas**	**3:00:19**	Kristoffer Eidså
	Spillprog bachelor	10:00-13:00	

04-26	**meeting at jonas**	**3:00:01**	Henningel
	Spillprog bachelor	10:00-13:00	
04-26	**world generation**	**0:58:57**	Henningel
	Spillprog bachelor	13:47-14:46	
04-26	**world generating and database stuff**	**0:51:50**	Kristoffer Eidså
	Spillprog bachelor	13:54-14:46	
04-26	**Installer**	**0:45:36**	Christerpsomby
	Spillprog bachelor	13:57-14:42	
04-26	**Installer**	**2:12:36**	Christerpsomby
	Spillprog bachelor	22:39-00:51	
04-26	**world generating and database stuff**	**1:44:21**	Kristoffer Eidså
	Spillprog bachelor	22:47-00:31	
04-27	**Installer**	**1:52:30**	Christerpsomby
	Spillprog bachelor	02:10-04:02	
04-27	**mustad meeting**	**1:28:29**	Jonas Reitan
	Spillprog bachelor	11:00-12:28	
04-27	**Meeting Mustad**	**1:28:00**	Christerpsomby
	Spillprog bachelor	11:00-12:28	
04-27	**Meeting at mustad**	**1:27:07**	Kristoffer Eidså
	Spillprog bachelor	11:00-12:27	
04-27	**meeting at mustad**	**1:28:00**	Henningel
	Spillprog bachelor	11:00-12:28	
04-27	**world generating and database stuff**	**3:36:28**	Kristoffer Eidså
	Spillprog bachelor	12:27-16:04	
04-27	**world generation**	**6:33:54**	Jonas Reitan
	Spillprog bachelor	12:28-19:02	
04-27	**Installer**	**1:40:25**	Christerpsomby
	Spillprog bachelor	13:36-15:17	
04-27	**world generation**	**0:18:25**	Henningel
	Spillprog bachelor	16:02-16:20	
04-27	**Installer**	**2:22:20**	Christerpsomby
	Spillprog bachelor	21:20-23:42	
04-28	**world generation**	**0:32:34**	Henningel
	Spillprog bachelor	02:13-02:45	
04-28	**Installer**	**4:07:34**	Christerpsomby
	Spillprog bachelor	02:20-06:28	
04-28	**world generation**	**6:06:49**	Jonas Reitan
	Spillprog bachelor	12:03-18:09	
04-28	**world generation with more database integration**	**4:39:04**	Henningel
	Spillprog bachelor	12:18-16:57	
04-28	**world generating and database stuff**	**3:45:31**	Kristoffer Eidså
	Spillprog bachelor	13:13-16:58	

04-28	**world generating and database stuff**	**2:21:09**	Kristoffer Eidså
	Spillprog bachelor	18:28-20:49	
04-28	**Checking on SQL debugging**	**2:23:52**	Christerpsomby
	(no project)	18:33-20:57	
04-28	**world generation**	**4:21:29**	Henningel
	Spillprog bachelor	21:47-02:08	
04-28	**Installer**	**2:46:35**	Christerpsomby
	Spillprog bachelor	23:06-01:52	
04-29	**world generating and database stuff**	**3:57:23**	Kristoffer Eidså
	Spillprog bachelor	12:02-16:00	
04-29	**world generation**	**1:48:44**	Henningel
	Spillprog bachelor	14:17-16:06	
04-29	**meeting**	**0:16:18**	Jonas Reitan
	Spillprog bachelor	16:00-16:16	
04-29	**Meeting**	**0:12:33**	Kristoffer Eidså
	Spillprog bachelor	16:00-16:12	
04-29	**Spring Meeting**	**0:12:17**	Christerpsomby
	Spillprog bachelor	16:00-16:12	
04-29	**friday meeting**	**0:34:39**	Henningel
	Spillprog bachelor	16:06-16:41	
04-29	**world generating and database stuff**	**0:52:14**	Kristoffer Eidså
	Spillprog bachelor	16:12-17:04	
04-29	**world generation with henning**	**3:10:42**	Jonas Reitan
	Spillprog bachelor	16:16-19:27	
04-29	**Installer**	**1:01:58**	Christerpsomby
	Spillprog bachelor	16:25-17:27	
04-29	**world generation with jonas**	**2:43:47**	Henningel
	Spillprog bachelor	16:41-19:25	
04-29	**world generating and database stuff**	**0:11:54**	Kristoffer Eidså
	Spillprog bachelor	18:16-18:28	
04-29	**world generator stuff, rotating balls**	**1:51:04**	Jonas Reitan
	Spillprog bachelor	20:06-21:57	
04-29	**adding cube**	**1:41:12**	Henningel
	Spillprog bachelor	20:07-21:49	
04-29	**Installer**	**2:17:11**	Christerpsomby
	Spillprog bachelor	20:21-22:38	
04-29	**world generating and database stuff**	**0:34:49**	Kristoffer Eidså
	Spillprog bachelor	21:18-21:53	
04-29	**pushed some broken stuff.**	**0:08:59**	Jonas Reitan
	Spillprog bachelor	23:40-23:48	
04-30	**world generation**	**3:05:37**	Jonas Reitan
	Spillprog bachelor	16:48-19:53	

04-30	**Installer**	**1:58:21**	Christerpsomby
	Spillprog bachelor	16:49-18:48	
04-30	**world generating and database stuff**	**2:35:27**	Kristoffer Eidså
	Spillprog bachelor	16:50-19:25	
04-30	**Installer**	**2:06:27**	Christerpsomby
	Spillprog bachelor	21:46-23:53	
05-01	**world generation**	**0:41:53**	Henningel
	Spillprog bachelor	01:14-01:56	
05-01	**world generation**	**2:26:06**	Henningel
	Spillprog bachelor	03:44-06:10	
05-01	**Installer**	**0:36:06**	Christerpsomby
	Spillprog bachelor	04:09-04:45	
05-01	**world generation**	**9:29:49**	Jonas Reitan
	Spillprog bachelor	11:08-20:38	
05-01	**world generating and database stuff**	**4:47:14**	Kristoffer Eidså
	Spillprog bachelor	12:10-16:58	
05-01	**world generation**	**7:08:59**	Henningel
	Spillprog bachelor	16:08-23:17	
05-01	**Installer**	**0:46:49**	Christerpsomby
	Spillprog bachelor	16:50-17:36	
05-02	**world generating and database stuff**	**5:03:00**	Kristoffer Eidså
	Spillprog bachelor	12:46-17:49	
05-02	**world generation**	**5:33:23**	Jonas Reitan
	Spillprog bachelor	14:13-19:46	
05-02	**fixing proper cube mesh**	**1:49:14**	Henningel
	Spillprog bachelor	18:17-20:07	
05-02	**Installer**	**0:24:23**	Christerpsomby
	Spillprog bachelor	22:51-23:15	
05-03	**Installer**	**1:17:56**	Christerpsomby
	Spillprog bachelor	05:32-06:50	
05-03	**world generating and database stuff**	**5:19:48**	Kristoffer Eidså
	Spillprog bachelor	12:15-17:35	
05-03	**world generation**	**2:13:48**	Henningel
	Spillprog bachelor	15:08-17:22	
05-03	**Installer**	**0:38:00**	Christerpsomby
	Spillprog bachelor	18:51-19:29	
05-03	**world generation and research**	**5:06:39**	Jonas Reitan
	Spillprog bachelor	19:53-00:59	
05-03	**Installer**	**4:36:22**	Christerpsomby
	Spillprog bachelor	20:00-00:36	
05-04	**meeting at school**	**5:20:00**	Jonas Reitan
	Spillprog bachelor	13:40-19:00	

05-04	**School**	**5:20:00**	Christerpsomby
	Spillprog bachelor	13:40-19:00	
05-04	**played with coin at school**	**5:20:27**	Henningel
	Spillprog bachelor	13:40-19:00	
05-04	**went to school**	**5:20:14**	Kristoffer Eidså
	Spillprog bachelor	13:40-19:00	
05-05	**world generating and database stuff**	**5:28:28**	Kristoffer Eidså
	Spillprog bachelor	12:18-17:46	
05-05	**research and note taking**	**4:55:07**	Jonas Reitan
	Spillprog bachelor		
05-05	**Installer**	**3:42:38**	Christerpsomby
	Spillprog bachelor	15:57-19:40	
05-05	**world generation**	**1:55:11**	Henningel
	Spillprog bachelor	17:00-18:55	
05-05	**world generation**	**1:06:17**	Henningel
	Spillprog bachelor	19:20-20:26	
05-06	**Installer**	**2:18:00**	Christerpsomby
	Spillprog bachelor	07:03-09:21	
05-06	**world generating and database stuff**	**5:00:05**	Kristoffer Eidså
	Spillprog bachelor	12:11-17:11	
05-06	**Installer**	**1:37:31**	Christerpsomby
	Spillprog bachelor	18:40-20:18	
05-07	**Reading bachelor reports**	**2:05:38**	Christerpsomby
	Spillprog bachelor	01:17-03:22	
05-07	**world generation**	**2:10:01**	Henningel
	Spillprog bachelor	02:50-05:00	
05-07	**world generation**	**5:11:26**	Jonas Reitan
	Spillprog bachelor	05:33-10:44	
05-07	**world generation**	**5:11:32**	Henningel
	Spillprog bachelor	10:38-15:49	
05-07	**world generating and database stuff**	**5:36:05**	Kristoffer Eidså
	Spillprog bachelor	12:34-18:11	
05-07	**Installer**	**1:59:44**	Christerpsomby
	Spillprog bachelor	16:13-18:13	
05-07	**world generation**	**1:10:17**	Henningel
	Spillprog bachelor	17:27-18:37	
05-07	**world generation**	**1:03:18**	Henningel
	Spillprog bachelor	20:10-21:13	
05-07	**Installer**	**3:17:26**	Christerpsomby
	Spillprog bachelor	21:13-00:30	
05-08	**Installer**	**6:45:11**	Christerpsomby
	Spillprog bachelor	06:22-13:07	

05-08	**world generation and some research**	**9:01:28**	Jonas Reitan
	Spillprog bachelor	08:12-17:13	
05-08	**world generating and database stuff**	**3:15:55**	Kristoffer Eidså
	Spillprog bachelor	12:11-15:27	
05-08	**enemy stuff**	**2:24:39**	Henningel
	Spillprog bachelor	12:29-14:53	
05-08	**Installer**	**0:59:16**	Christerpsomby
	Spillprog bachelor	14:14-15:13	
05-08	**enemy stuff**	**0:56:11**	Henningel
	Spillprog bachelor	15:35-16:31	
05-08	**enemy stuff**	**3:20:58**	Henningel
	Spillprog bachelor	17:02-20:23	
05-08	**documenting**	**0:37:42**	Henningel
	Spillprog bachelor	20:40-21:18	
05-08	**documentation /enemy stuff**	**5:57:35**	Henningel
	Spillprog bachelor	22:36-04:34	
05-09	**world generating and database stuff**	**5:04:10**	Kristoffer Eidså
	Spillprog bachelor	12:11-17:15	
05-09	**world stuff**	**5:06:00**	Jonas Reitan
	Spillprog bachelor	12:26-17:32	
05-09	**Installer**	**0:58:34**	Christerpsomby
	Spillprog bachelor	20:34-21:33	
05-09	**Installer**	**2:06:24**	Christerpsomby
	Spillprog bachelor	22:11-00:18	
05-10	**Moving database and stuff**	**5:14:39**	Kristoffer Eidså
	Spillprog bachelor	12:11-17:26	
05-10	**confluence, research and stuff**	**5:07:42**	Jonas Reitan
	Spillprog bachelor	13:03-18:10	
05-10	**mostly documentation stuff**	**4:08:57**	Henningel
	Spillprog bachelor	16:00-20:09	
05-10	**Installer**	**2:06:51**	Christerpsomby
	Spillprog bachelor	19:03-21:09	
05-11	**world generating and database stuff**	**5:29:07**	Kristoffer Eidså
	Spillprog bachelor	12:11-17:40	
05-11	**discussing confluence, bachelor and art stuff, changed some stuff, wrote some stuff**	**5:23:50**	Jonas Reitan
	Spillprog bachelor	15:42-21:05	
05-11	**documentation**	**2:00:16**	Henningel
	Spillprog bachelor	18:38-20:39	
05-12	**world generating and database stuff**	**5:32:11**	Kristoffer Eidså
	Spillprog bachelor	09:13-14:45	
05-12	**confluence writing and discussions**	**6:25:36**	Jonas Reitan
	Spillprog bachelor	11:02-17:27	

05-12	**document writing**	**3:21:16**	Henningel
	Spillprog bachelor	17:19-20:40	
05-12	**Documentation**	**2:20:00**	Christerpsomby
	Spillprog bachelor	19:03-21:23	
05-12	**Confluence Settings**	**3:13:52**	Christerpsomby
	Spillprog bachelor	21:52-01:05	
05-12	**documentation**	**1:32:29**	Henningel
	Spillprog bachelor	23:05-00:38	
05-13	**Writing and fixing stuff**	**4:30:24**	Kristoffer Eidså
	Spillprog bachelor	11:05-15:35	
05-13	**confluence writing**	**3:30:00**	Jonas Reitan
	Spillprog bachelor	12:02-15:32	
05-13	**documentation**	**2:11:17**	Henningel
	Spillprog bachelor	12:59-15:10	
05-13	**Documentation**	**5:35:50**	Christerpsomby
	Spillprog bachelor	16:05-21:40	
05-13	**meeting, more confluence stuff**	**4:09:00**	Jonas Reitan
	Spillprog bachelor	16:08-20:17	
05-13	**friday meeting**	**0:33:01**	Henningel
	Spillprog bachelor	16:17-16:50	
05-13	**Meeting**	**0:30:34**	Kristoffer Eidså
	Spillprog bachelor	16:18-16:48	
05-13	**Documentation**	**4:26:20**	Christerpsomby
	Spillprog bachelor	23:54-04:20	
05-14	**documentation**	**3:34:08**	Henningel
	Spillprog bachelor	00:13-03:47	
05-14	**Writing and fixing stuff**	**4:58:09**	Kristoffer Eidså
	Spillprog bachelor	13:04-18:02	
05-14	**confluence documentation**	**6:56:00**	Jonas Reitan
	Spillprog bachelor	17:05-00:01	
05-14	**Documentation**	**4:02:31**	Christerpsomby
	Spillprog bachelor	18:03-22:05	
05-14	**documentation**	**6:59:53**	Henningel
	Spillprog bachelor	18:12-01:11	
05-14	**Documentation**	**0:38:36**	Christerpsomby
	Spillprog bachelor	23:27-00:05	
05-15	**made new art for confluence**	**1:14:39**	Jonas Reitan
	Spillprog bachelor	00:25-01:39	
05-15	**Confluence Settings**	**2:24:00**	Christerpsomby
	Spillprog bachelor	07:32-09:56	
05-15	**Writing and fixing stuff**	**2:54:17**	Kristoffer Eidså
	Spillprog bachelor	13:50-16:44	

05-15	**Documentation**	**6:30:03**	Christerpsomby
	Spillprog bachelor	16:28-22:58	
05-15	**confluence writing about development process, noise, generation etc.**	**10:25:00**	Jonas Reitan
	Spillprog bachelor	18:20-04:45	
05-15	**Writing and fixing stuff**	**2:20:18**	Kristoffer Eidså
	Spillprog bachelor	20:02-22:22	
05-15	**Documentation**	**0:44:56**	Christerpsomby
	Spillprog bachelor	23:46-00:31	

Created with toggl.com

D. Medieval Brawl

Medieval Brawl

Installer: https://www.dropbox.com/sh/07bcrmhaafqvl3o/AABHKwj12Zr4hev3zCfMRgfda?dl=0

Repo: https://bitbucket.org/Almightyquad/roguelike-game-programming on the master branch.

We also made another game with the engine in android, which is on the Android branch.

Description of the game

Summary

Terminology:

Engine: Not really an engine, more of a tool.

Engine component: General term for what the engine can use

Actor: All objects in the game.

Actor component: Parts of an actor that make up the actor.

GUI element: example: Button, slider, dropdown, …

GUI state: Main menu, running, game over, …

Android NDK: Android Native Development Kit

Description of the game

Summary

We decided during the summer that we would be working on a roguelike, where we look at all the interesting design of some of the games in the genre, and improve on that. We set out to create a game with a seamless procedurally generated random world, with a crafting system and vendors. The player would progress by upgrading his equipment and finding new weapons. We set out by aiming high and delivering whatever we had enough time to do.

We were able to implement about half of the game elements that we wanted to, and we believe we made the best game we could make with the tech we wanted to implement

and the time we had.

Medieval Brawl has a randomly generated world, where the world is created with tiles, which are placed in 8x8 chunks (more on this later), which are seamlessly connected to each other, We have four fairly diverse enemies and the player can find three different weapons. The weapons are also somewhat unique. All enemies and weapons are spawned randomly throughout the entire world.

The player can choose between four different characters, which have some distinguishing features. The players can also spawn with set items, which can distinguish them even further with just one line of code. This is mostly done to display flexibility and ease of use of the engine, and not really utilized to it's full potential.

The engine currently support up to 32 player multiplayer, but starts getting very choppy at around 6 players. This number will go up and down depending on the hardware of the host and clients. In Medieval Brawl the players can fight in co-op against the horde of monsters with the spawned loot.

Players can pick up and drop items as they see fit, and there is some strategic element to how to beat the game. The player(s) beat Medieval Brawl after defeating the final boss which spawns after a certain amount of enemies have been killed in a single session.

Technology

Graphics

Our engine uses modern OpenGL and OpenGL ES 3 for drawing on various platforms. We decided to use a OpenGL version that also was supported for mobile phones and we also wanted to use the modern features ES3 offered, so we had to remove compatibility for older phones to make the game look better and not use the newer OpenGL version.

One of the main features we wanted from ES3 was instancing so we could add particle effects without having one draw call per particle, which would slow down any device. We used this particle effect system to add a weather effect, which is currently displayed as snow on our desktop game and leaves on our mobile game. The particles are also used

for entities in our engine, an example for this is when you pick up a star on the mobile game, it bursts out tiny stars in various red-orange colors, while on our desktop game we attach particles to projectiles, like arrows and magic.

Since we use OpenGL, everything is done in 3D, so we got a lot of flexibility even though our games are mainly 2D. This means that we can have actual 3D parallax background instead of layers and have 3D models in game. One example of 3D models we use are cubes for platforms in our desktop game.

We have a skybox implement which supports two texture and will fade between them based on the time of the day. In our desktop game, we use a day and night skybox but on our android version we use it as a parallax background effect and have a static picture that rotates at a much slower rate. This way we can use the skybox for multiple purposes in a lot of different type of games.

Very late into the development cycle we started implementing framebuffer objects (FBOs) so we could add various effects to the world. We started off by creating a lighting system where we would draw circles with a certain color and then multiply this with the world to give light sources like torches actual light while everything else in the world had a dark ambient to it.

This can be used to other things aswell, for example by using the stencil buffer, we could stencil out the background and overlay a blood texture to apply blood to the ground where something dies to increase immersion.

Audio Handling

We are using SDL_MIXER for our sound system. We were trying to get OpenAL to work but we ran into a few problems. We needed to make the Cmake file for openAL ourselves and the one we made only seemed to work for one of us and we decided to just run with SDL_MIXER because we did not want to spend too much time on the audio system because we felt that other aspects of the engine was more important. When we load sounds or music files we check to see if they already exist and if not we save them to a map with sounds with a string identifier.

Our sound system is really basic with these possibilities at the moment:

Play music and sounds: We can play a sound or a music track loaded from our sound

map.

Mute music / sound - Mute sound and mute music are two different functions that basically sets the volume to 0.

Change volume: You can change the volume to a specific number. This function works great when linked to our sliders.

Change the master volume: changes both the sounds and the music at the same time.

stop/pause/resume music: we can stop, pause and resume the music. Especially great to use these when we were making a game for Android since we needed to stop sounds from playing while our application was minimized

We should look into using something different than SDL_MIXER if we decide to keep using our engine in the future, since it is a bit too basic and lacks some core functions that we want. For example SFML has a great function to play sounds from a location in the world to make the game feel more alive and real. This is something we would want with our future sound system and SDL_MIXER couldn't do this the way we wanted it.

Android

We have a document for Android as well, which can be found here.

We have building and compiling for Android. There is a readme on the repo on how to build it and deploy it to the phone. The game on the master branch may not run on android due to the assets not being updated there, but the one on the Android branch should run just fine.
We used the Android NDK for our building and ant for deploying. The original intention was to make a make system, but time limitations made us only have the android make system.
Many things you would see as easy on PC is a lot more tedious on Android. Examples would be loading a file, keeping the window the same size when re-entering the program, stopping music when the game is in the background, movement, touch (which is the same as a mouse in some instances), and so on. Debugging can also be a challenge, since you mostly get back stack information and memory locations, which takes some time to learn how to read. So having logs in the project is very useful.

File Handling in Android

The file handling in Android is done slightly differently from PC. Everything in the APK is compressed, so you have to use Androids AAssetManager to get out your files from the assets folder. Everything you read here needs to be read into a buffer, which you can

then put into a stream.
What you can also do is save files and such in the android filesystem. Which you need to get using a seek from an Android function. The pach turns out to be something like: “data/data/[yourpackagename]/Medieval Brawl/files”, writing to here can be done using pretty much anything that reads and writes a file like fopen, ofstream and ifstream. This location is also specific for your application, only you can read and write here.

AI

AI is applied to an actor by adding the AIComponent to it. The AI component takes a string parameter as a path to a Lua file, but pure c++ code can be used if no string is passed along. We wanted the implementation of lua to be 100% dynamic, but that proved to be difficult, because none of us had any prior experience with Lua. Debugging in non-pure Lua was also no easy task, so we went for more c++ oriented AI towards the end of the project deadline.

Most of the difficulties with implementing Lua dynamically comes from the parameters you need to send from the main codebase. The need to know which, how many, and how many return parameters you need made it very tough to do with our knowledge.

Level editor

We are currently using the program Tiled to create our levels. Since our game world are a tiled platformer Tiled works good for this project. We use our own level loader to parse the Tiled files and create the graphics of the world with a tile atlas. We want to create our own level some time in the future because we feel that Tiled is a bit clunky to use and it makes it harder to just edit stuff and test on the fly. We didn’t create a level editor for this project because the world was going to be randomly generated anyway.

Font:

We are using SDL_TTF because we wanted something that worked and didn’t take a lot of time to learn or implement. We figured that we didn’t really need any advanced texts in our game and our priorities were elsewhere in the engine. Like with our audio system we really want to use something better if using the engine in the future, since SDL_TTF doesn’t have a lot of customization. For example getting a outlined text would just create a outlined text and remove everything in the middle of the text. The only customization we have at the moment is changing the color of the text. Since we just make an SDL_texture and make it into an openGL texture, the text also looks weird and stretched depending on the size of the word or mesh that it is attached to and all our customization

happens in a shader. We didn’t find any font system that was really easy to implement or that did enough of what we needed for this project so we just went with this solution.

Filehandling

Like the rest of the engine the graphical user interface is also loaded from xml files. We looked into using lua to do everything for us and we actually had some lua code for gui in the beginning. We found out that you needed to pre map functions you want to run to use them in lua files and we found out that it would take some time to rewrite the code to get this to work. Since we felt that the gui was already working great we decided to just use our current way so one of us didn’t have to use all his time coding the gui.

There is a gui loader file that has the name of all the game states in it. This file is read when we start our game and the gui handler creates all the states that it finds in this file. The gui handler then tries to open an xml file with the name of each state. For example RUNNING.xml. In each game state xml file all the gui elements that belongs to the state is listed. All elements of the same kind has to be in order so for example every button has to be in order. The file is then parsed and will look under each element to find their property and create the element. When a gamestate is changed the gui handler will load the correct state with the correct elements.

The elements that the user can interact with have a listener that is currently listed in the bottom of the engine.cpp (This is supposed to be decoupled from the engine but since we didn’t have time to decouple the game states this is the hacky solution for now). The listener listens for messages that it is subscribed to and when they get the right kind of message they will run the functions that is written in the listener.

Networking

We use SDL_net for networking in our engine. We wanted to use something more flexible and easier to use, but because we already had SDL implemented we wanted to keep everything as close to SDL as possible due to portability and familiarity with the structure. Our engine currently only supports TCP packets. We decided for using TCP because most of the speed problems and we minimize the packets as much as possible. We are looking into further reducing the amount of data being sent to be able to support more players.

All network clients only send their input to the host, and the host redirects the input to the correct components of the engine(more on this later), and is handled the same way input from the host is, with a one frame delay. The host sends the positions and velocity of all living moving actors across the network. This can be improved by making sure that the position that is being sent is in fact a new position, and not the same that was sent last frame.

The client stores all this data in his own actor structure, and creates new actors when necessary. The new actors are created with the velocity sent by the host. Old positions are currently being overwritten by new ones, even if they are only 0.001 units apart (only a few pixels). The game would feel more smooth for the client if he only updates the position for each actor if the current position is more than a certain distance apart.

Deletion of actors is handled by the client and host individually. All deletion is handled by the physics component(more about this under physics), so as long as the positions are the same one frame earlier, they should be the same on the next frame, and the collision should take care of the deletion. This is probably not a 100% safe assumption, but we have not seen a situation where the client and host has desynced. This could be solved by having the host send messages regarding deletion of actors as well as creations, and prevent deletion altogether on the client side.

We wish to rewrite and replace SDL_net with ZeroMQ for the bachelor if we were to improve on the engine. We also want to make the networking more flexible and customizable for each game, so we would have to rewrite most of the contents already, which is part of the reasoning for why some of the issues and weaknesses are not fixed.

Event handling

All our physical interface events are behind handled by SDL. This includes mouse events, keyboard events and touch events. The event handler is isolated from the rest of the game and engine, where the event handler only sends information around and other components know what to do with it. Objects which subscribe to the eventhandler will get the event that happened, and will make the own decisions on what to do with it. Most of the events sent to subscribers are discarded as they are not used by that specific object.

All actors in the game can listen for input as long as they have the inputComponent(more on this later). The gui handles input by checking for mouse input from the event handler. If the mouse is clicked the gui elements on that particular game state checks if the mouse click is inside their transform with a simple AABB check. If it is the function returns true

so the event handler can cancel out mouse events that should not happen when gui is clicked. I.E firing a fireball when clicking the pause button. The mouse click function also activates functions in the gui elements that does stuff when clicked. For example buttons gets clicked, sliders gets slid and drop downs gets drop downed.

Physics

Kremengine uses Box2D as physics handler. All game objects which need to collide with something need a body from the Box2D world. We also use box2d when moving objects around in world space.

Libraries used

SDL

Box2D (Physics)

SDL_Mixer (Audio)

SDL_TTF (Fonts)

SDL_Net

OpenGL 3.3

Glew

LUA (AI)

glu

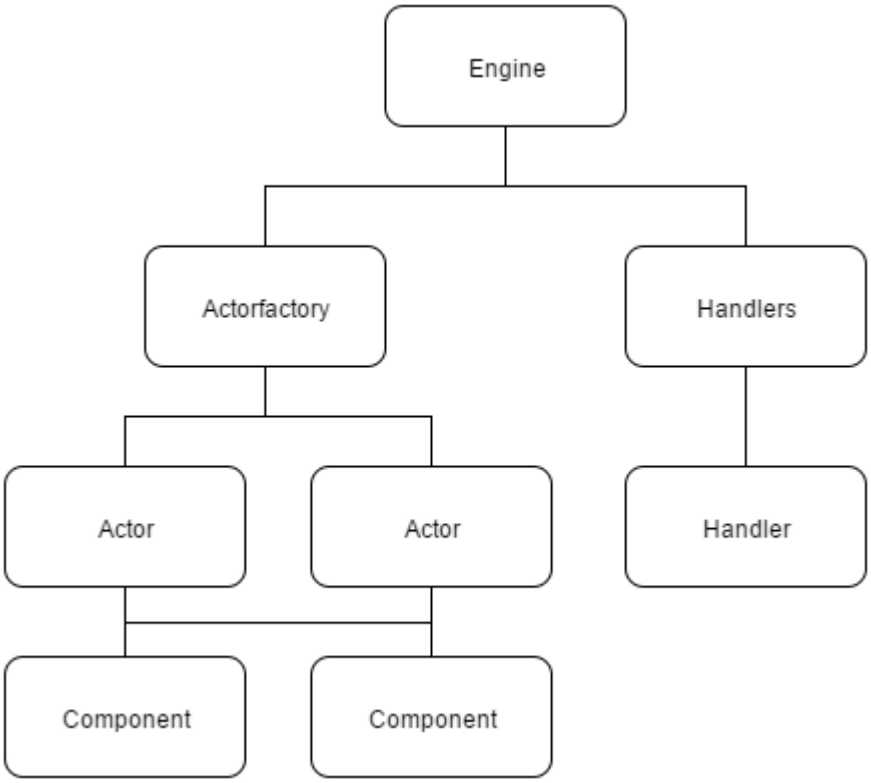
Engine Architecture

Engine

The heart brain of the engine is the Engine class. Engine is initialized when the player starts the game and contains every single handler for all major engine components. Engine also holds the gamestate, which is used for tracking where the player is in the game. Gamestates are currently tied directly to the game, so the programmer has to go directly into the engine and modify the gamestates if he wishes to do so. We currently directly handle typical gamestates like “playing”, “hosting” and “main menu”. Each gamestate has its own update function and initialization function. The update function handles generic drawing and updates, as well as major game specific events, i.e changing gamestates when player health drops to 0.

Engine also one of each major handler in the game. Handlers include PhysicsHandler, GraphicsHandler, EventHandler, Network handlers and many more. Each handler is derived from a base Handler class which provides some tools to make life easier for the programmer. All handlers have a pointer to their parent (Engine). We would like to include more functionality to the handler to make it act more as an interface for all handlers.

Engine is also the main culprit for creating new Actor objects. Engine holds the levelLoader, which loads all the tiles from their chunks (read more about this under filehandling). All tiles are created as Actors (more on this under ActorFactory). New enemies and items are spawned during a set interval in the update loop. The player creation call also comes from the engine for the host and single player player actor. The network creates all the client actors.



The engine object stores the Actorfactory and all Handlers. Handler inherit from Handler. The Actorfactory stores all the Actors, an Actor can have many Components.

ActorFactory

ActorFactory is the hands for Engine. Engine has one ActorFactory, and that one instance is responsible for creating and deleting all drawn objects other than GUI and particles on the screen. ActorFactory takes an XML path, position and velocity as parameters. Everything else that is needed is in the XML file (if it exists).

ActorFactory stores objects in two categories; moving and not moving. Moving actors are primarily actors that can be moved. Players, projectiles, enemies and items fall under this category. Non moving objects are tiles, background objects. Lightsources and invisible collision blocks will probably fall under this category in the future. The moving category is slightly misleading since you can have stationary enemies, which are technically “not moving”. The differentiation is made to handle static objects differently than everything else. Moving objects are stored in a map, with a name and an ID as identifier, while non moving objects are stored in a much faster vector, to more easily iterate between them during draw calls.

ActorFactory also holds all factories for component factories (more on this below).

Actors and Components

Every single actor has at least one component. Components are pieces of actors that make up the whole of one actor. Our engine currently has the following components:

Animation: Added for actors who need multiple sprites. Tracks which sprite it is currently on and the timer for switching sprite. Holds sprites for multiple actor states.

Audio: Used by actors who need to make a sound. Could for instance be added on the sword actor when it hits an enemy to make a slash sound. Holds sounds for different actions.

Combat: Used by actors who engage in some form of combat. Contains damage and/or HP. Actors with a combat component need more than 0 hp to not be deleted. Fireballs and enemies have damage, which deals their damage on contact with objects which are masked for that collision.

Graphics: Used for all actors who need to be drawn from a file. Pure light actors would not need this when we get that far.

AI: Mostly used for enemies, but can easily be used for more interesting items like homing projectiles.

Input: The main actor for the player has input component. Input component receives messages from the eventhandler, and are handled by the input component. Items which are activated on input also uses the input component for flexible input handling. Input can also come as network packets.

Particle:

Physics: This component is required by most actors. This creates a physical body for the actor with the parameters from the xml file. This body is used for all collision handling.

The last two components are more game specific, and we will add functionality for creating components outside of main codebase.

Inventory: This is used by components who can hold items. Enemies and players can both own items with this component attached.

Pickup: Pickup is for actors who are mostly immobile which are intended as items for the player to pick up. These are mostly new weapons and potions. Pickups can only be picked up by actors with an inventory.

Actors can have any number of components, but some components make little sense together in our game. You could have a game where an enemy can also be a pickup, but

this doesn’t fit in our game. Most actors are purely general, and only exist inside of the engine, but we are decoupling more and more as we need them decoupled. The trend looks to be to decouple everything over time.

Which components each actor has depends on the parameters loaded from the xml file. Certain components require certain parameters, while others have default parameters, and you only need to specify that you want the component. GraphicsComponent requires filepaths to the mesh and texture, while AI can just be added by adding the AI component. We have some debugging on the xml loading, so you can see if a parameter is not set in the xml file by enabling xml debugging in the code.

User interface structure

The graphical user interfaces was intended to use some of the components of the actor factory. We quickly figured out that it would be clunky and that the gui needed it owns camera anyway so that the elements wouldn’t be in world space but be visible on the screen all the time. Currently the gui is initiated in the engine by making a gui handler element. The gui handler reads all the game states that the game needs from a gui loader file. From this the gui creates all the states needed for the game through several gui state objects. The gui state object has control over all the gui elements that we want to display or use on that particular state.

The gui handler can find a certain element by searching through its state for a given name. You can use this for several different things. For example changing the container to be visible. This is used for the inventory system. The gui handler can also return objects of the gui element you found so you can subscribe to them and get info when they are clicked and updated wherever you want in the engine. We want to decouple the button listeners from the engine since it is specific to the game created and not the engine it self but we haven’t done this for now.

At the moment gui elements inherit from the gui state which inherit from the gui handler, but nothing is using this to do anything simpler. We would like to rewrite the entire gui system to be more flexible and to not duplicate code. We would like to use templates to make a lot of the functions a lot cleaner and also override functions that do the same thing.

We are currently having a draw call for every gui element. In the future we want to make this a bit better and do it by instancing. The gui isn’t using too much resources right now but in the future it may be a problem and therefore we should rewrite this part.

Enemies

Esmeralda: A crow that flies in the air and drops eggs on the player that does damage.

Chuck: throws a big rock at the player that does damage.

Kappa: jumps up and down and turns around when hitting a wall

Hestmeralda:

Heroes

Jakkheim: Jakkheim is our games young and inspiring hero. He is small and that is why he is faster than the bigger characters. He also has more health than his girlfriend Jackline because he is a male.

Jackline: Since she is the same race as Jakkheim she is faster than the bigger characters as well. She has more mana than Jakkheim to make the two unique.

Dwarfid: Dwarfid is a 400 year old dwarf and have been through some wars. Therefore, he has a lot more health than the other characters, but his age comes with a disadvantage, making him as slow as his wizard friend Frode.

Frode: Frode is the same age as Dwarfid and have been to all the same wars. Since he is a wizard he has tons of mana, but his age and body make him slow and fragile.

Items

Fireball: Fireball is a classic mage spell that shoots a ball of fire in a straight line depending on where you aim. The spell costs mana and can only be cast when the player has 20 or above mana. Does a lot of damage since it is dependent on mana.

Bowandarrow: The Bowandarrow is a weapon which has a string attached to a bow, with magical arrows inside an invisible quiver. The arrows travel like arrows would in real life. does a bit less damage than a fireball.

Sword: Sword is a melee weapon that does a lot of damage since it requires you to go up to the enemies and endanger yourself.

Health and mana potion: Refills the health or mana of the player by 50. Does this automatically and won’t save to the inventory or require the user to activate it. The potion gets removed even if the player picking it up has full health or mana.

Game balance

We have focused a lot more on technology and implementation techniques than game balance and content. We could make 5 iterations of the fireball and bow and arrow with slight differences, and could create other enemies which just spawned other enemies or fired arrows or fireballs towards the player, but these would just be other versions of existing items and enemies. Even if our engine is flexible and easy to use, it would still take some time to tweak these new items without learning anything new or displaying any skill. And this is time we would rather spend on improving our code and tweaking existing content.

Instead we chose to balance Medieval Brawl around having the objects in it be diverse enough to be worth displaying.

The three power-ups for the player are created outside of the engine, and only require a few lines of code each, but still display a lot of utility of the engine. The fireball fire in a straight line, the bow fires in a cone, and the sword is a melee weapon which equips a new physical body which shares a body joint(more on this later) with the player.

Reflection

Responsibilities

Shared responsibilities:

Content, optimization, error handling, actors, physics

Christer:

Android porting, Events.

Kristoffer:

GUI, pickups, inventory, audio

Henning:

Network, component system, decoupling

Jonas:

Graphics

Lessons learned

Don’t start porting to Android late, do it early. Add all the dependencies incrementally.

Sort files into folders. This is something that needs to be done, because the folder is pretty much just chaos.

Make a coding standard before you start. Everything is currently just clutter.

Do templating early. Also, learn templating.

Our experiences

Christer:

I hate the Windows file system, it’s not case sensitive. So everyone can name any file or directory whatever they want, someone names the folder for GUI, but loads from that folder by doing “gui/somefile.txt”, which then in turn works on Windows, but not on a Linux system like Android, since it’s case sensitive! So I ended up battling for a standard in naming files and having people use the literal path to the files.

Just getting started on the Android porting was also very tedious. Going one step forward, ending up going to steps back was a common thing in the beginning. I had to scrap everything I had done after 5 days work and begin again because I tried to do a make system. After scrapping it, I went over to the Android make. Basically, the porting took a week more than expected.
Getting SDL ported was relatively painless, but adding more after that, like Box2D, was challenging. But as soon as I got that working, everything else went smoothly. Until I started with the file loading. But as always, get it working one time, and you can get it working many times. I had to do different loading strategies for everything, some I just had to pass the buffer I got from the loader, some I had to make a stream, some I had to make a file in memory and some I just had to load using SDL specific loading. If you’d like to know more about the process, you can read more on it here.

Kristoffer:

I have learned that coding graphical interfaces is a lot more work than i originally thought. Every time you want to create a new gui element you have to think about a lot of things. Gui elements needs a lot of different properties to be reusable for different games and they need to be dynamic and easy to edit. They also need to run different functions from game to game. Almost every gui element i have added to the engine have

had functions added to them later because they suddenly needed something i didn’t think about when i first created them. For example it's really simple to create buttons that can be clicked. But then you remember that you might want a texture for three different states. Some sound, the button may change its text when something happens, you may want to move it around or the scale can change. This have made me think more about what a class needs from the beginning so i don’t have to rewrite stuff later. So for the gui elements that i added later in the development process where almost complete when i was done adding them instead of a few days/weeks later. I also learned that gui is complicated to get right and needs a lot of tweaking and flexibility to work good, especially if you want to just reuse classes. I would also like to rewrite a lot of the classes. Some of the classes is really hacky written, For example the slider. I also want to override some functions that basically does the same for a lot of the classes to not have duplicated code and use the inheriting that i was supposed to. I also want to use templates. I decided to not do this for the project because I wanted to work on different things in the engine and not focus all of my time on gui since i wanted to learn more about actual game coding.

I also learned a lot about working on a code base that is really big. Since the codebase is so huge everything you want to do or change may have consequences elsewhere in someone else's code or maybe even in your own. I am now more careful when changing stuff around and i always try to understand what the other people in my group has done to not ruin their classes or function. Because of this i feel like i could work on almost any part of the entire engine with little to no effort because i know the structure of the engine really well.

Henning:

I did quite a bit of reading on both the Internet and the book for the course (game coding complete), to try to sew together everything I’ve learned throughout the degree. I was assigned to design a lot of the architecture behind the actor system, and really enjoy working with architectural design even if I’m very inexperienced and new in the field.

Looking back at everything I’ve done I notice that some of the code is subpar, while other pieces are fair to good, and the common denominator for all the good code is that it is rewritten multiple times. I never looked back on how the core of the actorfactory is set up after it was up and running (and to be fair, it is not an easy task to rewrite it at this point), until a few weeks before the due date, at which point it would take too much time to rewrite it. The same goes for the engine.cpp file. These are some of our weakest code bits, and ironically the most important. So rewriting code is good. The collision system is around it’s fourth iteration and is coming along nicely. The physics has undergone some revision and is getting better. Through this project I have learned that when you traverse

unexplored areas, you will have to rewrite it at least once or twice, if not more (with very few exceptions).

When we started making the game, I noticed how every small piece was fairly fast to implement. The system was simple and adding new content was easy for multiple reasons. The first reason is that there's much less code and structure to organize when implementing it, and the other reason is that I did it! It is often tempting to add content because of how easy it is, even if it doesn't really add that much to the engine (that we ended up making, even if it is not really a true engine yet. not by a long shot). But towards the end of the project I could probably solve one or two big parts per day because of how complicated they were (and how tired I often was).

Jonas:

I was very interested in improving my OpenGL knowledge, so I looked at this project as an opportunity to do so, as we wanted to have a lot of different tech in our engine. I started off with looking at shaders, loading models and textures, various technologies OpenGL could offer us and how to implement all of this into a neat and easy to use package.

To save memory, I made a handler to check which assets where loaded and only load new assets. This function returns a raw pointer so every object on screen that needs to be drawn have a pointer to a shader, mesh and texture.

I always like to think about optimization, which is why I went this path. As all of the textures, shaders and objects(mesh) are unique pointers, I could in theory check how many pointers point to a certain object and delete any resources we currently don't need.

This is something I planned to implement but the project never got to the scale where this would be needed, so this is something I got on my "to do" list.

I didn't really know a lot about shaders prior to this, other than the very basic shaders we used in graphical programming course. So that is where I started, by implementing lambertian shading, and from there it just escalated into all the shaders we currently use.

We have a specific shader for everything in our game so the shaders don't do unnecessary checks or look cluttered, this is something I researched and people recommended having a lot of different shaders instead of having shaders doing things you don't really need.

So I went ahead and made a shader for every thinkable scenario we would encounter. Our shaders range from basic with just a color, to just a texture, to multiple textures and to textures where one is generated in code while the other is loaded from file.

When we started porting to Android I had to rewrite all of the shaders to support the OpenGL ES3 format, which was very similar to what I've used as I went for the modern approach.

My knowledge of models and drawing were also very limited to what we learned in graphical programming so I went researched this a lot before starting to find an optimal way. It turns out the method I used were the "hello world" of OpenGL and I realized very late in the development cycle that I should have started using instancing at day one as everything in our game are either cubes or quads. We're currently using one draw call per object, except for particles, which is very expensive. Using instancing would probably improve our performance, even though our performance is fine as it is right now (vsync stable 60 on phone, 500+ on windows).

After I discovered instancing, I started thinking about cool ways to implement this, so the first thing I thought of was particle systems, so I made a simple weather system to rain down blue boxes from the sky. This looked pretty cool so I increased the amount and noticed I could go into the millions without any issues. I learned how to apply textures to particles and that's how our snow was born.

I started this project with some knowledge around SDL and it's various libraries, as we used SDL on previous game jams, so implementing those were a breeze. We had used SDL_TTF before, but never with OpenGL, but this turned out to be easy with how my texture class were built up.

Something I would have changed however is how we display the text. Right now, we load the texture up to a shader and draw it onto a mesh, which means it will scale with the mesh and look rather bad. I would like to redo this in the future by either adding the font as a decal and apply it to an existing texture or look at distance field text to make the scale better.

Theres still a few projects I would like to implement, like water with reflection and ripples, multi textured particles for flame or spell effects, real time shadows using the depth/stencil buffer.

One project I started with was a lighting system, where I planned to use framebuffer objects to draw circular shapes onto and then multiply this with the existing world to give it a really nice look where torches actually emitted light and everything else had a dark ambient color to it.

I could also use this to apply textures to the world, like decals. Something I wanted to

implement was a blood splatter system where I would apply a texture, much like the light, to the position something dies and stencil out the background to only draw blood on the terrain.

This was sadly never completed because I had to prioritize other things.

Overall, I’ve learned a lot, but that’s as expected when we spend 8-12 hours a day for a few months straight and I will definitely use what I’ve learned in future projects.

E. Bachelor Work Log

Bachelor Work Log:

13/01/2016 Wednesday

Undocumented work hours (03:15 - 05:30).

First day of discussing the bachelor project. We started off by writing down group rules so we all agreed on everything before starting the development.

Discussed game design and ideas.

Workload:

Everyone: wrote in document.

(18:00 - 22:00).

Discussing tech.

ZeroMQ.

Discussing game design and ideas.

Workload:

Everyone: wrote in document.

28/01/2016 Thursday

Undocumented work hours (18:50 - 23:59).

Discussing and writing project plan.

Workload:

Everyone: wrote in document.

Spring meeting 19:00 05.02.2016

The meeting was postponed due to some miscommuncation about the meeting times.

The meetings are to start at 16:00 on fridays.

Jonas has double vote.

List:

Jonas

Henning

Christer

Eidså

Issues:

Pathfinding

Pathfinding AI

Level generation

Design (level generation)

Database implementation

Network

Console

Make

Data gathering

and how to use the data

UI

Document

Work:

Henning:

Christer: Make

Jonas: Pathfinding

Eidså:

Spring meeting 16:00 12.02.2016

List:

Jonas(16:30)

Henning

Christer

Eidså

Discussing different methods of generating level:

1. Use perolin noise (1d). place platforms on top points. Multiple passes. generate enemies on top points on next iteration etc.
2. Use perlin noise (2d/heightmap). use height thresholds to decide what is solid and what is empty space/enemies/spikes etc.
3. Use ANN to generate tile for tile, starting from start tile. Generate enemies using the same method after terrain is generated to make enemy placement feel better. ANN trained users.

Meeting 13:30 16.02.2016

Meeting with Mariusz. Discussed level generation methods and database security.

Hash passwords but it’s impossible to prevent unauthorized access, can only reduce the chances of it happening.

Experiment with Artificial Neural Network, Genetic Algorithm and Noise functions. Find out what works the best for our case. Combine them all. Example: Noise for layout, ANN for chunks and genetic for AI.

Sprint meeting 16:00 19.02.2016

List:

Jonas

Henning

Christer

Eidså

Discussing:

game mechanics -

level generation -

level mechanics -

- spikes(moving up and down, moving between 2 tiles, stationary),
- moving platform(any direction),
- spring (increases jump height),
- powerups (invincibility, double jump, jetpack, ranged weapon)
- crouch
- ledge grab
- platforms that disappears

enemy design -

- walking
- flying
- jumping/bouncing
- stationary
- ranged/melee
- pathing enemies (moving along walls, roof, etc)

New weekly limit. Change from 40 down to 35.

Resources:

Cloudberry kingdom:
this game has the level generation we want ish.
http://www.gamasutra.com/view/feature/170049/how_to_make_insane_procedural_.php?page=1