NTNU
Norwegian University of
Science and Technology

Video Games: Game AI

Remy Jensen

Master of Science in Computer Science
Submission date: June 2010
Supervisor: Alf Inge Wang, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

With the breakthrough of games into the mainstream and the increasing popularity of on-line gaming, creating a believable challenge to the player becomes increasingly challenging. The gamer no longer accepts challenge by just increasing the numbers of the same simple-minded enemies, they want an experience as if there was a human player controlling the non playing characters(NPCs).

The goal of this project is research how to create challenging and believable game AI that controls a games NPCs given the NPCs restricted actions. Various methods to control the NPCs will be tested and evaluated. The study will evaluate how believable the NPC controller is, as well as trade-offs that has to be made concerning the game's computational performance. This particular project aims for cooperation with a simultaneous ongoing project(http://www.idi.ntnu.no/education/prosjektoppgaver.php?p_id=974).

Assignment given: 15. January 2010
Supervisor: Alf Inge Wang, IDI

Abstract

The goals of this project was to learn the industry standards of what good and challenging game AI was. The author reviewed literature on the topic and had personal correspondence where the research questions was answered by professionals within the field. The availability of open literature and the openness of the professionals really helped with understanding the industry standards to game AI.

Using the information from the research a prototype system adhering to the industry standard was made with the intention of expanding it into an experimental prototype using unorthodox techniques to achieve the appearance of intelligence. By practical application of the methods learned it became apparent certain theoretical ideas was not optimally compatible with the provided framework. And a redesign of the conflicting module was nescecary.

The system implemented performed well within the industry standards. But no experimental prototyping of unorthodox methods could be made with the system due to lack of time to implement such features. A couple of optimality tweaks been discovered since the end of the implementation phase of the project and the author will keep theese in mind when continuing work on the system in the future.

Contents

Part I

Introduction

Chapter 1

Project Background

This chapter explains of the motivation and goals of this project. The first half of this chapter explains the motivation and the goals of the project. The last half explains the context of the project as well as stakeholders' interest in it.

1.1 Motivation

Video games have over the last few years become a multi billion dollar industry[1][6] even bigger than the music industry[9] With the breakthrough of games into the mainstream and the increasing popularity of on-line gaming, creating a believable challenge to the player becomes increasingly challenging. The gamer no longer accepts challenge by just increasing the numbers of the same simple-minded enemies, they want an experience as if there was a human player controlling the non playing characters(*NPCs*)

The graphical capabilities have with the current generation of game consoles and home computers reached levels many gamers considers "real enough." Also the industry's problems with the "power wall", limiting the speed of the processing unit clock has given rise to multi core processing units. The division between "real visuals" and "fake behavior" breaks the immersion in a game, this along with more cores to run sequential code on in parallel means intelligent agents running in their own thread might be the future of game programming.

1.2 Project Goal

The goal of this project is research how to create challenging and believable game AI that controls a games NPCs given the NPCs restricted actions. Various methods to control the NPCs will be tested and evaluated. The study will evaluate how believable the NPC controller is, as well as trade-offs that has to be made concerning the game's computational performance. This particular project aims for cooperation with a simultaneous ongoing project[1]

[1] http://www.idi.ntnu.no/education/prosjektoppgaver.php?p_id=974

1.3 Project Context

This project is a master thesis at the *Norwegian University of Science and Technology(NTNU)*. It is a part of the game technology research program at the *Department of Computer and Information Science(IDI)* under the *Faculty of Information Technology, Mathematics and Electrical Engineering(IME)*.

1.4 Stakeholders

Following this is a list of stakeholders, and their concerns regarding the project.

1.4.1 Author

- Remy Jensen

The product will be worked on also after the master thesis, and presented at a competition. The concern of the author regarding the product is the product's ability to give a fair and fun challenge to the player. The quality of the documentation, on which the author will be graded upon, is also important.

1.4.2 Course Staff

- Alf Inge Wang - Supervisor
- Meng Zhu - Assistant supervisor

The course staff is concerned about the academic value of the thesis. This means the quality of the report and the documentation must be high enough so that further research within the topic research can be derived from them.

1.4.3 Cooperation Group

- Kjell Ivar Bekkerhus Storstein
- Kjetil Guldbrandsen

The cooperation group's concern, in regard to this particular thesis, is mainly the quality of the product and it's ability to showcase a concept in an entertaining manner.

Chapter 2

Research

This chapter the Research Questions the author wish to be answered, as well as the methodology to achieve this.

2.1 Research Questions

RQ 1: How is the game AI implemented in modern games?

- **RQ 1.1:** What methods and algorithms, not typically considered part of traditional AI, are important for a good game AI?
- **RQ 1.2:** What traditional AI methods and solutions are used in game AI?
- **RQ 1.3:** When and where are traditional AI methods used in game AI?
- **RQ 1.4:** How important are traditional AI methods to game AI?
- **RQ 1.5:** How much leeway is there for implementation of AI methods that requires a lot of computational resources?

RQ 2: How can the answers from the above research questions help creating a better game AI?

- **RQ 2.1:** What are the performance issues connected to a game AI and what are the trade offs usually taken?
- **RQ 2.2:** Which methods from traditional AI, that are currently uncommon in game AI, could enhance the gaming experience?
- **RQ 2.3:** How much must the game AI know about the game world to give a good gaming experience, which parts must be omniscient?

2.2 Research Methodology

Chapter 3

Development

This chapter explains the method and tools used for the project.

3.1 Development Method

The development method will be a modified version of the waterfall method. The waterfall method is a well known development method and is described by Braude[2] and others like this:

1. *Requirements analysis* - A description of how the system is supposed to work when it is finished. These are a set of written rules the system has to satisfy.
2. *Design* - The creation of a plan for the software solution. Includes high level architectures as well as low level algorithms to be implemented into the system.
3. *Implementation* - The execution of the design plan also known as the coding of the architecture.
4. *Integration* - Assembly of the system's parts into a whole, as the implementation can occur in parallel on different modules of the system.
5. *Testing and debugging* - Testing the system through formal tests the system must succeed at, as well as debugging of the system as a whole after integration.

This project will adopt a modified version of the waterfall method where one can go back to earlier steps after testing and redesign, reimplement and reintegrate parts or the whole system if tests were not satisfactory.

3.2 Development Tools

This is the various tools used in the project and a brief description of each.

MiKTeX 2.8 A tool set of TEXfor Microsoft Windows. TEXis a typesetting language for technical writing popular in academia. It was designed especially for mathematics writing.

TeXnicCenter 1.0 An IDE for the TEXtypesetting language. It uses MiKTeX or TeX Live distributions. TeXnicCenter provides an overview of the TEXproject as well as code completion and easy compilation of the whole project to a single output file.

Unreal Development Kit January-April Beta A free version of the Unreal Engine and tools for visual editing of assets and scripts.

Visual Studio 2008 An IDE for several programming languages, including the C programming language family.

nFringe A Visual Studio 2008 language support plugin for UnrealScript, the scripting language used in the Unreal Engine.

Tortoise SVN A free subversion client for Windows. Used for version control.

Part II

Prestudy

Chapter 4

The Current Situation of Game AI

Game AI has a very broad and loose definition. Ranging from the popular definition implied at Wikipedia[19] being the limited sub set of problems centering around action selection and pathfinding, to the extremely broad defenition of "The AI part of a game is everything that isn't graphics (sound) or networking..."[12]. The defenition that includes animation and character locomotion as part of game AI is also increasingly geting recognition.

4.1 Traditional AI Game AI intersection

Traditional AI and game AI have generally little in common right now with only a handful of methods shared between the two fields. These fall on the long used algorithms of A*, Finite State Machines and decicion trees and their variations. The overall consensus is that in main stream games the traditional AI methods is simply too CPU and memory intensive to be of any use. There is allways the odd title that is both main stream and dares using unorthodox technology for its game AI.B.5;

Chapter 5

Pathfinding

The search for an optimal path between two points is essentially a search in a graph. The pathfinding function explores nodes adjacent to the current node until it reaches it's goal. This chapter explores searching algorithms used in pathfinding and the different kind of graph structures used to represent the game world.

5.1 A* Searching Algorithms

There are several well established algorithms for searching through a graph. Games with few nodes and a not too complex graph might use Dijkstra's algorithm to handle navigation, but the by far most widely used graph search algorithm in video games one is the A* algorithm.

The A* algorithm is an extension of Dijkstra's algorithm. In addition to evaluating the the path-cost($g(x)$), it also evaluates an heuristic estimate from the node evaluated to the goal node($h(x)$). Thus using a total cost($f(x)$) = $g(x)$ + $h(x)$ for it's evaluations. It follows the following process:

1. Take the node with the lowest $f(x)$ from the priority queue.
2. Update f and h for it's neighbors and add them to the priority queue.
3. Repeat 2 until goal is found or there is no nodes left.

The challenge in A* is finding a good formula for $h(x)$, but the Euclidean distance works pretty well as a staring point. Do not try to use the Euclidean Distance squared to serve computation power on the skipped square root since this will make the search a greedy best-first-search on longer routes, essentially ignoring the $g(x)$ focusing only on $h(x)$ in the computation of $f(x)$.[10]

5.2 Graphs

Purely 2D games can get away with just dividing the surface into a grid, usually the size of the smallest character moving around, and make each tile of the grid a node. In a 3D environment this simple setup with an added dimension(i.e. cubes) would make traversing the graph prohibitively expensive. Games with a 3D environment, where the characters

mostly move around on the ground, need to find other ways, more efficient ways to represent the area available for movement to a searching algorithm.

5.2.1 Way Points

A common way to make the graph is to make use of way points placed on walkable terrain. Each way point is connected to one or more other way points, making the graph to be searched, this is shown in figure 5.1. They are usually placed at junctions and points of interest to the NPC and with a semi-regular distance between these key points. A variant of the way points is giving the way points a radius, thus making them circles.

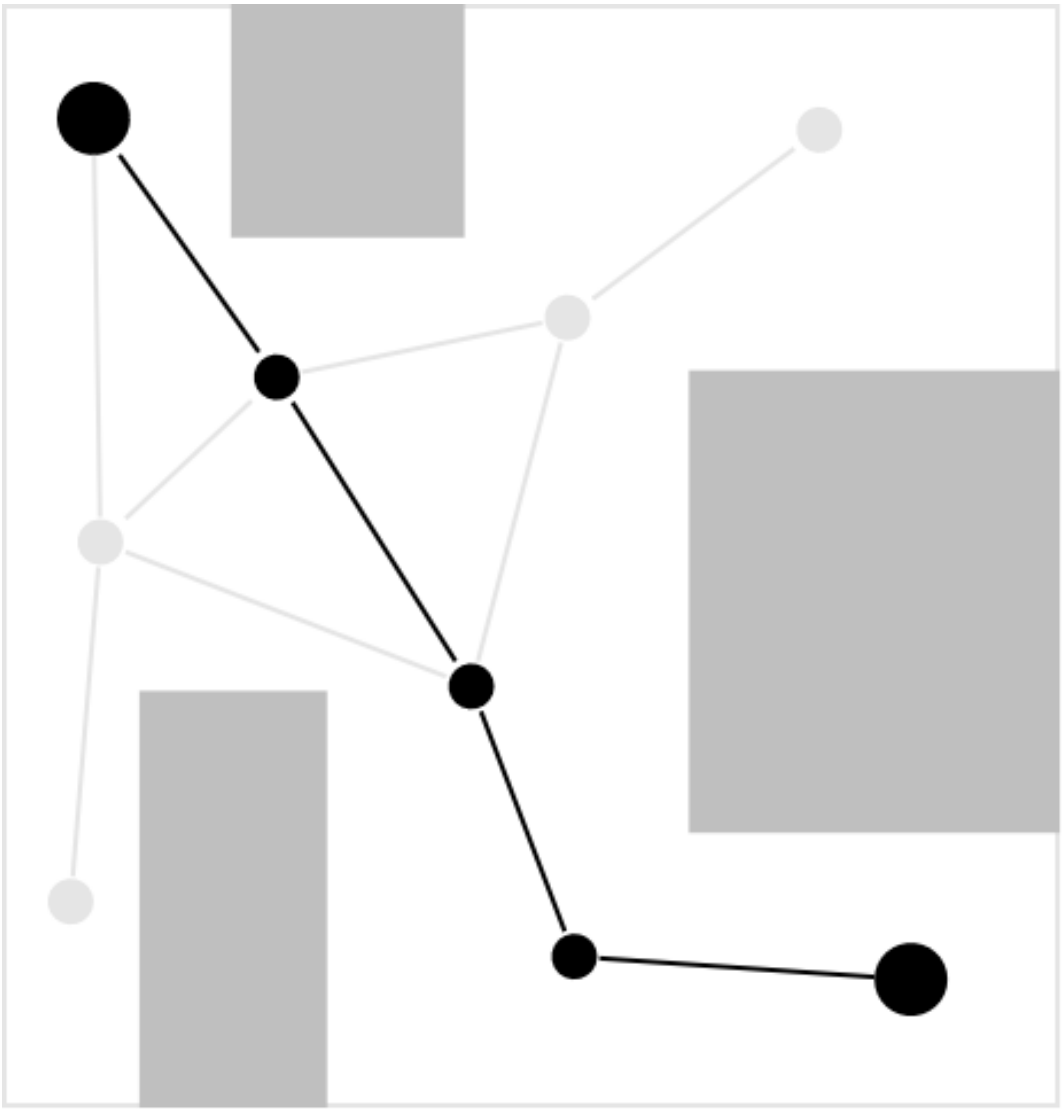


Figure 5.1: A simple way point graph. The path between the two enlarged points is darkened.

5.2.2 Navigation Mesh

Navigation meshes divides the movable area into convex polygons and makes each polygon a node in the graph. The connections in the graph is then decided by shared edges. If two polygons shares an edge, then the two are connected in the graph. A possible navigation of the map shown in figure 5.1 is shown in figure 5.2.

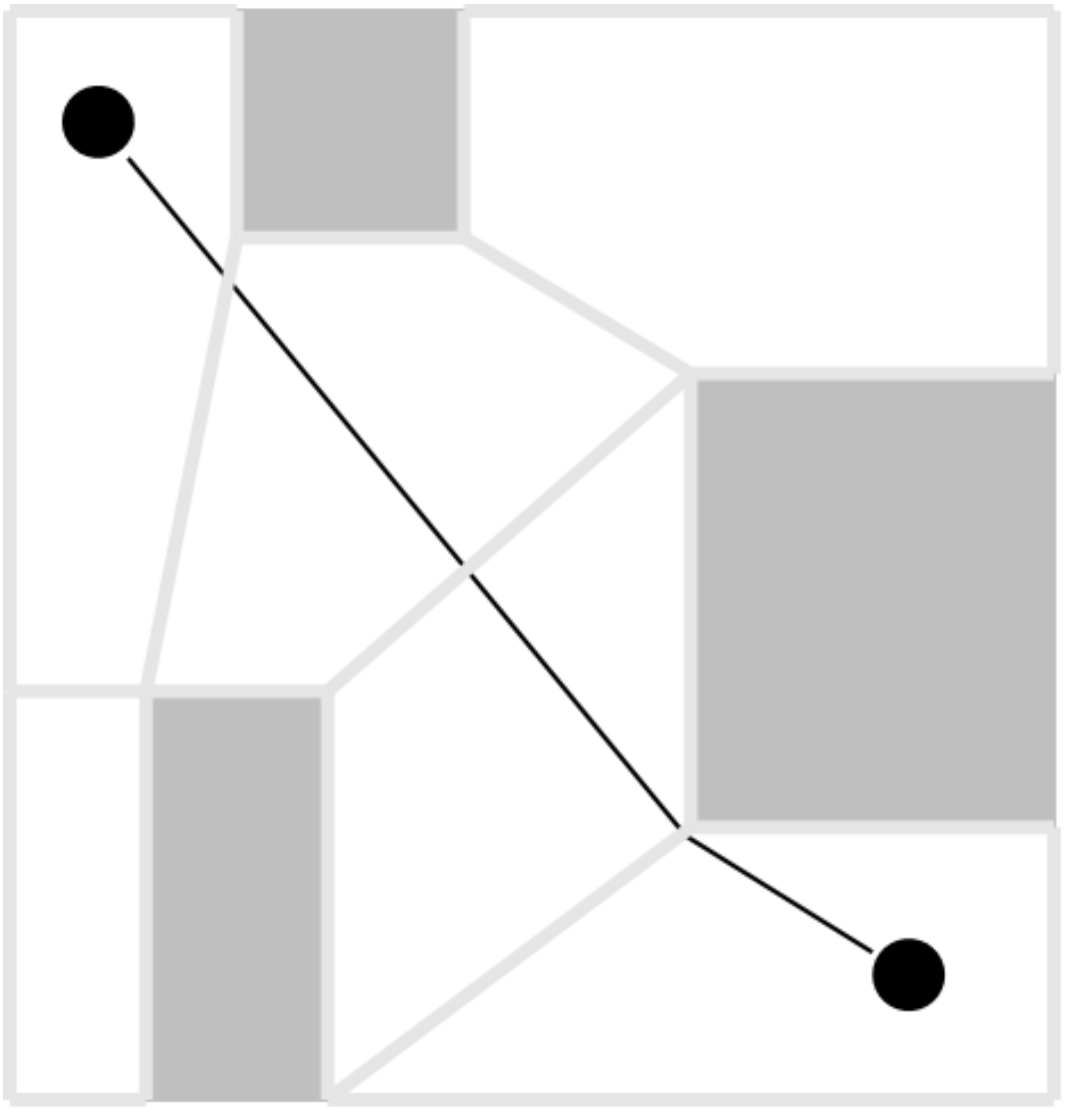


Figure 5.2: A simple navigation mesh graph. The path between the two points is darkened.

These navigation meshes can be generated to a near optimal level automatically[18] using the following approach:

1. Start with world geometry fitting the criteria *walkable*, this is usually those polygons facing more or less up, and make those your nodes.

2. Merge nodes into the minimum amount of convex polygons possible with the Hertel-Mehlhorn algorithm.
3. Run the $3 \rightarrow 2$ algorithm to find and split nodes that might be inefficient. $3 \rightarrow 2$ merging is explained extensively on page 177 in the book AI Game Programming Wisdom.
4. Run another pass with the Hertel-Mehlhorn algorithm to merge the split nodes into more optimal convex polygons.
5. Cull nodes with a surface area less than a set minimum to save memory resources used on trivial nodes.
6. To handle superimposed geometry, subdivide recursively nodes that intersect with the geometry until the surface area of the trivial node is reached and discard the whole node if it intersects.
7. Run a final pass with the Hertel-Mehlhorn algorithm to merge the nodes left from the subdivisions into a minimum amount of convex polygons.

Chapter 6

Action Selection

A central part of game AI, and the part most people probably think about when they hear or read ”Game AI” is how the NPCs choose which action to what to do next. This chapter takes a look at the different methods and algorithms, commonly used in games, that makes the NPCs plan actions that interact with the game world beyond the purely cosmetic.

6.1 Decision Trees

Decision trees is a decision tool using a tree-graph that structures the decisions available and their consequences in relation to satisfaction and cost. They are modular, easy to create and easy to visualize and understand during design as shown in figure 6.1. The decision tree consists of decision nodes as the non-leaf nodes and action nodes as leaf nodes. Each decision node in the tree is an if-then decision which is followed until an action node is reached. The action described in the action node is then carried out by the NPC. Decision trees can be learned relatively fast through methods in machine learning.[7][8]

6.2 Finite State Machines

State machines is a structure where when a character is within a state, it will perform a set of tasks describing a behavior until a conditional check initiates a transition to another state. A Finite state machines(*FSMs*) contains a finite (i.e. countable) number of such states and a network of transitions between the states. FSMs has traditionally been hard coded, and while new methods to handle FSMs exists, hard coding is still a common approach to FSMs.[7][4]

The big problems of FSMs is little code reusability, and non-modularity. Also because transitions often are coded directly into each state, it becomes difficult to read a complex FSM.

Hierarchical Finite State Machines

An attempt to solve the problems with FSMs is by arranging the FSM into a hierarchy. One such approach is to consider each state its own FSMs. By using this structure, small modular FSMs is encouraged with each modular FSM being grouped into a super-state that

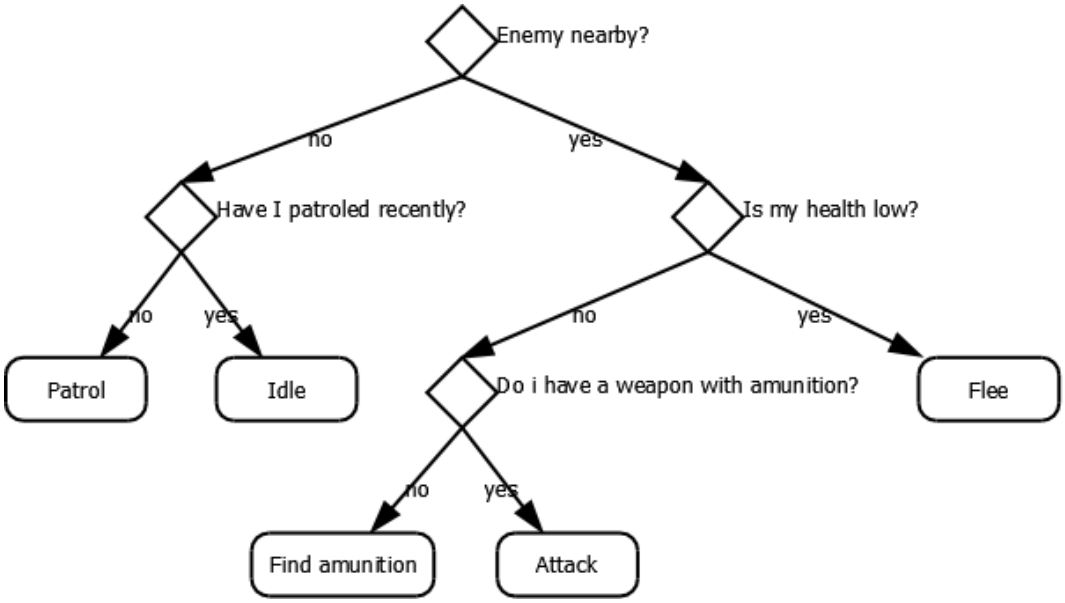


Figure 6.1: An example of an decision tree describing simple enemy behavior in the player's presence.

keeps its own transitions to states outside the modular FSM.[7] Another approach is to loosely collect sets of states that share the same transition logic together in super states. This second approach have a focus on reusability over modularity. Any hybrid version of this can also be made to achieve a compromise of wanted modularity and reusability.[3]

Chapter 7

Unreal Development Kit

The features the Unreal Development Kit(*UDK*) provides of interest to this project is navigation handling and means to process decisions and make them into actions. This chapter gives an overview over the features the UDK provides for game AI programming.

7.1 Navigation

The UDK provides means two means to represent the game world as a graph to search trough, *Way Points* and *Navigation Meshes*. The way points system is referred to as the old system and implicitly discouraged in favor of the use of the navigation meshes.[15] This project will therefore focus on the navigation mesh features the UDK provides for navigation.

7.1.1 Navigation Meshes

The features of the navigation meshes in the UDK allows competent A* path finding on a navigation mesh with no in-depth knowledge about the structures and algorithms involved in the path finding. These features includes:

Pylons Adding these actors to the game world will automatically generate a navigation mesh ion a radius around the pylon upon the next run of the built navigation mesh tool within the UDK editor.[17].

Navigation Handle In order to make a controller path find on the navigation mesh it needs to:[16]

- implement interface_navigationhandle.
- have a navigationhandle component accessible somewhere.

The *Navigation Handle* provides all the functions to perform search for and follow a valid path from the current position to a goal position. Additionally the Navigation Handle contains functions that check for the validity of potentially reachable points.

Path Constraints are the components that define the function *f(x)*. The path constraints modify both the heuristic function *h(x)* as well as the actual cost *g(x)*.

Goal Evaluators determines which nodes are considered a *goal* node as well as setting up the search and saving the path found so far.

Relic gives a list over the provided path constraints and goal evaluators on their reference pages.[14]

7.2 Controllers

In UDK there is a division between the component interacting with the game world and the component that makes the first perform actions on said game world. The *Pawns* are the physical actors in the game world, they are directly affected by the game world through the physics engine, collisions, taking damage etc. Each *Pawn* has a non-physical actor, the *Controller*, that makes decisions for what a pawn does. A controller can be an interface for a human player to control a character through input from game pads, keyboards etc. or the computer controlling the pawn through game AI methods. This system was designed so the same AI controller is capable of controlling pawns with different abilities.[11]

7.3 UnrealScript and Finite State Machines

UnrealScript is a high level scripting language with syntax and behavior similar to languages such as Java and C#, with some quirks and extra features that differentiates it from the two languages mentioned. The biggest difference and the one worth paying attention to from this project's point of view is that FSMs are supported on the language level with UnrealScript, and with UnrealScript3 state stacking.[13]

States in UnrealScript 3 have several features that makes it more than a simple FSM. They have a hierarchical structure. Each UnrealScript state contains labels, that essentially functions as a case statement within the state. One can then use the "Goto('label name')" function to continue executing from the label given as input.

They have inheritance, one can make several states similar to each other and have them all extend a super-state containing code and labels common to all sub-states. Using a weapon may have the common functions of drawing the weapon. and reloading the weapon, with only the firing behavior being different. A super state called "Use Weapon" may then contain logic for drawing and reloading the weapon and an label for firing the weapon containing code that will be overridden. States can then inherit the common functionality and override the fire weapon accordingly.

The third major feature to UnrealScript states is the state stack. One can now with ease interrupt a state with a high priority behavior just by pushing it on top of the state stack. When the high priority state terminates and pops from the state stack, the previous state it interrupted continues execution of it's own code from where it was interrupted.[13]

Part III

Own Contribution

Chapter 8

Requirements

This chapter describes the requirements to the system and will be used as guidelines for the design of and implementation of it. The following sections describe what functional and non-functional requirements the system has.

8.1 Functional Requirements

This section describes the features required from the system. The following tables describes each functional requirement with an *ID*, a short *description* of the requirement and the *priority* of the requirement. The ID is a unique identifier of the requirement for easy identification. The description explains the content of the requirement. The priority shows the relative importance of the requirement to the rest of the requirements. The priorities are indicated with the *H*, *M* and *L* which refers to *High*, *Medium* and *Low*.

ID	Requirement description	Priority
NPCR1	NPCs shall ensure it's own survivability in a game world not yet manipulated by the player.	H
NPCR2	NPCs shall path find without getting stuck.	H
NPCR3	NPCs should showcase human decision making behavior.	M

Table 8.1: NPCR: Overall NPC behavior requirements.

ID	Requirement description	Priority
PR1	Peasant characters should gather resources for future use.	M
PR2	Peasants characters shall "'report"' the player when noticing the player character.	H
PR3	Peasant characters should execute flavor actions when no emergency is present.	L

Table 8.2: PR: Peasant character behavior requirements.

8.2 Non-Functional Requirements

The non-functional requirements do not directly dictate the system's demanded functionality, but serve as guidelines for the design of the system.

NFR 1 The game AI shall not reduce the frame rate to below 30 FPS.

NFR 2 The game AI should be designed within the game AI framework provided by UDK.

Chapter 9

Architecture

This chapter explains the initial architecture used to plan and implement the game AI. This chapter starts by identifying the framework provided by UDK followed by the initial overall class-level design rounding off with the initial design of a FSM using the knowlege from section 6.2.

9.1 UDK Framework

UDK provides a distinct division between the visual character and it's controller through the *Controller - Pawn* arrangement, which implies that every *Pawn* should have a *Controller* in order to do something interesting. Likewise the *Controller* needs a *Pawn* to make an impact on the game world. This project is also done in cooperation with an another project (section 1.4.3) which provides an additional framework to work within, mostly in regards to what actions the *Pawns* has that interacts with the game world.

9.2 Class Architecture

The initial overall class architecture of the part of the system that is relevant to this project is relatively simple. The class-level architecture is shown in figure 9.1.

The Pawn classes consisting of *ApocPawn* and it's subclasses mostly contain attributes relating to the status of the *Pawn* and logic how these are updated. The subset of attributes describing the vitality of the *Pawn*, such as *Health*, *Thirst* etc. will be referred to as *stats*, which is short for statistics. The enemy classes are supposed to change their speed depending on their status, and contain functions that change this when certain stats are changed. The *ApocPeasant* class also contain functions that estimate the time until certain *stats* reaches 0 if the *ApocPeaant* does nothing to improve these. The *Pawn* classes being extended by *ApocPawn* are shown in the architecture to show which classes are polled by the controllers and how they connect to the UDK framework.

The *ApocEnemyController* is handles the navigation logic as well as provide the framework for the action selection. This class is responsible for the pathfinding from the *Pawn's* current position to the location provided to the *ApocEnemyController*. The *ApocEnemyController* also handles the search for the closest instance of a given *Actor*. The navigation logic in this

class address the NPCR2 requirement described in table 8.1. The *DecideWhatToDoNext* state contain only debug code and is in practice acting as an abstract state providing a framework the subclasses to implements.

The *ApocPeasantController* handles the action selection for the *ApocPeasant* pawn. This class realized the *DecideWhatToDoNext* state with a *decision tree* using what was learned in section 6.1. The *DecideWhatToDoNext* state address the NPCR3 requirement described in table 8.1. The remaining behavior logic realized with an FSM.

9.3 Finite State Machine Architecture

The initial FSM in *Apocalypse* was designed around the *DecideWhatToDoNext* state acting as a hub the FSM always returns to when an behavior is completed. The initial FSM design can be seen in figure 9.2.

The *GoHome* state makes the *ApocPeasant* go to his designated home and unload carried goods as well as replenish the *ApocPeasant's stats* if the conditions are right. This behavior address the NPCR1 requirement described in table 8.1.

The *GatherWater* and *GatherFood* states makes the *ApocPeasant* search for and collect the given resources, water and food respectively, and then go home to store the resources at home. This behavior addresses the NPCR1 and PR1 requirements described in the tables in chapter 8.

The *AttendMass* state makes the *ApocPeasant* search for the closest church and activates a function that fully replenishes the *ApocPeasant's stats*. This behavior address the NPCR1 requirement described in table 8.1.

The *Strolling* state makes the *ApocPeasant* chose a random point on the map and walk to it, simulating the peasant taking a stroll to enjoy the scenery. This behavior is purely a flavor behavior to address the PR3 requirement described in table 8.2.

The *RunAway* state makes the *ApocPeasant* run away from a perceived danger and report the danger to the nearest church. This behavior addresses the NPCR1 and PR2 requirements described in the tables in chapter 8.

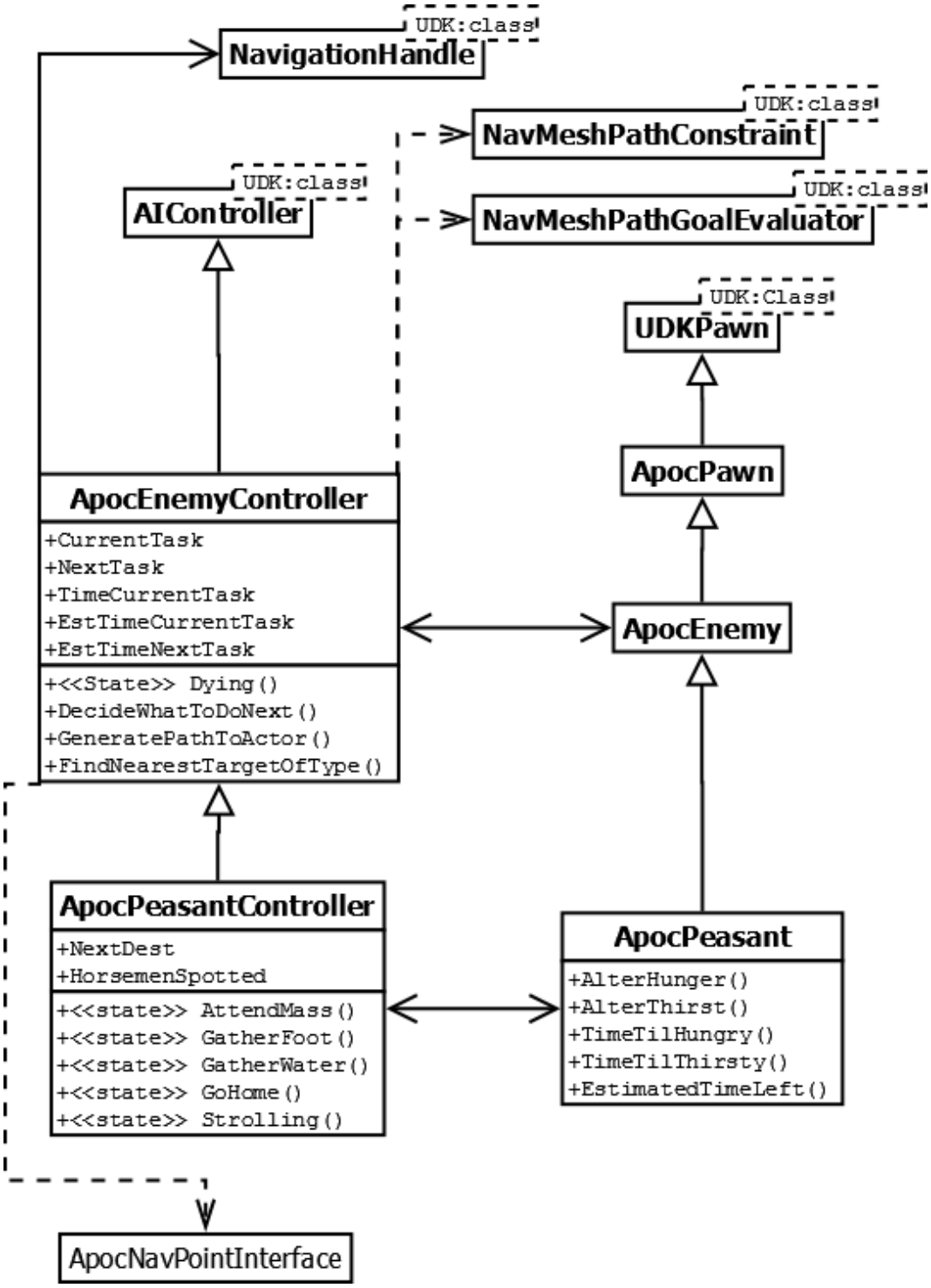


Figure 9.1: The part of the Apocalypse system relevant game AI. The focus of the system will be on the two controller classes *ApocEnemyController* and *ApocPeasantController*.

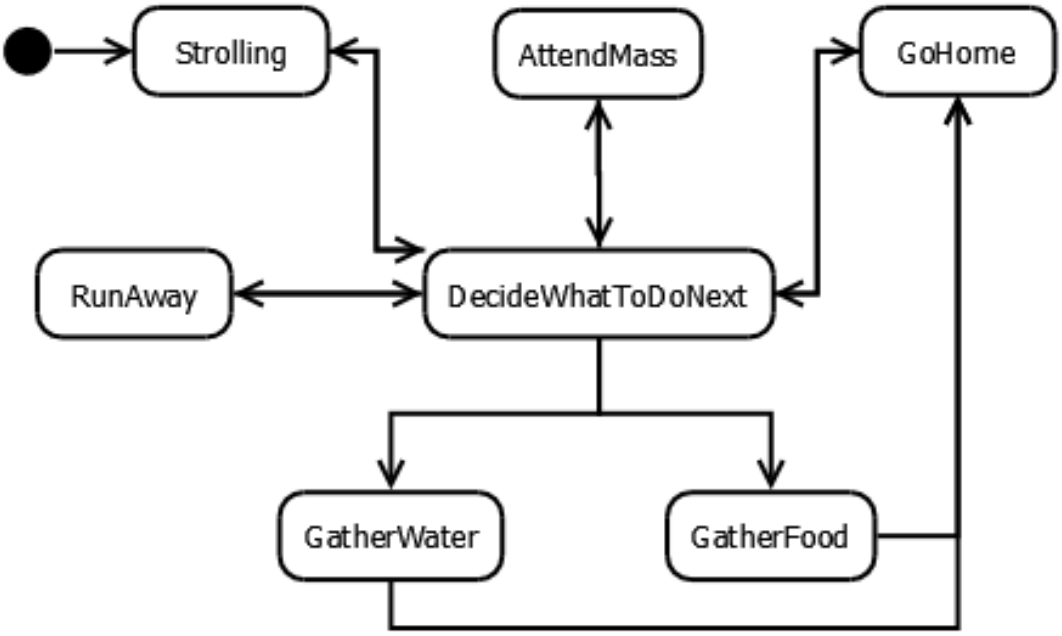


Figure 9.2: The initial design for the ApocPeasantController state machine.

Chapter 10

Path Finding

This chapter describes the implementation of the path finding logic and support functions related to the navigation logic. The *GeneratePathToActor* function is explained in the first section followed by a description of the support functions. The creation of the navigation mesh is just a matter of using the UDK editor to place *Pylons* in the game world and automatically generate a mesh. This process is explained in section 7.1.1 and thus will not be explained here.

10.1 GeneratePathToActor

The *GeneratePathToActor* function is the function in charge of generating a path to the desired location given as an input from the current position of the *Pawn* and return with the vector describing the next destination in the path generated. The *NavigationHandle*, *NavMeshPathConstraint* and *NavMeshPathGoalEvaluator* classes and their subclasses are all *native* as of the *UDK April Beta* and thus locked from insight and inheritance. This makes it so the only means of manipulating how the pathfinding works is through the use of functions provided by these classes. The *GeneratePathToActor* therefore mostly contain code handling exceptions generated by these functions. The full source code of *GeneratePathToActor* is found in the appendix C.1.

The function first checks if the *Pawn* is stuck when it should be moving and handles this through the use of the *TardCounter* value and the not yet integrated *ObstructionJump* function. The ObstructionJump and it's intended function is described in section 10.2. The function then checks for a direct route, if the *Pawn* can see the *goal location* from it's current position the *Pawn* will walk directly to the *goal location*.

After the *TardCounter* and direct route checks a NavigationHandle is initiated, if not already initiated, and a A* search is performed on the navigation mesh using the a set of *Path Goal Evaluators* and *Path Constraints*.

The set of *Path Goal Evaluators* and *Path Constraints* describe the heuristic function the A* search operates with. The *ApocEnemyController* by using the *NavMeshGoal_At Path Goal Evaluators* will terminate and return true when the *Pawn* reaches the *goal location* within a given distance. The *Path Constraints* change according to the situation. Under normal conditions the constraint is *NavMeshPath_Toward* which is constraint describing the

Euclidean distance heuristics. If an enemy is spotted the *Path Constraint* becomes the *NavMeshPath_WithinDistanceEnvelope* constraint, which applies a penalty to nodes within a certain radius range from a point. The point from which to calculate th envelope is set to the location of the spotted enemy. The max range is a large number, that effectively is infinite within the game world. The minimum range is set to *ApocPawn.AVOID_DISTANCE* which currently is set to *300.0f*. This effectively makes a heuristic high cost field around the enemy thus making the *Pawn* try to avoid the enemy.

10.2 Navigational Support Functions

The functions described in this section are important to the navigation logic and is either used directly or indirectly by the *GeneratePathToActor* function.

10.2.1 ObstructionJump

The *ObstructionJump* function will make the *Pawn* move in a jumping motion to try getting out of a situation in which it is stuck in the world geometry. The idea and implementation is taken from Trendy Entertainment's Dungeon Defense [5] which is a showcase provided with open source code for the UDK.

10.2.2 FindNearestTargetOfType

FindNearestTargetOfType is used before the call to *GeneratePathToActor* to find the *goal location* passed to the *GeneratePathToActor* method.

Chapter 11

Action Selection

This chapter describes the implementation of the action selector as well as the actions themselves.

11.1 Finite State Machine

The FSM is mostly like the design described in section 9.3 and the states that realizes will not be described again here. The implemented FSM can be seen figure 11.1, the differences in the FSM from the designed FSM will be explained below.

The biggest difference is the change of *DecideWhatToDoNext* from a state in the design to a function in the final implementation. This partly because the pushed state did not support forced change to the desired state without a hack, but also because of the discovery of the latent functions provided by *UDKBot*. These latent functions provides a framework to run the code within safely without it interfering with the physics of other world variables. The choice was made to approximate this design in order to make an eventual move to *UDKBot* as a superclass easier.

Each state is also implemented as a collection of states in a hierarchical manner with sub states pushed for sub behaviors such as *Walking* and entering buildings. This ensures maximum reusability of the *Walking* state logic.

11.2 Decision Tree

The decision tree is implemented considering the most critical state first. The tree is therefore imbalanced as seen in figure 11.2.

The *ApocPeasantController* evaluates it's stats in the order of health, the minimum of food and water, the water stock held and lastly the food stock.

The "Compare mass to X" choice checks if it is a shorter amount of time left until mass compared to how long it takes to complete "X" and chooses the action that takes the least time.

The weighted random choice gives each action a heuristic based on the status of the stats the action affect and randomly chooses one of the actions with a higher heuristic representing a

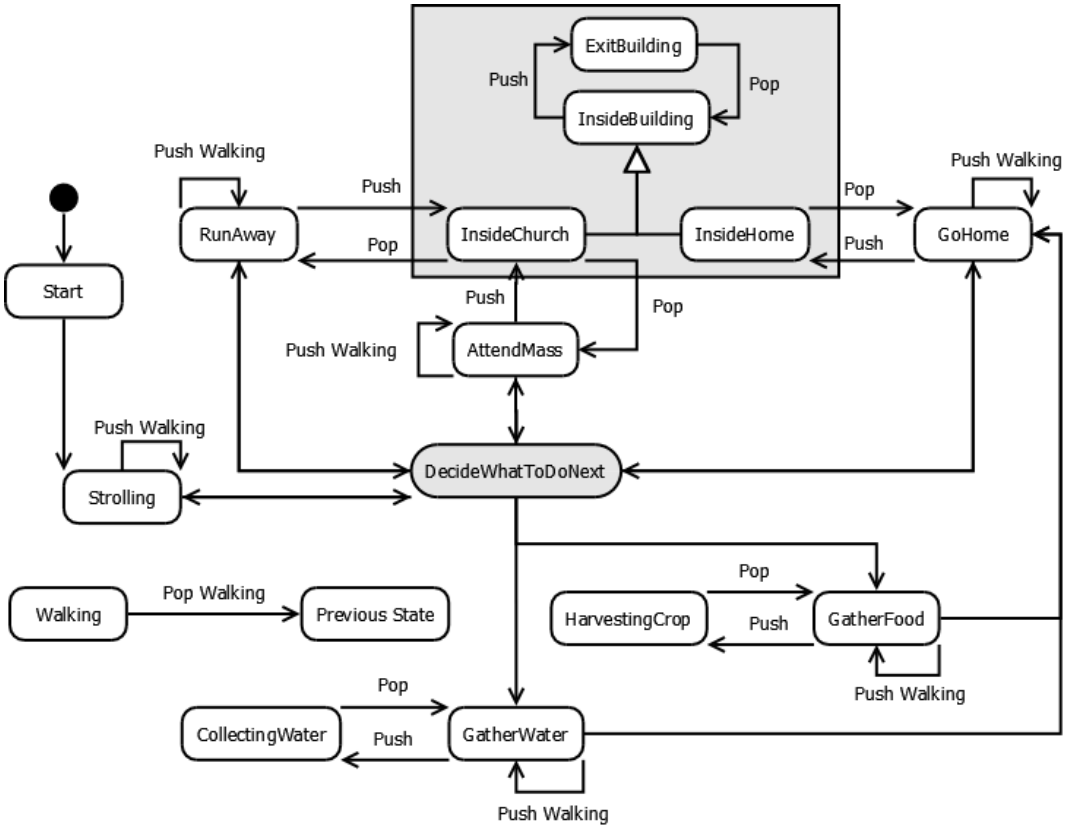


Figure 11.1: The states of an ApocPeasantController.

higher chance to be picked.

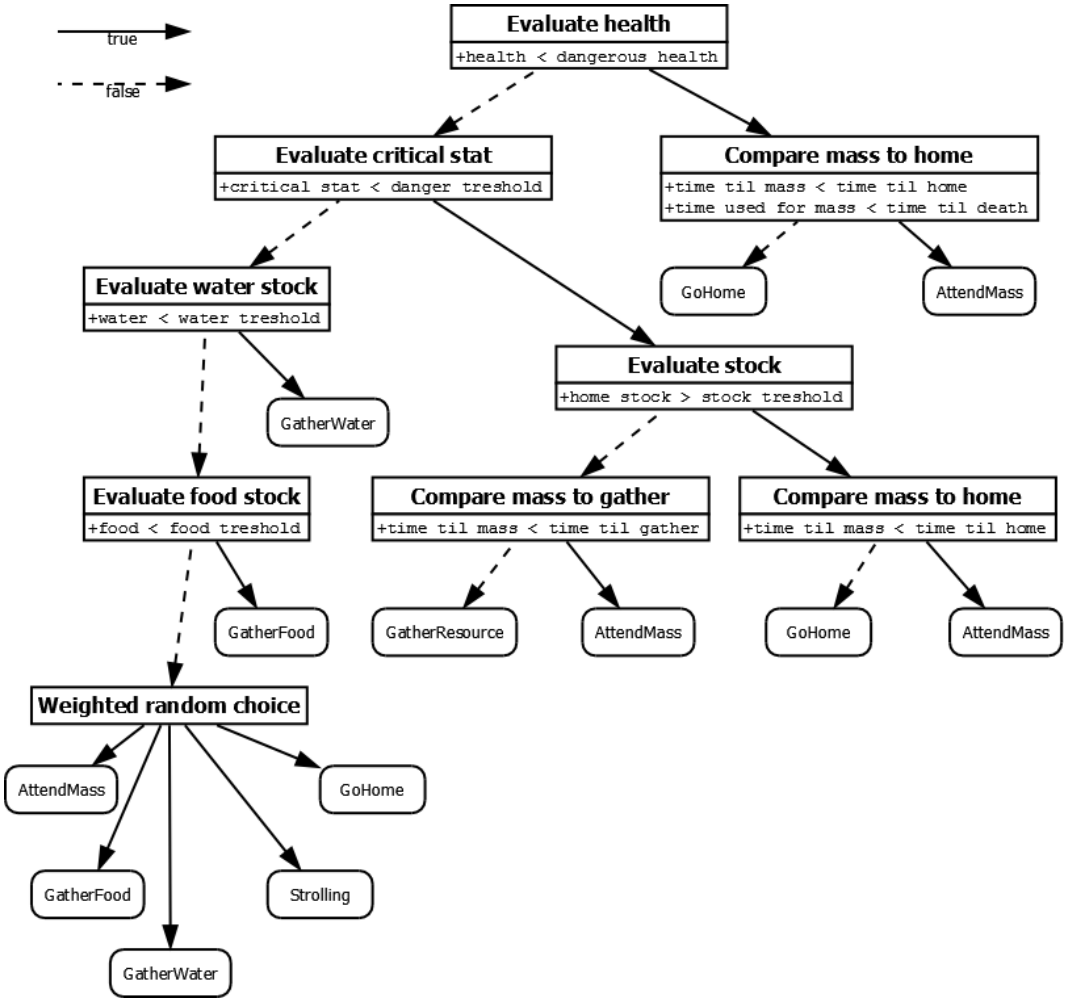


Figure 11.2: The decision tree of an ApocPeasantController.

Chapter 12

Results

In this chapter the results of the implementation is presented. There is not much to say about successes beyond that the system performed as required within the requirements given. The following sections will explain what went well and what went wrong within the confines of each requirement.

12.1 Partially Fulfilled Requirements

The table shows which requirements was partially fulfilled.

ID	Requirement description	Priority
NPCR3	NPCs should showcase human decision making behavior.	M

Table 12.1: Partially fulfilled NPC behavior requirements.

The NPC does not show the optimal human behavior, it does however behave like a human that ”goes easy” without ever adapting to become better if the human player becomes better. This requirement is therefore considered partially fulfilled.

12.2 Unfulfilled Requirements

The table shows which requirements was unfulfilled.

ID	Requirement description	Priority
NPCR2	NPCs shall path find without getting stuck.	H

Table 12.2: Unfulfilled NPC behavior requirements.

NPCs occasionally get stuck in their houses, because of this NPCR2 is then considered unfulfilled.

Part IV

Evaluation and Discussion

Chapter 13

Evaluation

This chapter provides the author's own evaluation of the project. This chapter starts with an evaluation of the development method used and the evaluation of the system's design. The chapter rounds off with the answering of the research questions.

13.1 Development Method

Given the that the project had to work within certain restrictions set by the cooperation group. Since the cooperation group was prototyping a game new ideas for fun game play was constantly pitched, tested and implemented. The modified waterfall method worked well for this kind of project since steps back was taken when needed to implement and fit the rest of the system to a new feature.

13.2 Design

The design of the system was satisfactory according to the requirements set, but could be better from the when considering the the design with the research questions in mind. The design adhered to what the industry standards where, and performed well within those conditions. In regard to how game AI could be improved, the author could have been more experimental. In the end there was too little time compared to the vision of the author to implement such an experimental system while still keeping the system operationable and not hindering the cooperation group.

13.3 Research

This section will look the research questions set in section 2.1 and see what answers the research provided.

13.3.1 RQ 1: How is the game AI implemented in modern games?

To answer this question a questionnaire was sent out to game developers profiled for their work on Game AI. Using the answers from the questionnaire and information found in literature the following trends can be identified in regards to game AI in modern games.

Game AI is considered all methods and algorithms that immerses the player into the game world, making the player believe he/she is a part of the game world. This includes custom animations as well as NPC action selection and movement within the game world.

The intersection of the methods used in game AI and methods common to traditional academic AI is a rather limited set of algorithms and methods. Mainly because most traditional AI methods are too costly in terms of CPU time and memory usage. With as little as 10% of the total available CPU time budgeted to game AI and even less of the memory, real time neural networks and genetic algorithms is still a far way off.

13.3.2 RQ 2: How can the answers from the above research questions help creating a better game AI?

With the answers given in RQ 1 along with knowledge of the field of AI and machine learning, is may indicate that the focus on game AI should be along the lines of making the tools currently used in the field more versatile. FSMs have evolved from a flat structure into increasingly more versatile hierarchical structures using the same base technology, and further refinement of these techniques seems quite possible.

Parallelization also seems to be a field game AI should focus on. modern games' optimal usage of all the CPU cores lags behind hardware manufacturers adding core to the CPU chip. This may however face the issue of the memory wall.

Because of the nature of AI, it need to know a lot of things to make good choices. so the AI should know a little more if it gives a fun game experience.

Chapter 14

Conclusion and Further Work

This chapter wraps up the project with a conclusion and what future work of the project.

14.1 Conclusion

The goals of this project was to learn the industry standards of what good and challenging game AI was. The author reviewed literature on the topic and had personal correspondence where the research questions was answered by professionals within the field. The availability of open literature and the openness of the professionals really helped with understanding the industry standards to game AI.

Using the information from the research a prototype system adhering to the industry standard was made with the intention of expanding it into an experimental prototype using unorthodox techniques to achieve the appearance of intelligence. By practical application of the methods learned it became apparent certain theoretical ideas was not optimally compatible with the provided framework. And a redesign of the conflicting module was nescecary.

The system implemented performed well within the industry standards. But no experimental prototyping of unorthodox methods could be made with the system due to lack of time to implement such features. A couple of optimality tweaks been discovered since the end of the implementation phase of the project and the author will keep theese in mind when continuing work on the system in the future.

14.2 Further Work

Using chapter 12as a base a couple issues and points of improvement has been identified.

In order to improve the partially fulfilled functional requirement NPCR3. One may consider using another algorithm than a decision tree for the action selection. To improve performance a move to the changing the inheritance from the current AIController to the UDKBot class which provides a framework to implement *DecideWhatToDoNext* latently, distributing the computation of the *DecideWhatToDoNext* over several tics without interrupting the physics and collision parts of the UDK.

The *ObstructionJump* is currently not properly implemented. If implemented this may fulfill the NPCR2 functional requirement.

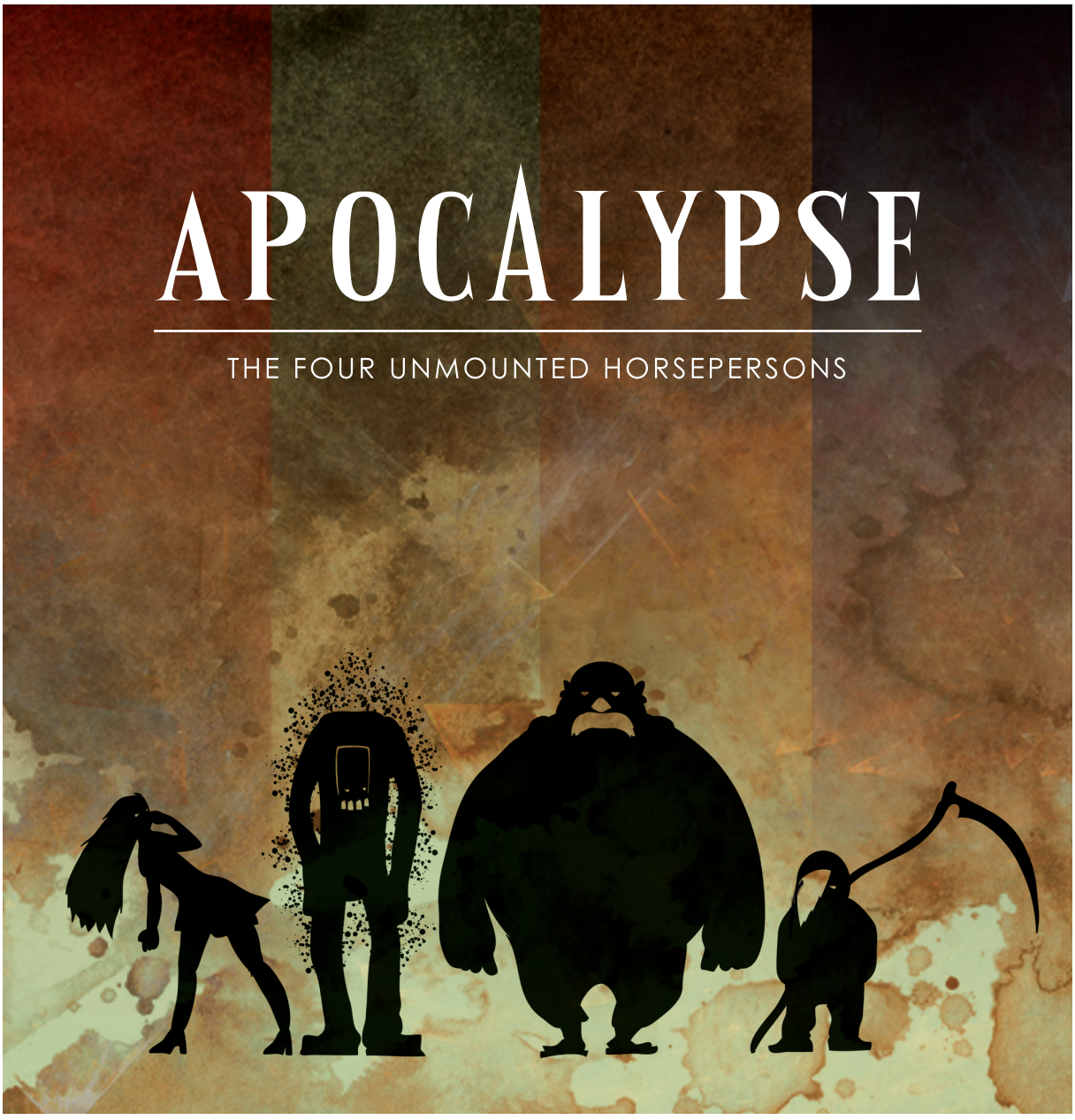
Part V

Appendix

Appendix A

"Apocalypse" Concept Document

1. Game name: Apocalypse: *The Four Unmounted Horsepersons*
2. Team name: HAX.EXE
3. Game genre: Strategy
4. Multiplayer support: Yes
5. Platform: Xbox 360



APOCALYPSE: THE FOUR UNMOUNTED HORSEPERSONS

Apocalypse is an action-oriented, real-time tactics game, where your ultimate goal is to corrupt mankind, thus bringing the End of Days. By taking command of the Four Horsemen, which ironically neither have horses nor are all men, you will plunge into the land of happiness and bliss, leaving only death and darkness in your wake.

The game is set in the Dark Ages, where lawful and God-fearing peasants happily harvest the fruits of the land. Peasants are strong and sound, attending every mass in the local church, gratefully praising the Lord for their good health, their wealthy crops and their pure and fresh water and air. This happy ecosystem is for you to destroy.

Your various unmounted horsepersons, have demonic powers related to their apocalyptic domain. "Pestilence" is the manifestation of sickness and plague, infecting men and livestock alike, leaving the lucky dead and the unlucky suffering horrendously on their deathbeds. The she-devil "War" possesses demonic beauty and immense wealth, corrupting the weak minds of men, turning brother on brother leaving villages in shambles and ruins. The mustachioed and fatally obese "Famine" has the appetite of a thousand hogs, consuming crops and livestock as he plods through the land, leaving the populace starved and weakened in body and mind alike. "Death" is the ultimate release for the plagued and starved peasants, harvesting souls to be sacrificed upon unholy altars in the eternal fires of Hell. These sacrifices enable the unmounted horsepersons to call upon the Seven Deadly Sins to aid their sinister work.

Unfortunately the Holy Church of Mankind stands in your way thwarting your every move, undermining your powers by the hands of their zealous priests. The priests can cure the sick, and remove the taint of corruption from the land. Also the priests seek to repel the unmounted horsepersons with their blessed auras. Their one weakness lies in their dependency on the peasants attending their holy mass, fueling their powers. The peasants also benefit from attending the mass as their curses are lifted and they recover their strength from the Holy Commotion. When peasants weaken in body and spirit, their faith and ability to attend church diminishes and they stop attending mass, weakening the powers of the priesthood and decreasing the

In-game screenshot showcasing the character Pestilence

holy aura surrounding the church. As the situation in the church deteriorates, the priests are blessed with increased speed and zeal, counteracting the onslaught of chaos. At the time of ultimate corruption, when no peasant is left to serve the holy cause of the church, the priests in their despair find the temptation of sacramental wine to be overwhelming, and the corruption is total.

The peasants lead a simple life, having only three major concerns: Gather precious food to store in their houses, get pure, pristine water from the surrounding wells, and attend holy mass. The unmounted horsepersons collectively possess the necessary skills to disrupt these activities. Each peasant has a combination of strengths and weaknesses making them more resilient or more vulnerable to different types of corruption. It is paramount that the unmounted horsepersons combine their strengths and abilities to destroy each soul.

Gameplaywise Apocalypse is a multiplayer game, intended for 1-4 players. where one player may control one, two or all four “hero“-characters. The game

APOCALYPSE: THE FOUR UNMOUNTED HORSEPERSONS

Concept art for the Famine character

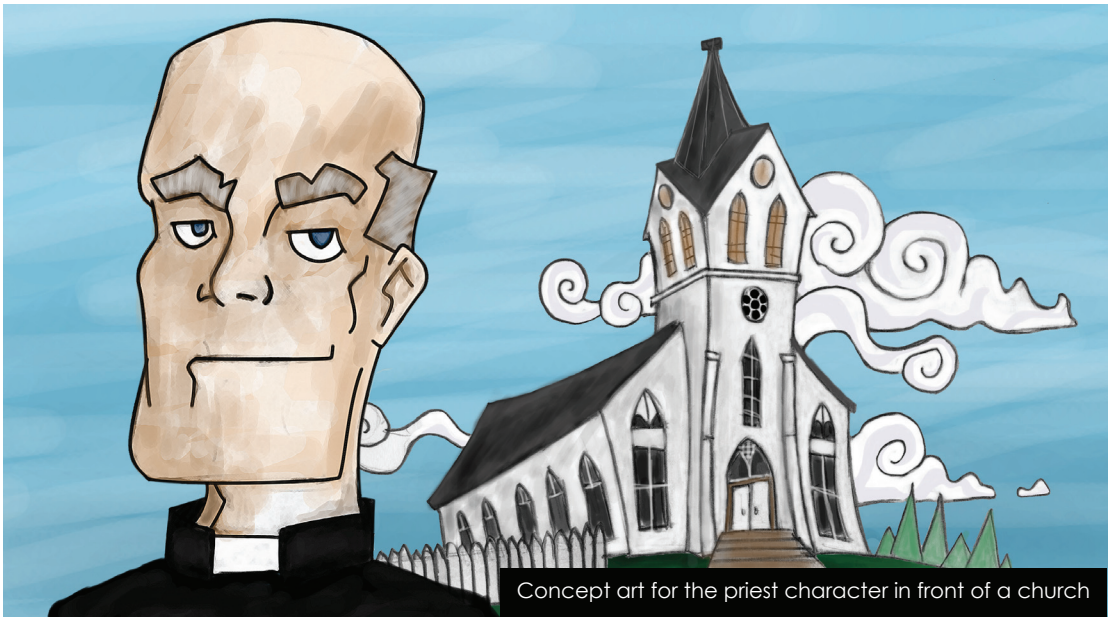
is not intended as a network-multiplayer game, but as a simultaneous hotseat game, where players may dynamically join and quit as the game passes. Depending on the number of players, different players will be assigned different unmonted horsepeople. Since all characters share the same screen, one is assigned leader of the group, and is targeted by the camera.

Leadership is easily transferred between the different characters, and the presence and direction of off-screen char-acters is visualised. The game is presented through a top-down camera as seen in many RTS-games, enabeling broad overview of all players and the surrounding environment.

The game features a campaign with several levels. Each level introduces the players to a peasant community under one or more church jurisdictions. A level is completed when all churches are completely corrupted. Levels start out in an overly happy, bright and idyllic environmentwith strong, bright colors, dan cing peasants, rainbows and singing birds. As the taint spreads the environment turns gloomy , the colors dims, birds die, the sky turns red and pigs start flying. The music changes from happy, medieval tunes, to heavy metal.

Despite of the dark theme of the game, all characters and the environment are presented in a highly cartoonish style, contributing to a comical and unrealstic look and feel.

Peasants have two major statistics: Health and zeal. Their health is depen-



Concept art for the priest character in front of a church

dent on three factors: Food, pure water and combat, influenced by the abilities of respectively Famine, Pestilence and War. A peasant with low health moves slower and is thus more vulnerable to the scythe of the relatively slow moving Death.

The zeal of the peasants is dependent on how often the peasant has attended mass. When zeal is low the peasant is more receptive to moral corruption such as bribes and mindless violence against their fellow peasants.

Priests have one major stat, fanatism, which increases as peasants in their jurisdiction start dying and stop attending mass A priest with high fanatism runs faster and performs prayers and blessing faster.

Appendix B

Correspondence With Game AI Developers

This appendix contain the correspondence the author had with different AI developers.

B.1 The questionaire

This section presents the standard e-mail that was sent to the different game developers known for their use of game AI:

```
Dear [SIR/MADAM],
I am a computer science student at the Norwegian University of Science and
    Technology( http://www.ntnu.no/english ). I am currently conducting
    research for a Master's thesis about how game AI is implemented in modern
    games, with emphasis on how a believable and challenging AI is made and
    which trade-offs that has to be made concerning the game's computational
    performance. I hope that this research could be of help to starting
    developers, or smaller game studios.

I have compiled a list of high-profile games I would like to investigate, and
    [COMPANY]'s [GAME(S)/SERIES] is one of these.

My research questions are as follows (in regards to game AI):
- What methods and algorithms, not typically considered part of traditional AI
    , are important for a good game AI?
- What traditional AI methods and solutions are used in game AI?
- When and where are traditional AI methods used in game AI?
- How important are traditional AI methods to game AI?
- How much leeway is there for implementation of AI methods that requires a
    lot of computational resources?

I understand that some things have to stay a company secret, but I would be
    extremely thankful if you have any material you could share that would
    shed some light on these questions.

Sincerely,
Remy Jensen
```

Listing B.1: Our questionaire e-mail

B.2 Chris Jurney

This section reproduces our correspondence with Chris Jurney, AI programmer of Brütal Legend and former AI programmer in Relic Entertainment.

```
Hi, Remy,

I'll try to answer your questions, but they're pretty broad, so I'm not sure I
    'll be able to give a full answer. You should take a look at http://
    aigamedev.com/. It's run by industry AI programmers, and they look at the
     crossover between practical game AI and academic AI on occasion.

* What methods and algorithms, not typically considered part of
traditional AI, are important for a good game AI?

The main tools in the game AI toolbox are decision trees, hierarchal finite
    state machines, and A* search. AI in games is a lot closer to an expert
    system custom made for the specific game's problems than any kind of
    generic academic AI style system.

* What traditional AI methods and solutions are used in game AI?
* When and where are traditional AI methods used in game AI?
* How important are traditional AI methods to game AI?

Very few traditional AI methods are used. Some games have tried using very
    basic planning for unit control (F.E.A.R., an FPS did this for its units),
     but it's generally considered not worth the effort. With the advent of
    camera gaming, there's a lot more overlap from the field of computer
    vision, since that research is generally directly applicable.

Generally, traditional AI is avoided because games need reliable and
    reproducible results to craft a quality gaming experience. Traditional AI
     is also usually prohibitively expensive in terms of CPU cycles and memory
    .

* How much leeway is there for implementation of AI methods that
requires a lot of computational resources?

On a modern console, you have access to a number of cores (6-8), so there is
    more opportunity for using computational resources, but it is still
    limited. Anything that consumes more than a few milliseconds in any one
    1/30th of a second frame is probably not useful. Being able to spread the
     work across cores, or across frames helps. AI is pretty core to many
    modern games, so I'd say it's very common for the AI system to receive
    anywhere from 10 to 33% of the overall available CPU. Memory is very
    constrained on consoles, with a total of 512MB on both PS3 and 360, so AI
    is usually restricted to use 10-50MB of that total, depending on the title
    .

I hope that's helpful. If you have any more specific questions, let me know.

Cheers,
Chris
```

Listing B.2: The reply from Chris Jurney

B.3 Crytek

This section reproduces our correspondence with Crytek.

Hello Remy,

Below are some answers to your questions. I am also cc'ing a few other people who may want to add additional comments.

– How much leeway is there for implementation of AI methods that require a lot of computational resources?

A: Unfortunately not nearly enough as would we would like. Games, especially cutting edge games, really push the hardware on all fronts. AI has to share limited hardware resources with high end graphics, advanced physics simulations, networking, audio, etc. all while maintaining real time frame rates of 30 to 60 times per second. The good news is that with GPU's doing more and more, and multicore processors becoming more common, AI is now, and will be in the future, receiving significantly more processing power than ever before. But even still many 'basic' services such as path finding, visibility checking, etc. often still have to be time sliced or multithreaded over many frames so as not to interrupt smooth game play frame rates.

– What traditional AI methods and solutions are used in game AI?

A: Most games include basic path finding with A*. Many games also make heavy use of in game scripted events. Some games do some basic pattern recognition, especially fighting games which tend to try to learn a player's favorite combo of moves. State machines, and priority based behavior trees are used heavily. Influence maps can also be used in many games, as can hierarchical planning.

– What methods and algorithms, not typically considered part of traditional AI, are important for a good game AI?

A: Anything that: 1) fits into strict computational/memory budgets, 2) provides an enjoyable experience for the player. It is easy to make an AI that can beat the player. It is hard to make an AI that provides just the right level of difficulty without appearing to the player to be either god like in power, cheating, or stupid. Being smart is not the goal, appearing smart to the player in ways he/she can understand and compete against is the goal.

We often put a lot of knowledge in the world instead of AI units. For instance, if we have a fence that can be jumped over, instead of programming each AI to jump over fences, we have the fence tell AI's how to use the fence. We could also, for example, have an alarm system button in the world that takes control of the AI under certain conditions and tells the unit how to use the alarm button to alert other bots of the player's existence. We embed other information into the world such as hide spots, or other hand placed markers, special navigation information, etc. The fact that we have complete control over the virtual world means we can do many things that someone building the mars rover cannot. We tailor our worlds around the AI as much as possible instead of the other way around.

AI units can be made to seem smart even when they are doing nothing

```
because they play animations and sounds that give a sense of character.
AI's seem intelligent because they play an animation of smoking a
cigarette or something when actually doing nothing, or because they play
animations and prerecorded conversations indicating that the unit is
scared, suspicious of a given area, or even just talking ambiently about
something in the game world.  Perception of intelligence is more
important that actual intelligence in a game.


- When and where are traditional AI methods used in game AI?

When they: 1) fit into strict computational/memory budgets, 2) provides
an enjoyable experience for the player

More generally, in a single player game we use less fancy traditional AI
systems and often just script bots to do specific things in the
situation that will be presented to the player.  In multiplayer type
games with bots we are more likely to have a wider array of traditional
AI functionality including but not limited to machine learning,
knowledge based systems, etc.  This is because the bots must be more
flexible in this environment, where as in single player they often need
to simply do the things pre planned for the encounter as designed by the
game designer.


- How important are traditional AI methods to game AI?

Game AI programmers try to learn from traditional academic AI whenever
possible.  But because of limited resources and that our end goal is
entertainment, not all traditional academic methods are currently
feasible or useful at this time.



I hope that this answers your questions sufficiently.  If you have more
questions please feel free to email me again directly.  I also recommend
some game AI reference materials: The "AI Game Programming Wisdom" text
book series, as well as "Game Programming Gems" text book series.  These
sources give very good overviews of Game AI programming techniques in
use by many companies today.


-Jeremy Gross
Crytek Sr. AI RnD Programmer
```

Listing B.3: The reply from Crytek

B.4 Epic Games

This section reproduces our correspondence with Epic Games.

```
Hi,

Here are some quick answers.  Also feel free to check out the Unreal
    Development Kit (www.udk.com) to see our AI code firsthand in a game
    scenario.

> What methods and algorithms, not typically considered part of
> traditional AI, are important for a good game AI?

In a game, AI comprises systems for movement (short-term movement of
    characters through an environment), path-finding / navigation (long-term
    determination of movement plans), tactical AI (decisions about which
    weapons to use and which opponents to fight or flee from), and in some
    games, strategic AI (for example, an army's overall strategy in a real-
    time strategy game).

> - What traditional AI methods and solutions are used in game AI?
> - When and where are traditional AI methods used in game AI?
> - How important are traditional AI methods to game AI?

The field of game AI isn't related to the field of computer science AI as far
    as I'm aware.

> - How much leeway is there for implementation of AI methods that
> requires a lot of computational resources?

AI is generally a significant fraction of our overall computational budget --
    say, 10% to 20% of overall CPU time.  While graphics and physics are vital
     to a game's appearance, the overall experience tends to feel hollow
    without good AI in place.

-Tim
```

Listing B.4: The reply from Epic Games

B.5 Jeff Orkin

This section reproduces our correspondence with Jeff Orkin, former AI programmer of Monolith Productions.

```
On Wed, Feb 3, 2010 at 11:15 AM, Remy Jensen <remyjensen@gmail.com> wrote:

    Dear Sir,
    I am a computer science student at the Norwegian University of Science
    and Technology( http://www.ntnu.no/english ). I am currently
    conducting research for a Master's thesis about how game AI is
    implemented in modern games, with emphasis on how a believable and
    challenging AI is made and which trade-offs that has to be made
    concerning the game's computational performance. I hope that this
    research could be of help to starting developers, or smaller game
    studios.

    I have compiled a list of high-profile games I would like to
    investigate, and the F.E.A.R. and Condemned series you worked on while
    still with Monolith Productions are among these.

    My research questions are as follows (in regards to game AI):
    - What methods and algorithms, not typically considered part of
    traditional AI, are important for a good game AI?


The way you've phrased this question assumes game AI is all about algorithms,
    which is an incorrect assumption. Game AI is a combination of decision
    making (algorithms), presentation (animation), and authoring (knowledge).
     Characters use knowledge to make decisions, which they express through
    animation. Traditional AI works well for decision making algorithms, but
    does not give enough insight into how best to select and blend animation,
    and get knowledge into the system (through authoring tools).



    - What traditional AI methods and solutions are used in game AI?


Most games use Finite State Machines. Some use Hierarchical Finite State
    Machines, sometimes referred to as Behavior Trees. FEAR and Condemned use
     a planning system based on STRIPS. We're beginning to see some games use
     hierarchical planning systems, such as Hierarchical Task Networks. Also,
     most games use the A* algorithm for path planning. There has been lots
    of work on optimized variants of A*.


    - When and where are traditional AI methods used in game AI?


A* for path planning. The others mentioned above for decision making,
    sometimes referred to as action selection.


    - How important are traditional AI methods to game AI?


Well, you need characters to make decisions somehow, and you are probably
    better off using a well thought out formalism than something ad hoc. The
    beauty of using planning algorithms is that characters can start to reason
     autonomously, rather than relying entirely on hard-coded behaviors. They
```

```
can search for sequences of actions to satisfy their goals, and adapt to
whatever they encounter in the game. But, there is still a problem of how
to author enough actions and goals for the character to do everything
designers want them to do. Today, this is largely up to the engineers,
but hopefully in the future authoring behaviors will be more accessible to
designers.


- How much leeway is there for implementation of AI methods that
requires a lot of computational resources?


Graphics and physics will always take the bulk of the processing resources.
In general, AI gets maybe 10% (which is more than used to be available).
But game developers can find ways to do more with less by distributing the
processing load, to process in small increments and cache results for
algorithms to exploit. An exciting prospect for the future is AI that
runs remotely "on the cloud", which would free us to spend much greater
processing resources, because there would be no interference with
rendering on the client.



I understand that some things have to stay a company secret, but I
would be extremely thankful if you have any material you could share
that would shed some light on these questions.

Sincerely,
Remy Jensen
```

Listing B.5: The reply from Jeff Orkin

Appendix C

Apocalypse Code

This appendix contain the excerpts of the source code of the classes discussed in the report.

C.1 ApocEnemyController

This section contain excerpts of the source code in ApocEnemyController class.

```
event vector GeneratePathToActor( ApocNavPointInterface Goal, optional float
    WithinDistance, optional bool bAllowPartialPath )
{
 local vector NextDest;
 local Vector eyePos;
 local bool samePos;
 local float deltaDist;

 FlushPersistentDebugLines();

 deltaDist = VSize(Pawn.Location - LastPawnPos);
 samePos = ( deltaDist < 0.0001f);

 if(samePos)
 {
  TardCounter++;
  `log(name @ Pawn.Name @ "increased tardness to" @ TardCounter);
 }
 else
 {
  TardCounter = 0;
 }

 NextDest = Goal.GetNavPoint();

 if(Enemy == none && self.IsWithinLineOfSight(Goal.GetNavPoint()) && (
     TardCounter < 2))
 {
  //`log(name @ Pawn.Name @ "is in a straight line of it's goal and deltaDist
      is:" @ deltaDist);
  LastPawnPos = Pawn.Location;
  return NextDest;
 }
 //else
```

```
 //{
 // `log(name @ Pawn.Name @ "is currently retarded");
 //}

 if ( NavigationHandle == None )
 {
  InitNavigationHandle();
 }
 // Clear cache and constraints (ignore recycling for the moment)
 NavigationHandle.PathConstraintList = none;
 NavigationHandle.PathGoalList = none;




 if (Enemy != none)
 {
  //class'NavMeshPath_BiasAgainstPolysWithinDistanceOfLocations'.static.
      BiasAgainstPolysWithinDistanceOfLocations(NavigationHandle, Enemy.
      Location, Enemy.Rotation, ApocPawn(Pawn).AVOID_DISTANCE, VectorList);
  //class'NavMeshPath_MinDistBetweenSpecsOfType'.static.EnforceMinDist(
      NavigationHandle, ApocPawn(Pawn).AVOID_DISTANCE, NAVEDGE_Normal, Enemy.
      Location);
  class'NavMeshPath_WithinDistanceEnvelope'.static.StayWithinEnvelopeToLoc(
      NavigationHandle, Enemy.Location, 1000000000.0f, ApocPawn(Pawn).
      AVOID_DISTANCE, true, 3000.0f, true);
 }
 else
 {
  class'NavMeshPath_Toward'.static.TowardPoint( NavigationHandle, Goal.
      GetNavPoint());
 }

 class'NavMeshGoal_At'.static.AtLocation( NavigationHandle, Goal.GetNavPoint()
     , WithinDistance, true);

 if ( NavigationHandle.FindPath() )
 {
  if (!NavigationHandle.GetNextMoveLocation(NextDest, 120))
  {
   `log(name @ Pawn.Name @ "Found path but GetNextMoveLocation failed hard. He
       so want to run to (0, 0, 0) but we won't let'im");
   NextDest = Pawn.Location;
  }
 }
 else
 {
  `log(name @ Pawn.Name @ "found no path to" @ Goal.Name);
 }

 // TODO fjern debug lines
 eyePos = Pawn.Location;
 eyePos.Z += Pawn.EyeHeight;
 //DrawDebugLine(eyePos, NextDest, 255, 0, 0, true);
 //NavigationHandle.DrawPathCache(vect(0, 0, 0), true, MakeColor(0, 0, 255,
     255));

 NavigationHandle.ClearConstraints();
 LastPawnPos = Pawn.Location;
 return NextDest;
}
```

Listing C.1: The the *GeneratePathToActor()* event in the *ApocEnemyController* class

Bibliography

[1] Nate Anderson. Video gaming to be twice as big as music by 2011 [online]. Available from: http://arstechnica.com/gaming/news/2007/08/gaming-to-surge-50-percent-in-four-years-possibly.ars.

[2] Eric J. Braude. *Software Engineering: An Object-Oriented Perspective*. Wiley, 1^{st} edition, 2000.

[3] Alex J. Champandard. Choosing a hierarchical fsm or a hierarchy of nested fsms? [online]. Available from: http://aigamedev.com/open/articles/hierarchical-or-nested-fsm/.

[4] Alex J. Champandard. Common ways to implement finite state machines in games [online]. Available from: http://aigamedev.com/open/articles/fsm-implementation/.

[5] Trendy Entertainment. Dungeon defense [online]. Available from: http://www.udk.com/showcase-dungeon-defense.

[6] DFC Intelligence. Dfc intelligence forecasts video game market to reach $57 billion in 2009 [online]. Available from: http://www.dfcint.com/wp/?p=222.

[7] Ian Millington. *Artificial Inteligence For Games*. Morgan Kaufmann Publishers, 1^{st} edition, 2006.

[8] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Companies, Inc, international edition edition, 1997.

[9] Norwegian Ministry of Culture and Church Affairs. Video games. *Reports to the Storting (2007-2008)*, 14:29–30, March 2008.

[10] Amit Patel. Heuristics [online]. Available from: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html.

[11] Steve Polge and Matt Tonks. Ai system overview [online]. Available from: http://udn.epicgames.com/Three/AIOverview.html.

[12] Charles Rich. Basic game ai [online]. Available from: http://web.cs.wpi.edu/~rich/courses/imgd4000/lectures/B-AI.pdf.

[13] Tim Sweeney. Unrealscript language reference [online]. Available from: http://udn.epicgames.com/Three/UnrealScriptReference.html#States.

[14] Matt Tonks. Navigation mesh path constraints and goal evaluators [online]. Available from: `http://udn.epicgames.com/Three/NavMeshConstraintsAndGoalEvaluators.html`.

[15] Matt Tonks. Navigation mesh reference [online]. Available from: `http://udn.epicgames.com/Three/NavigationMeshReference.html`.

[16] Matt Tonks. Navigation mesh technical guide [online]. Available from: `http://udn.epicgames.com/Three/NavigationMeshTechnicalGuide.html`.

[17] Matt Tonks. Using navigation meshes [online]. Available from: `http://udn.epicgames.com/Three/UsingNavigationMeshes.html`.

[18] Paul Tozour. *AI Game Programming Wisdom*. Charles River Media, 1^{st} edition, 2002.

[19] Wikipedia. Game artificial intelligence [online]. Available from: `http://en.wikipedia.org/wiki/Game_artificial_intelligence`.