CHALMERS
UNIVERSITY OF TECHNOLOGY

Realistic NPCs in Video Games Using Different AI Approaches

Bachelor of Science Thesis in Computer Science and Engineering

Gustav Grund Pihlgren
Martin Nilsson
Mikael Larsson
Oskar Olsson
Tobias Foughman
Victor Gustafsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, June 2016

Realistic NPCs in Video Games Using Different AI Approaches

Gustav Grund Pihlgren
Martin Nilsson
Mikael Larsson
Oskar Olsson
Tobias Foughman
Victor Gustafsson

Examiner: Niklas Broberg

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover: An image from the video game created during this project that has been transmuted using Deep Dream Generator [1], a website that uses neural networks to transmute images. This image along all other images from the game is made from several other images. Some of these images are made by the project members and the others can be found correctly accredited in Appendix A]

Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Abstract

This bachelor thesis describes the process of creating an AI for realistic NPCs. It compares different approaches used to implement AI in order to try to convey their advantages and disadvantages to contribute towards the creation of more immersive NPCs in video games.

The work was divided into two main categories: The first to create a video game in which to implement NPCs and the other to create an AI to control these NPCs. A variety of different implementations were investigated during the course of this project and are discussed in this report, and a game was created to test one of them.

Sammandrag

Denna kandidat rapport beskriver processen av att skapa en AI för realistiska NPCer. Den jämför olika tekniker som används för att implementera AI för att försöka förmedla deras fördelar och nackdelar för att kontribuera till skapandet av mer inlevelsefulla NPCer i datorspel.

Arbetet var indelat i två huvudkategorier: Den första var att utveckla ett datorspel för att utveckla NPCer i, och den andra var att utveckla en AI för att kontrollera dessa NPCer. En mängd olika implementationer undersöktes under projektet och diskuteras i denna rapport, och ett spel utvecklades för att testa en av dessa.

Keywords: AI, programming, NPC, videogame, simulation, realistic.

Dictionary

Character - An object without intelligence that may be controlled by an intelligent source (i.e. an AI or a player).
Artificial Intelligence (AI) - Intelligent control unit for the characters.
Non-player character (NPC) - A character which is controlled by the AI.
World - The complete collection of items, characters, structures and environment that makes up a video game.
Model-view-controller (MVC) - A software design pattern used to separate underlying data and the visual presentation of that data from each other.
Tileset - A collection of images and textures in fixed sizes that are used to model the game world, merged into the same image, side by side in a grid.
Java - An object-oriented, multi-platform programming language developed by Sun Microsystems, specifically designed to let a program run unaltered on many different devices and operating systems.
Slick2D - Slick2D is a set of tools and utilities for handling 2D graphics, as well as other things such as music and particles.
Sprite - An image or animation in a video game.

Preface

This report is a bachelor thesis at the Department of Computer Science and Engineering, Chalmers University of Technology, performed in the spring of 2016. It describes and discusses the planning and implementation of a realistic AI to be used in a resource based simulation game.

We want to thank: the institution for technical language, the Chalmers library and Daniel Sjölie, our supervisor.

Contents

1 Introduction

Video games have gone from beginning to gain popularity in the mid-1970s to being a part of modern culture as well as an economic powerhouse where the global game market is estimated to reach a total of 100 billion USD in revenues in 2016 [2]. Even though many high quality games of today have enormous 3D-worlds, with graphics near photorealism to show for it, immersion is still lacking when it comes to the behaviour of the NPCs inhabiting these worlds. “The believability of NPC behavior is crucial for immersion in games and enables seamless interaction between players and between players and NPCs”[3]. In terms of game immersion, storytelling and AI research (and other research areas as well)[4], a more intelligent behavior of video game agents opens up a vast array of unseized opportunities.

1.1 Background

The term Artificial Intelligence was first coined by the American computer- and cognitive scientist John McCarthy, at the second Dartmouth Conference in 1956 [5]. According to McCarthy "every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" [6]. Still, the idea of artificial intelligence can be traced back to philosophy, fiction and imagination as early as in the classical period of ancient Greece [7].

Video games didn’t reach mainstream popularity until the 1970s when arcade games and gaming consoles were first introduced to the general public. However, research on game AI was already being conducted as early as 1948 on a computerized game called “Nim” [8]. Just a few years later, in 1951, Christopher Strachey wrote an AI program for playing checkers and Dietrich Prinz wrote one for chess using the Ferranti Mark 1 machine at the University of Manchester. Further advancements in gaming AI led to an AI (IBM’s Deep Blue) beating the ruling world champion of Chess, in 1997 [9]. More recently the world champion of the board game Go was beaten by AlphaGo, an AI developed by researchers at Google [10]. Go was previously the last non-chance combinatorial game with perfect information in which the best human players were considered better than their AI counterparts [11].

It is worth mentioning that there are some discussions involving whether or not the term “game AI” exaggerates the valuation of its notion. AI in video games fill a different role from AI in the academic field [12]. In academics there are several definitions of what an AI is, but most agree that an AI automates intelligent behaviour or reasoning [13].

Despite these discussions, there are others who think video games and AI research go hand in hand. Looking from an entertainment point of view, advancements in AI research can help to increase the demands to add realistic and intelligent behaviour to characters in video games. The other way around, as video game environments become more complex and realistic, they also open up possibilities for acting as a testing ground for AI research in many different fields [14][4]. Furthermore Demis Hassabis (CEO of Google DeepMind) stated in his presentation, The Theory of Everything, that “Games are the perfect platform for developing and testing AI algorithms” [15]. An example of this is the use of behaviour trees in AI which originates from its use in video games [16].

There are games out there which have succeeded and have been awarded for their contribution of influential AI-games. Black & White, currently ranked number one on the list of the most influential AI games, is a video game developed by Lionhead Studios and was created in 2001; where Demis Hassabis was one of the lead programmers. The game incorporates artificial life simulation, in the form of giant animal-like creatures, combined with strategy. An AI architecture known as belief-desire-intention (BDI) was used in the game engine and the game also had great success with machine learning, such as neural networks and decision trees [17].

Despite the progress made in AI research over the last few years, the potential of AI in video games remains unfulfilled. Even in very recent, “triple A” titles, such as Grand Theft Auto or Assassin’s Creed, a player will still encounter numerous situations where the AI behaves so unrealistically that it will completely break immersion. NPCs in these games representing humans lack some of our most fundamental cognitive skills, such as memory or the ability to communicate information to one another. A common trait of the titles mentioned is that a player can commit a crime in front of the guards, flee and then return to the very same guard minutes (or even seconds) later. The guard will have no memory of the recent event, nor will he remember the player. [3].

Even games that use AI as a selling point, or have been praised for their AI, often will not provide what one could expect these days in terms of game immersion. In 2012, the game Skyrim [18] won the Academy of Interactive Arts & Sciences “Game of the Year” award [19], as well as numerous other prizes and is considered one of the best games of all time[20]. Still, the AI of the game is prone to break immersion time and time again through unrealistic or irrational behavior. A typical example from Skyrim would be that an NPC could ask you to perform some mundane task that he or she is not capable of doing themselves, like clearing their house of mudcrabs. However, when a dragon suddenly appears, the NPC will forget that he or she is merely a weak citizen and will attempt to slay the dragon with their bare hands, showing no signs of fear [3].

1.2 Purpose

The purpose of this bachelor thesis is to describe the process of developing a video game and the AI that controls the NPCs in this video game. This process also serves as basis for investigating what it takes for an NPC to be realistic and how different AI approaches can be used to implement realistic NPCs in video games.

1.3 Scope

Considering that the focus of this project lies on creating artificial intelligence, the foundation of the game (the game without NPC) will be simple in both its visual aspect and its gameplay aspects. The foundation of the game will be designed with focus on making a world that NPCs can interact with, change, and be challenged by. The game will thus not necessarily be challenging (or fun) for a human player, but rather be an open world to test and challenge the AI.

The game to be created will be a survival game extended with elements from the strategy genre. The elements from strategy are there to challenge the AI. It forces the AI to take different factors into account, such as resource management, in order for it to survive. The initial intention was to implement multiple different AIs using various approaches in order to compare them to each other.

2

Technical Background

These first subsections will briefly describe the tools and techniques used in order to create the game; since a lot of work is required to get a game running. The last subsection of this chapter focuses on giving the reader a brief introduction to well known AI approaches.

2.1 Slick2D

Slick2D is an open source Java library, wrapped around LWJGL (Lightweight Java Graphics Library) [21]. It contains tools to facilitate the use of images, animations and other graphic components, as well as tools for things like sound and particles. These tools exists to make Java game development easier, as it relieves a lot of the workload for programmers whose main focus does not comprise of graphics handling.

2.2 Model-View-Controller

Model-View-Controller, MVC for short, is a design pattern for software applications [22][23]. The design is popular as it separates pure logic and the user interface from one another into three interconnected parts: model, view and controller. The most prominent benefit of the design is more organized and manageable code, which is an important feature of larger applications. Other benefits of dividing the code in this way are:

- Reusable code
- More extendable code
- Allows for concurrent work between developers, responsible for different fields, without stepping on each others toes.

Some possible drawbacks to Model View Controller:

- Increasing complexity as applications sometimes cover other patterns, in ad-

dition to the MVC.
- If the model is actively changing it can mean excessive update of the corresponding view.
- Excessive information passing between the three interconnected parts, if poorly constructed for the specific purpose.
- The three parts have to be synchronized in order to display correct information in real-time.

How these three parts are interconnected vary significantly between applications, depending on what is most preferable for that specific application. It also comes down to different opinions and interpretations on what is the overall best way to implement the design pattern.

2.3 Tiled Map Editor

Tiled is a free map editor program that can be used to create 2D-maps for games [24]. It's compatible with the Slick2D library and the two are easy to use together. Tiled makes world creation easier as it removes the coding that has to be done in order to create a world in run time. Instead, one basically paints the world with a tile set of one's own choosing.

2.4 UML Diagrams

UML stands for Unified Model Language and is a general-purpose, modelling language in the field of software engineering intended to provide a standard way of visualizing the design of a system. The notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation to be used for object-oriented design, but has since been extended to cover a wider variety of software engineering projects[25][26].

2.5 Pathfinding

In games where NPCs move dynamically around the terrain, as opposed to predetermined paths, there needs to be some way of moving about the world with regard to solid objects and non-traversable terrain. This way of finding a viable path is called 'pathfinding' and utilizes some kind algorithm in order to to so. Some instances may require the absolute shortest path from point A to point B, while others are content with just finding a path. Many different versions of this kind of algorithm have been used throughout the industry [27].

A pathfinding algorithm will, given a graph, find a path from one point to another. More advanced algorithms also guarantee that the found path is the shortest path between the two points, at a small reduction in efficiency. There are several way to define the ‘shortest path’; it can mean simply the least amount of transitions in the path or the lowest cost of all the transitions in the path.

Initially, when searching a graph, you add the starting node to a list of discovered nodes, called the ‘frontier’. Then the algorithm will check each discovered node and if that node is the final node, the path has been found. If the node was not the final one then the that node will be added to a list of considered nodes, to keep the algorithm from checking a node several times, then adds all connected nodes to the frontier and selects a new node to search from. If there are no discovered nodes left to search from then there is no path to be found. In what order the nodes are searched depends on what pathfinding algorithm is used.

2.5.1 The A* Search Algorithm

One of the most popular pathfinding algorithms is A* (sometimes A-star). A* searches the graph by selecting what node to search from next using a programmer-defined heuristics function. This function is based on what knowledge the programmer has of the graph and determines what node is most likely to be part of the shortest path to the end node. For the algorithm to be guaranteed to find the shortest path, the heuristic needs to be admissible. This means that, in all cases, the value of the heuristic is always equal or better than the real value of that path. Because of this the performance of A* depends heavily on how good the heuristics function is at guessing the shortest path. Guessing a bad value will force the algorithm to explore more nodes than it would need to, resulting in worse performance.

2.5.2 A* Algorithm On Two-Dimensional Grids

In 2D-video games, one particular problem is having NPCs find their way around the world without walking into solid objects. A solution to this problem is converting the map into a 2D grid. A cell in the grid can either be empty or occupied. Empty means that the character can walk to or through that cell whereas occupied means that the place is unreachable. An example can be found in figure 2.1. To be able to pathfind through this grid it is modelled as a graph. Each empty cell becomes a node with transitions to all empty adjacent cells. Always selecting the node with shortest straight line to the end is an efficient heuristic when searching for the shortest path through a 2D grid.

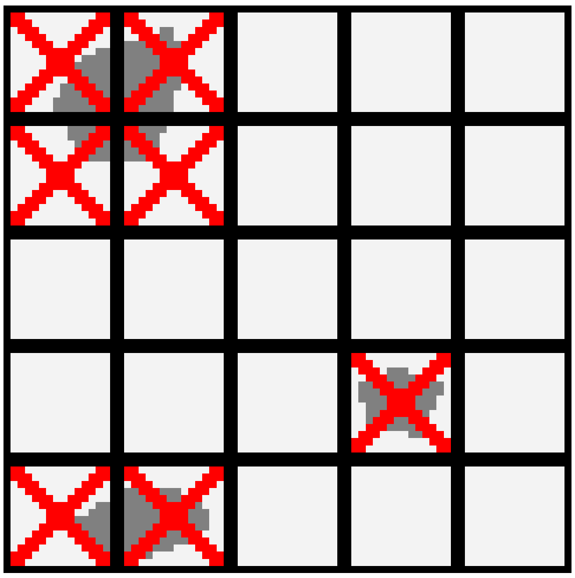


Figure 2.1: A pathfinding grid on a world of rocks. Crossed out cells are occupied

For searching this grid by using the A* algorithm, an efficient heuristic is to always select the node with the shortest possible path to the end. This makes A* on 2D grids excellent for finding paths around solid objects [28].

2.6 Defining Believable and Realistic NPCs

This section and most of the report will concern NPCs that represent humans or human-like characters. For the purpose of this thesis it is important to make a difference between the words believable and realistic. Note that these definitions are neither standardized in the academic field of AI nor in the video game industry; they are here simply to make the report easier to understand.

According to the book “Artificial Intelligence: A Modern Approach”, the field of AI can be divided into four main categories [13]. These can be seen as different approaches to intelligence. First, the field can be divided into rational and human AIs. Rational AIs are made to reach a goal or find a solution. This could lead to a rational AI taking an unintuitive decision because it calculated that would be the best decision at the moment. Human AIs are made to make the decision a human would. Further these two groups can each be divided into AIs that act and AIs that think. When designing an AI that acts one does not care for how that AI works as long as it acts intelligently. Conversely when designing an AI that thinks the underlying decision process is important. This gives us the four groups: Thinking rationally, acting rationally, thinking humanely and acting humanely [13].

In our definition, an NPC is believable when it appears to be human. There are no requirements on the underlying mechanics of the NPC as long as the NPCs’ behaviour is apparently human. An example of this could be that an NPC has no need for drinking, as it cannot become thirsty, but it will purchase a drink anyway to keep up the illusion that it is human. The character of a believable NPC could be viewed as a puppet and the AI is tasked with making this puppet seem human. Referring to the categories from the previous paragraph, the AI of a believable NPC would therefore be of the type that “acts humanely”.

In our definition, an NPC is realistic when it simulates a human. The character can be seen as a body and the AI as the brain. This type of AI does not simply perform typical human actions for the sake of illusion, but has an actual need to do so or else it will perish. A good example is The Sims; in The Sims, a character does not drink because “that is what a human would probably do”, but because it has an underlying need to do so. The AI in this case would either fall in the category of thinking rationally or humanely, depending on the design of the AI.

2.7 Artificial Intelligence approaches

There is a multitude of designs available to the programmer when implementing artificial intelligence for video game NPCs. What design to choose should be determined by the desired behaviour of the NPCs in question. The following paragraphs will describe how some of the most popular designs work, as well as what their advantages and disadvantages are.

2.7.1 Finite State Machines as Artificial Intelligence

A Finite State Machine, FSM for short, is a system that has a finite number of states that an AI can enter. Each state is unique and determines the behaviour of the AI. What state the AI will transition into next is decided by predetermined conditions. Let us take an example of a basic NPC in the form of a city guard. For simplicity, the following states the AI can enter and transition between will be patrol, suspicious and attack. If the guard has not spotted the player, it will patrol the area. Granted that the guard is in patrol state and the player is suddenly spotted, the guard will transition into the suspicious state and actively search for the player. When in suspicious state, the guard has 15 seconds to spot the player again in order to transition into attack state, chasing and attacking the player. If the condition of spotting the player in the following 15 seconds is not met, the guard will transition back to patrol state. This FSM is visualized in figure 2.2.

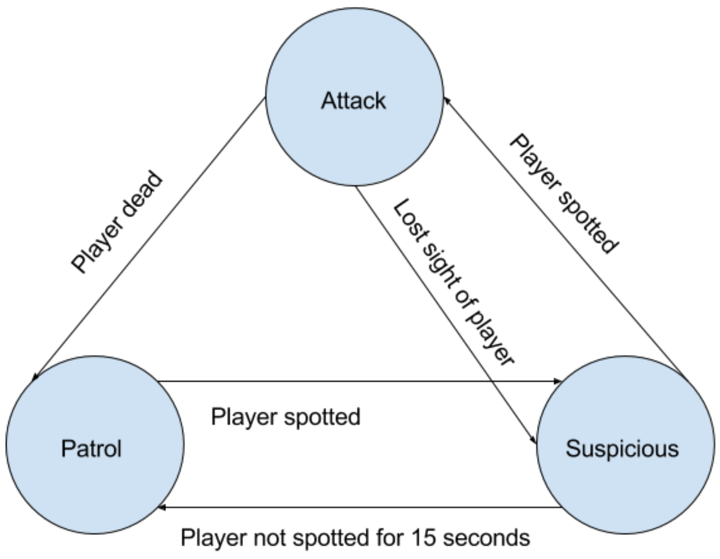


Figure 2.2: Visualization of a simple FSM, using three states.

The advantage of an FSM is that it is easy to implement and understand. They are easy to expand, as long as the number of states is relatively small. Debugging also becomes relatively simple, since the actions performed can often be traced back to a specific state [29].

One of the disadvantages of an FSM is that it become increasingly difficult to manage, maintain and expand as the number of states grow larger. For every state already defined in the state machine that should be able to transition to the newly created state, new transitions and conditions for them have to be defined. FSMs can be predictable if their states and transitions are defined by overly simplistic rules. They can also get stuck in unrealistic behaviour if the conditions for transitioning are too many, too strict or poorly defined [29].

2.7.2 Structuring FSMs Using Hierarchy

A common problem with FSMs is that often the logic of a state cannot be reused in a different context. For example, you might want to reuse the attack state of an NPC on several occasions, but the transition conditions may not allow it. States are often designed around specific logic that has to be provided in order for the state to work as it should. If that logic is not provided, which is most likely the case if the state is used in a context it is not designed for, the FSM will fail. As a developer you are faced with two options. Either states have to be complicated and account for all contexts the state might be used for, or you have a lot of redundancy and have to make numerous specific transition conditions for a lot of states [30].

Hierarchical FSMs allow you to keep your states fairly simple while avoiding a lot of the redundancy [31]. Hierarchical FSMs use something called “super states”. For example there is a game with five different weapons, all of which change the way an NPC attacks and requires its own state. Instead of having five different attack states that have to be considered in every state that can transition into an attack state, there can be one attack super state.

The super state contains the logic required to determine which of the five attack states that should be used, depending on what weapon the NPC has. Now every time the NPC wants to attack it will first go to the attack super state and then to the correct sub-state corresponding to the weapon used.

2.7.3 Problem Solving as AI Technique

In the field of AI, problem solving is the study of finding a correct sequence of actions needed to solve a given problem; the solution is often called the goal of that problem. A problem can be modelled as a directed graph where each node represents a possible state of the problem and each transition is an action that changes the state of the problem. For a problem to be solvable there needs to be at least one solution (goal state) that is reachable from the starting state [13].

A problem solving AI is a design that, given a certain problem, uses this modelling technique to find a solution. To get a sequence following some specific constraints, for example the sequence with the least number of actions, one could use a more advanced problem solver.

Commonly a problem solver will not have a complete graph to work with but rather have a starting state, the possible actions for that state, conditions for what is regarded as a goal state and knowledge of how the state will change when performing an action. With this information, the problem solver can generate its own graph by creating the states that each action would transition to; consequently having an AI repeat the process from the possible actions in each new state until a goal state is found.

2.7.4 Machine Learning

Machine learning is a field within computer science that deals with constructing algorithms that look for and recognize patterns in data in order to make predictions based on new data. Implementations of these algorithms are trained on a set of training data, allowing them to improve themselves. The training data is the same kind of data that is to be supplied to the running program, along with the desired result. The program then evaluates its own performance and adapts in order to improve [32].

A subgroup to Machine Learning is the so called Unsupervised Learning. The central concept of Unsupervised Learning is that no desired result is supplied. The algorithm gives a result based on all the data supplied by trying to find patterns in it [33].

Machine learning is not a new concept, but has gained a lot of momentum in the recent years through the expanding collection of data, affordable processing power and inexpensive storage.

3

Development of the Game

This chapter is going to describe the creation of the game. To be able to implement the AI in this project a world with distinct rules for the AI was needed. Creating this world was done in several steps.

The first step was to define the humans in-game, the world, the objects within it and the interactions between these objects. This was done in a software description which consists of two parts, a plain text description of the desired result and then a formal definition using functional requirements. Below is an excerpt of the list of functional requirements:

1. Characters should have needs.
2. The magnitude of the needs should increase over time.
3. Characters should have skills.
4. The skills should increase when used.
5. Characters should have traits.
6. ...

The full software description, including the full reasoning behind these functional requirements, can be found in appendix B.

The second step was to structure the process of implementing the game. Firstly how to handle version control. Secondly decide on the overall software design, which was decided to be a implementation of the design pattern MVC. And thirdly to create a UML-diagram based on the functional requirements.

The third, and last, step was to implement the game. This step was naturally the largest, and was done over several iterations. This allowed for the development of the AI to start as soon as possible, as only a very basic game was required for testing the first versions of the AI.

3.1 Software Design

It was decided that this project should utilize one of several different possible model-view-controller (MVC for short) implementations. In this section, the way that the different parts of MVC were designed to work and interact in this particular project will be explained.

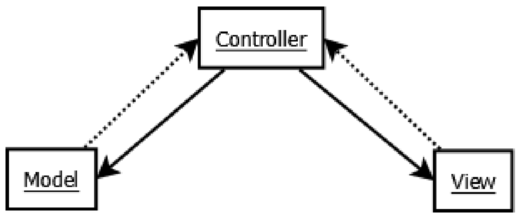


Figure 3.1: Illustration of the implementation of MVC in this project.

The controller is the main thread in the program. It works as a bridge between the model and the view, handling events and passing information between these packages. Its main tasks are to update the game at every cycle, listen to the input from the view and run the AI. The controller also sends a list containing what objects to render and where to render them. The controller is in theory split into two parts; one is the controller that handles the public information and the other is a smaller AI-controller that only has access to the data linked to the characters.

The view in the MVC presents the data stored in the model. It displays everything that exists in the game such as characters, structures and resources. The view is the front-end of the application and works by constantly listening to the input of the player, passing on the received information to the controller.

The model is a logic representation of the game. It acts as the memory of the game, keeping track of what objects are active, their position and their state. The model also contains the functions for modifying the world, used in the update stage of the controller. The model will, when asked by the controller, pass along or update information. The main controller can ask for any public data whilst the AI-part only can ask for data related to the character it controls. A visualization can be seen in figure 3.1.

3.2 Creating the UML-Diagram

In order to outline how all the classes in the model relate to one another as well as achieving a more sensible structure in the code, a UML-diagram was developed. The UML-diagram was created by adding a class for every noun found in the functional requirements. Then, some were combined into one as they shared most properties with each other, and as such could be represented by the same class. Next interfaces and help classes were added where they were considered needed. After a few repetitions of these two steps, the UML-diagram in figure 3.2 was the result. Although some changes were made during the iterative development of the game, the most parts of the UML-diagram remained unchanged.

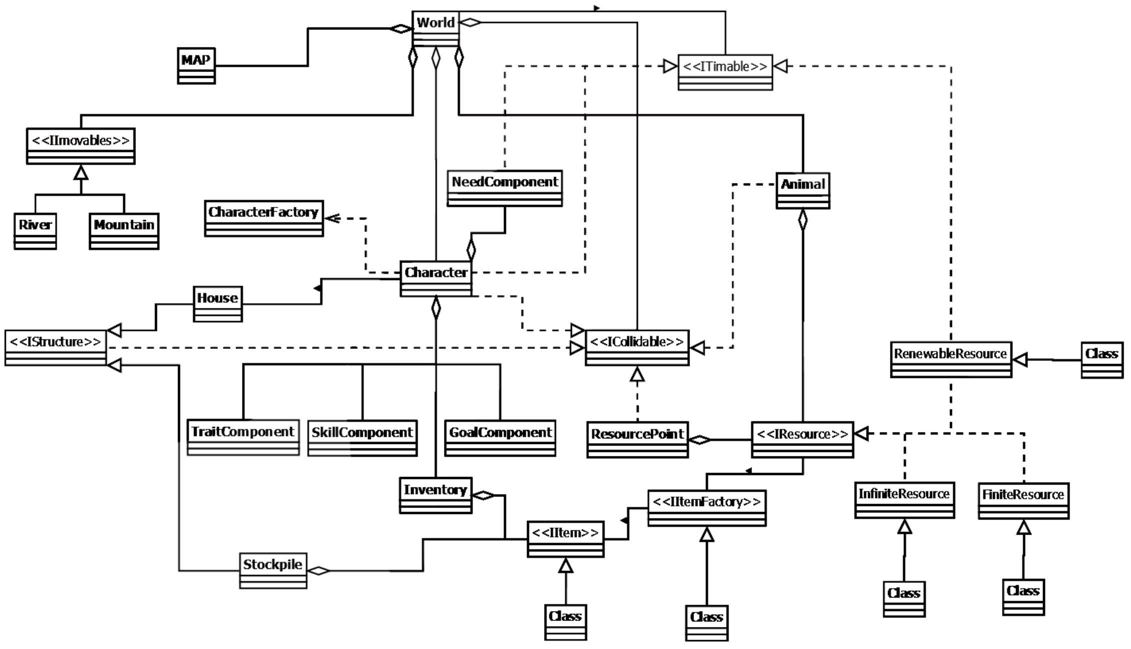


Figure 3.2: The UML-diagram that was used as outline for the development of the model.

3.3 Defining a Human In-Game

As the purpose of this game was to implement a realistic AI, it was important to properly define a human in-game. Several different aspects of the in-game humans were planned for several different reasons.

In order to make the characters realistic, it was decided to give them needs similar to those of a human. For the character to survive, it has to manage these needs. It was also decided that the characters should have traits that determine how they act and how their needs change, forcing them to think differently. These two aspects of the characters where the most central and will therefore be covered more thoroughly in the two following sub chapters.

In order to give the NPCs something to do, other than just satisfying their needs, the concept of goals was introduced. Two different types of goals were envisioned: short term goals as well as a lifelong goal. To further differentiate the NPCs from each other, the concept of skills was discussed. Skills were supposed to decrease the time it takes for the character to gather a corresponding resource as well as increasing the yield from that resource, thus allowing the character to gather it more efficiently.

3.3.1 Needs

A character was decided to have three different basic needs; hunger, thirst and energy. The three needs were based on the the underlying physiological needs of a human being according to Maslow's hierarchy of needs [34]. Hunger represents the need to eat, thirst the need to drink and energy the need to sleep. The higher the value of the need, the better for the character. These needs must be considered by the AI, with a high priority, in order to avoid certain death. The main objective for the AI is to survive as long as possible. Managing resources is therefore important because by eating, drinking and sleeping, the AI can restore its needs and consequently survive.

- Hunger: Managed by eating food such as meat, crops or fish, gathered from places like animals, farms and lakes.
- Thirst: Managed by drinking water, gathered from lakes.
- Energy: Managed by sleeping in houses (or resting outside if not a house owner).

The needs should deplete continuously over time in the game, though the depletion rate can vary depending on the traits of the character. If any basic need is fully depleted the character should die.

There were also plans to implement secondary needs for the characters. Secondary needs are non-fatal and will instead affect the character in a negative way if not managed properly. There were several secondary needs planned for the game, such as socializing, intimacy and attention. These were more centered around interaction compared to the basic needs and were supposed to give the NPCs a more realistic behaviour around each other.

3.3.2 Traits

The purpose of the traits was to vary a character's behaviour, and were inspired by the seven deadly sins. Here follows a description of the way the traits were intended to work. The character is affected by its traits, which are randomized when the character is initiated. These were implemented to give some uniqueness to each character. The character is affected by each trait on a scale from 0 to 100. The effect of each trait is the following:

- Greed : Will be more likely to amass wealth, in the form of gold.
- Gluttony: A character will get hungry faster.
- Sloth: A character's energy will deplete faster.
- Lust: Will be more likely to reproduce.
- Wrath: Decreases the damage taken from attacks.
- Envy: Increases the range of vision for a character.
- Pride: More difficult to form relations as well as less likely to accept unfavor-

able trades

3.4 Defining the Game World

When the in-game humans (or characters) had been defined, the next step was to define the world they were to inhabit. To give the characters a way to manage their needs, it was decided that the world should contain a number of resources. For the AI to be able to improve their situation and allow for more effective management of needs the concept of structures was introduced. The next two sub chapters will in further detail describe the envisioned resources and needs.

3.4.1 Resources

This chapter will describes the way that resources were envisioned. The resources in the game are means for the NPC to survive. A character can gather these resources and use them for various purposes. Resources are either spawned at random locations at initialization of the world or spread from an already existing source. They are divided into three major categories: renewable, finite and infinite resources.

- Renewable resources - these resources reproduce as long as they exist in the world. For example, new trees will grow in proximity to other trees. If all trees are chopped down, no more trees will grow.
- Finite resources - these resources spawn in a fixed amount when the world is initiated. When these resources have been depleted the characters will have to manage without them. Stone will exist in the world as finite resources.
- Infinite resources - a resource that can be gathered infinitely and will never deplete. Water is an example of an infinite resource, as it will remain in the game no matter how much water is gathered.

Animals are a special case in this project, but can be considered a renewable resource. They will wander the world and may reproduce when in close proximity to one another. The characters can hunt these animals for meat, which is why they are mentioned in this section.

3.4.2 Structures

This subsection will describe the way that structures were envisioned. One choice the AI has, in order to replenish needs more conveniently, is to construct different buildings. Structures require resources in order to be built, and these have to be gathered before beginning construction. There are a few buildings that can be constructed for different purposes and they are the following:

- Houses - accommodate characters. Here they should be able to sleep and recover energy. A character should preferably have a house, as sleeping in a house restores more energy than resting out in the open.
- Stockpiles - a place where characters can store their resources. The stockpile inventory should be larger than a single character's inventory.
- Windmill - a building that characters can work at to produce crops they can harvest for food. Any character should be able to work at any windmill and harvest the crops growing there.

3.5 Implementing the Game

The implementation of the game was done by dividing the development process into several steps. At each weekly meeting, decisions were made on further improvements that had to be done as well as what to implement next. The first step in the process consisted of implementing the most essential and basic parts of the model. This step included the coding of necessary interfaces as well as setting up the basic skeleton code to some of the classes in accordance with the UML-diagram.

Next came the development of the view and controller, along with the communication between the three interconnected parts of the MVC. After this step, the only thing remaining before obtaining a simple but working game was to add all the residual necessities regarding the model, which the AI would later need access to. Following this, was an iterative process of refactoring the code to be more efficient and manageable, as well as adding new features.

The following subsections will describe some of the most central parts of the development; pathfinding, update function and displaying the game. These parts were developed in several iterations, just as the rest of the game. However, as they were very central, they did not undergo any major changes in the later iterations.

3.5.1 Pathfinding in the Game

In order for the NPCs to navigate through the world, a pathfinder class was implemented. This class has two main uses; the first is to map a grid over the world containing information of where the character can, and cannot, move. The second one is to calculate a path between the character and some point in the world. In case there is no way to get to that specific point, the pathfinder will return an empty path. The path is generated as a list of nodes. The character will visit each node in order, finally reaching its destination at the final node. By using these nodes, which are essentially straight lines in the bigger path, the character can move around the world whilst avoiding solid objects it should not be able to walk through.

To find the shortest path between two points the pathfinder uses a A* algorithm specialized on finding the shortest path on a 2D-grid. This path will be converted into a list of nodes that is returned by the pathfinder.

3.5.2 Updating the Game World

In this game it was decided that the update function should normally run 60 times per second. The update function loops through every dynamic object in the world and executes a specific update method for each and every one of the different objects. These update functions should do different things depending on what kind of object it is; trees will spread, crops will grow, characters will move, and so on and so forth.

3.5.3 Displaying the Game World

To make the game interesting as well as understandable, sprites were utilized to display the world graphically. The project took imagery from the web that is licensed under Creative Commons. The images used in the game that were neither made for this project nor public domain can be found in appendix A. These images were then applied to the objects in the world, such as vegetation, structures or characters; so that it could be seen on the screen. A screenshot of the game can be seen in figure 3.3.

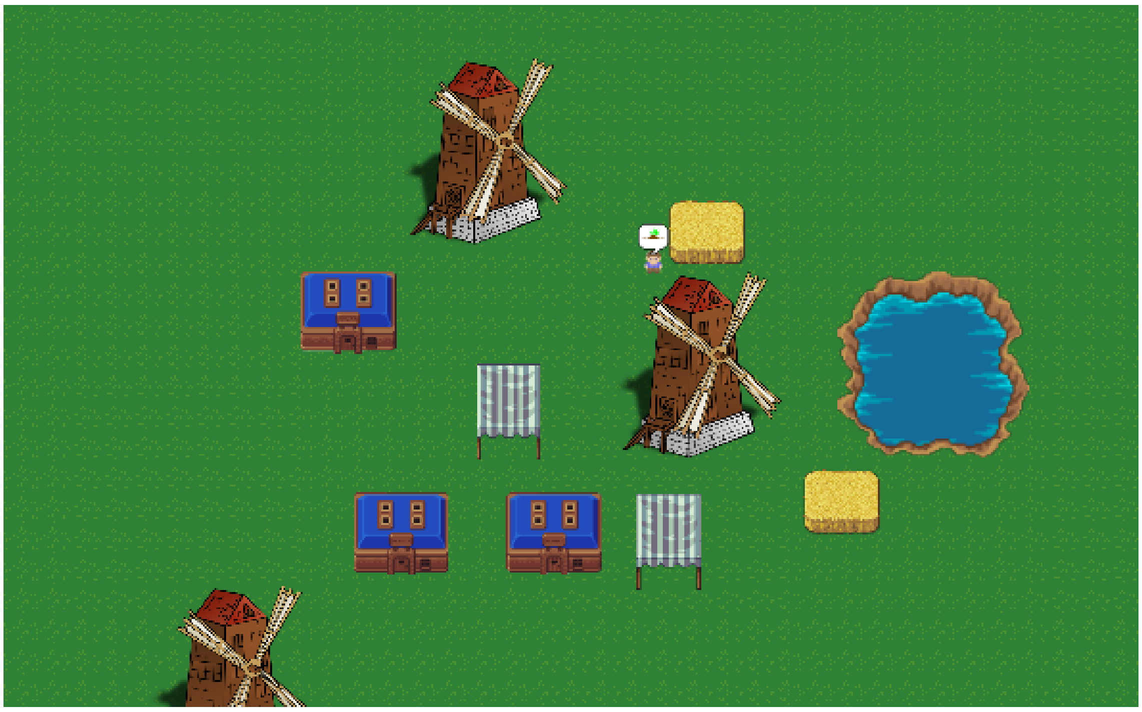


Figure 3.3: A screenshot from the game displaying a NPC sowing crops at a farm in a small village. References to the creators of images used in the game can be found in appendix A

4

Development of the AI

This chapter will focus on how the AI was developed and implemented into the game. It will cover early testing, how an early stage FSM was implemented as well as how it developed over time. This chapter will only cover the development of the actual implementation. Comparisons and thoughts on how additional approaches might have been used can be found in chapter 7.

4.1 Preparatory Literary Study

Entering the project all members were more or less inexperienced with Artificial Intelligence. Learning more about AI in general, as well as the standards used in the industry, was a top priority. This included reading about AI in popular games, how they were used and how to implement them. Since AI is such a broad field and implementations vary in so many ways, focus was on the most popular approaches used in the video gaming industry.

4.2 Early Experiments

Early in the development process, before the characters were complete enough to be controlled by an AI, a prototype problem solver was implemented. The goal of the prototype was to see what it would take to implement an actual problem solver AI later on. The prototype could take an abstract model of a character and a list of possible actions the character could perform and then calculate what sequence of actions would lead to the optimal outcome. Optimal in this case refers to keeping the character's needs as satisfied as possible, as well as collecting the maximum amount of resources. Though limited, the prototype worked satisfyingly.

4.3 Early stage FSM

Development of the FSM implementation started halfway through the project. The game had enough of the basic elements required to test the functionality of the AI, however, it was not complete. From this point in the process the AI and game functionality was developed in parallel. In many cases, when new functionality was added, support for that functionality were implemented in the AI as well.

A basic AI was desirable as it could be used to test and confirm that the most trivial logic of the game was working. A primitive FSM was implemented where the AI would only have a few states for collecting resources. This early implementation was, as stated, a good start to find bugs in the program and to have something to expand upon. Only a few resources were available for gathering to satisfy the needs of a character. There were no AI methods for constructing different buildings or socially interacting with other characters.

Additional functions were added to the AI continuously. As new things like construction and more resources were added, states were implemented in the AI that could handle these new functions. This kept bugs and other issues to a minimum as they could be fixed before going on to the next feature.

4.4 Development process

The first step was to make a blueprint, seen in figure 4.1, of the states that were needed, as well as mapping transitions between these states. State transitions were designed hierarchically, meaning that states could only transition from a higher level in the hierarchy to a lower level, or vice versa, never between states on the same level.

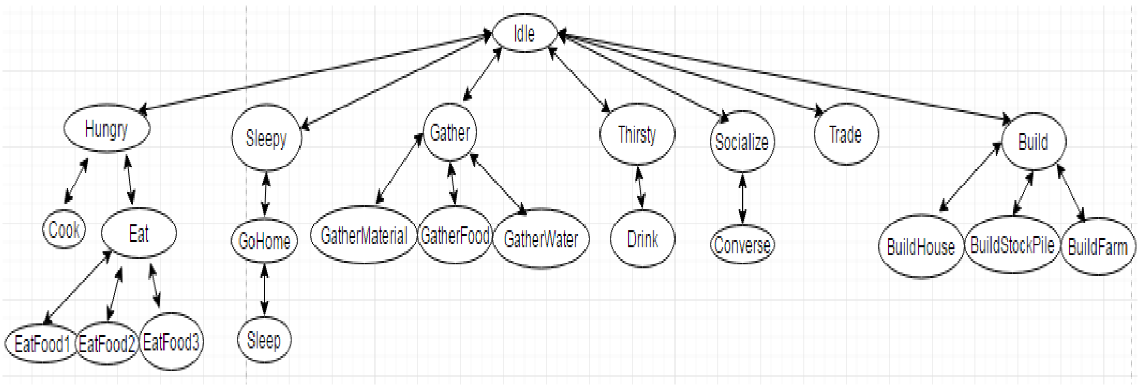


Figure 4.1: The state transition blueprint, arrows represent possible transitions, bubbles represent states.

This was the envisioned state machine, what the blueprint looked like before the first line of code was written. At first glance it looks a lot like a behaviour tree, but it differed from a common behaviour tree in the sense that there was very little planning at the node level. Additionally, it also had to transition to idle state before going deeper in the tree structure, meaning this implementation can disregard levels of the tree when transitioning [35].

When the blueprint was laid out, an *ArtificialBrain* class was made along with classes that represented each state. The *ArtificialBrain* is the main class of the AI, it contains all the AI variables along with the functionality to make transitions between and executing each state. At this stage, queues were used to control the flow of states, they were later replaced with stacks. The idle state is at the top of the hierarchy and was intended for planning the AI's next move.

At this point the AI could walk, eat, sleep, drink and gather resources. Although simple, it was advanced enough to test how additional features not essential to survival played together with the essential features. A list of the states implemented in the AI at this point in the development process can be found in appendix C.

The AI had perfect information about the world, meaning it knew where everything was at all times. There was no functionality for social interaction, reproduction or building houses yet. From this point the AI was developed iteratively as more elements of the game were finished and could be used by the AI.

4.5 The AI Class: ArtificialBrain

The initial implementation of the *ArtificialBrain* was simple. It had an *update()* method that was executed by the controller once every iteration. It was decided that each state should have its own *run()* method which determines the AI's behaviour. This method should be executed from the *update()* method of *ArtificialBrain*.

As stated in the previous section, the AI had perfect information about the world. As perfect information is not a realistic representation of a human's perception of the world, the AI should not have this kind of knowledge. For this reason a simple form of memory was implemented in the AI. This memory was a list of objects that the character had walked past in the world. As structures were implemented, this solution was used to keep track of their position as well.

As new states were implemented and the AI became more advanced, new tools had to be implemented in the *ArtificialBrain* to accommodate for the increased complexity. These tools were stacks and lists, intended to store information needed by certain states, as well as allowing the AI to perform sequences of several states, giving it a basic sense of planning.

To increase performance, it was decided that the supervision of primary needs, that was previously in the idle state, was to be moved to the *ArtificialBrain*. This removed the need for the AI to enter the idle state to gain access to this functionality.

4.6 Transitions Between States

A state transition is moving from one state to another. Without them, the AI would never change state and consequently never do anything.

In the early implementation the transitions were designed to be hierarchical, meaning that the AI could only transition from a higher to a lower level and back again, in the hierarchy of states. It could not transition between states on the same level. This design was used to keep the transitions neatly structured but as the FSM grew more complex, there was a need for some states to break this structure. The transitions were kept as hierarchical as possible, giving some states the ability to transition directly to states that would not have been possible in a pure hierarchical design or behaviour tree. Additionally, the idle state was initially always used to perform state transitions.

5
Final Implementation of the Game

The implementation of the game resulted in a functioning world which the NPCs could exist in and change. The NPCs were able to gather water from the ponds and gather food, either by killing cows, fishing in the ponds or farming crops. They were also able to construct the three different structures described in subsection 3.4.2. The NPC could also collect various amounts of the raw materials wood and stone in order to construct these structures. Of the original 43 functional requirements in the software description, all but 8 were successfully implemented in the game. These 8 were:

- 44. Characters should have skills.
- 45. The skills should increase when used.
- 50. Characters should be born.
- 55. The AI should make decisions based on the current goals of the character.
- 56. The effect on the needs and wishes of the character, that the traits and
- skills have, should affect the AI's decisions.
- 58. The AI should take the actions of other NPCs into account when deciding on actions for the NPC.
- 63. The world should have a basic weather system.
- 69. The number of houses in the village should set the max population of the village.

The majority of the envisioned concepts in chapter 3 were implemented. However, Skills, Goals and three Traits, were not implemented due to time limitations. The three unimplemented Traits were Lust, Greed, and Pride.

Most of the classes in the UML-diagram were implemented, however a few were removed for various reasons. Some minor changes were also made to the UML-diagram during the course of the project. The final UML-diagram can be seen in figure 5.1.

In the finished game most of the NPCs could successfully live long enough to die from old age and before that had time to build a house, farm and stockpile. A screenshot of the finished game with explanations can be seen in figure 5.2.

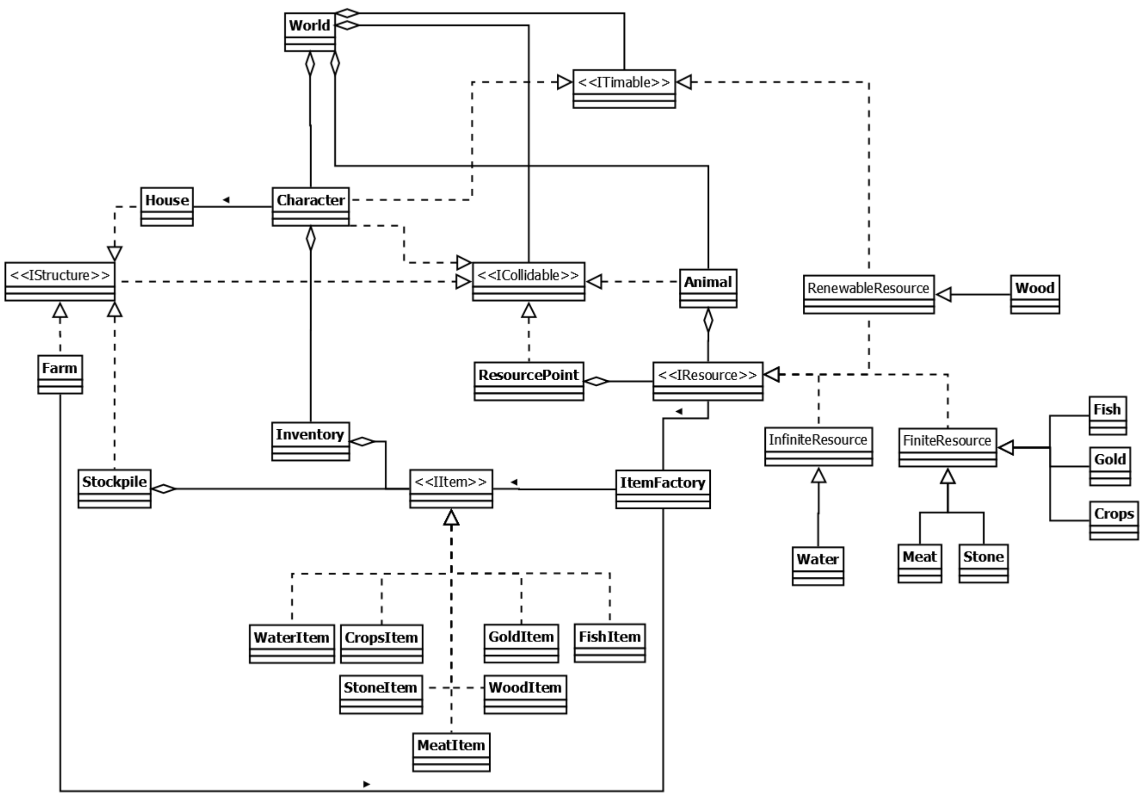
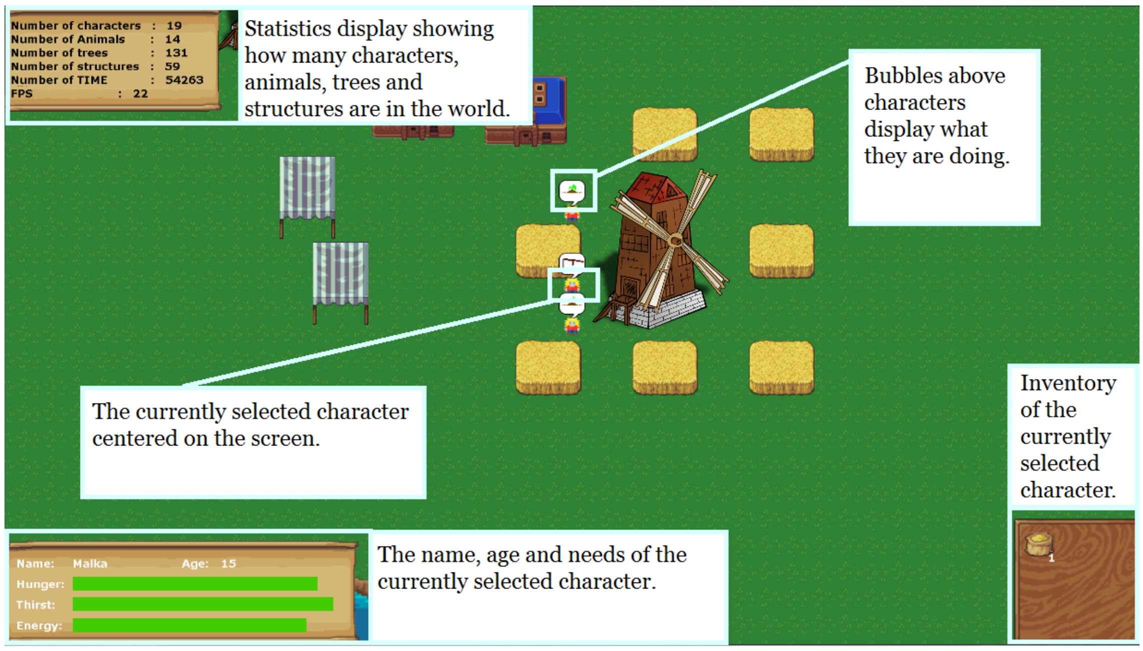


Figure 5.1: The final UML-diagram of the model made in this project.



Number of characters : 19
Number of Animals : 14
Number of trees : 131
Number of structures : 59
Number of TIME : 54263
FPS : 22

Statistics display showing how many characters, animals, trees and structures are in the world.

Bubbles above characters display what they are doing.

The currently selected character centered on the screen.

Inventory of the currently selected character.

Name: Malka Age: 15
Hunger:
Thirst:
Energy:

The name, age and needs of the currently selected character.

Figure 5.2: A screenshot from the game with explanatory text explaining the different parts of the game. The screenshot displays three NPC working at a farm in a small village. References to the creators of images used in the game can be found in appendix A.

6

Final Implementation of the AI

The final implementation of the AI is a “stack based finite state machine”. The AI has a set of states that determine its behaviour depending on which state it finds itself in, at a given time.

The AI is controlled by the class *ArtificialBrain*. This class can, as the name suggests, be seen as the brain of the character. The brain class is updated and executed every controller update and contains all the functionality for executing each state. The brain class is also responsible for checking if the character is hungry, thirsty or tired. If so, the AI will enter the state corresponding to the behaviour necessary to address this. For instance, if the character is hungry, the AI will stack its current state, allowing it return to it later and transition into the hungry state.

Each state is an individual class that has its own unique methods that represents the actions an AI can perform, these methods are then executed in the *run()* method. The *run()* method of each state is called by the *update()* method in the *ArtificialBrain*. A complete list of the states in the AI can be found in appendix D.

The *ArtificialBrain* consists of several components that are used to model the behaviour of the AI, these components are:

- **A stack of states** - This stack contains all the states that are waiting to be executed. This is used to control the flow of the states. Some actions require several states and transitions to be performed, for instance reaching a satisfying level of hunger, or building something. By using a stack it can push all the states required to perform an action and then execute them in order. It is also used as a means for the AI to be able to remember what it was doing previously, in case it has to interrupt itself to do something of more importance.

- **A stack of structures** - This stack contains all the structures that are currently due for construction.

- **A stack of objects to move to** - This stack is used to remember where the NPC was going when returning to a stacked MovingState.

- **Two stacks of resources** - One stack, the FindResourceStack, contains all the resources that the AI currently wants to gather, but does not have in memory. The other, the GatherStack, contains all the resources that are necessary for a stacked state. In gather state, the AI will gather the first resource on this stack. If the AI does not know where the first resource on the GatherStack is, it will push that resource onto the FindResourceStack and go look for it.

- **A resource memory** - This is a list of resources that the AI has walked past. The AI remembers where these resources were so that it can easily find them again. For instance, where the closest lake that it knows of is.
- **A structure memory** - This a list of structures that the AI has walked past. The AI remembers where these structures are so that it can easily find them again. For instance, where the closest farm that it knows of is.

The idle state is the state the AI enters between states or when it has nothing to do. It is also used for popping states from the stack. If it has states on the state stack, the first state is popped once the idle state is executed and the AI transitions into that state. The *run()* method of the popped state is executed, and when the state reaches the end of *run()*, the AI will go back into idle state and pop the next state from the state stack. If it does not have any states on the stack, the AI will pick another task randomly based on where it is in its lifecycle. For example; if the NPC has not yet built a house, and is lacking the resources to do so, it will choose to gather these resources unless there is something of more importance to attend to, like managing primary needs.

The AI will always start by building a house for itself. If the AI has a house it will proceed to build a stockpile and a farm, the order in which these structures are built is random. However, the AI will only build a farm if there is no farm present in its structure memory. A farm requires a lot of resources to build, and many characters can work on the same farm. Therefore it would be unnecessary to build a farm if one has already been built and it knows where that farm is. If the AI has built a stockpile and has a farm in its memory, the AI will either hunt for food, gather a random resource or work on a farm for crops.

In the beginning of its lifecycle, the AI follows a simple, hardcoded plan. The AI will start by checking if it has a home, i.e. somewhere to sleep. If it doesn't, its first task will be to build one if it has the materials required. If it doesn't have the materials, it will gather them. It does not make plans in advance or prepare for the future. It simply reacts to its current state and does what is necessary to proceed. When the AI is finished with these predetermined actions, it will gather a random resource.

The AI will only interrupt itself in a task if it recognizes one of its primary needs is under a certain threshold. If so, the *ArtificialBrain* will push the AI's current state onto the state stack, saving it for later. It will then force the AI into a state increasing the corresponding need. When the AI has finished eating, drinking or sleeping, it goes back to do what it did before it got hungry, thirsty or tired.

The AI has almost no ability to plan for the future. The AI will only take on a task if it is what is necessary at that moment in time, and as can be seen earlier in this section, sometimes it is completely random. The only thing the AI plans for are the states needed in order to complete a certain task, which could be argued not to be planning at all. For instance if it wants to gather a certain resource that it has in memory, the AI knows that it will have to stack the moving state to get there, and the corresponding gather state for that resource in order to gather it once that resource has been reached.

7 Discussion

The desired behaviours of NPCs in video games differ wildly. It is difficult to say that there is a best approach to AI for NPCs. With our limitations we have narrowed our scope down to NPCs in survival games. Still the survival genre is too broad a field to raise one AI approach as superior.

In this section we will first discuss alternatives to creating a game from scratch, and features that were left out of our game. This is followed by a discussion of believable contra realistic NPCs. Finally we will go through three different AI approaches that one may use to implement an NPC in a video game. We will discuss the advantages and disadvantages from our own experiences in development, as well as what we have learned about implementing realistic NPCs through the course of this project.

7.1 Making the game from scratch

We decided to make the game ourselves from scratch to retain control of what features we could implement. The group was in general interested in creating a game from scratch and saw this project as a good opportunity to explore this area.

Other options were considered, such as Unity or Unreal Engine, which could have been very good options since they both have very good AI frameworks that could have streamlined the process. Unreal was discarded because we felt like the real power tools of it are for 3D games, and since we wanted to have our game very simple, we wanted to make a 2D game. Unity uses C#, which is an object oriented language, but the group had no experience with either C# or Unity. What we had experience with though, was Java, and a few members of the group were already experienced with game development using this language. We realized that making the game would take quite some time, and we wanted to hit the ground running to finish the game as soon as possible. Hence, Java was the language we chose.

As creating a game is no small task, it might have been easier, and quicker, to create a modification (or mod for short) to an already existing game with good support for mods. However, the choice of game would limit us to using the tools of that game, and the AI to that world. So while it might have been quicker, it would have been necessary to research the code of said game, it would also have been a limiting factor to what we could achieve. A benefit of creating a mod would have been that we would be able to compare our result with the original NPCs.

7.2 Scrapped ideas

Entering the project there were many ideas concerning how we could make the AI more realistic. We found many of these interesting and we think that they could have given a lot of depth to our AI. Unfortunately, many of them had to be discarded for various reasons. This chapter will cover those features, what they were intended to accomplish and why they had to be tossed.

7.2.1 Social Interaction between characters

To increase the sense of realism and immersion in the game, we wanted to give the NPCs the ability to socially interact with each other. Social interaction was supposed to be a tool for the NPCs to build relations. One NPC talking to another would either increase or decrease the relationship status with that NPC, depending on how similar their traits were. In this way NPCs could make new friends or enemies. This would have given the AI more depth.

Social interaction was also supposed to be a tool for the AI to gain more information about the world, without having to go out and explore everything for itself. When two NPCs would talk to each other, they would exchange information about the location of resources and structures.

Socialize was supposed to be a secondary need that, when low, would negatively affect the character's ability to work and possibly lower the amount of energy gained from sleep. Interacting with other characters would replenish this need. The effect of a low level of the socializing need would be determined by the traits of the character. For instance, if a character had traits making it more of a lone wolf, it would not be affected negatively by isolation from other characters.

A lot of work went into making the idea of social interaction a reality, it was something we wanted to work because it was a prerequisite for many other features. Among these features were reproduction, relations and trading. There was partial success. We managed to get two NPCs to talk to each other, but as soon as a third one wanted to join the conversation, the game would break. This issue proved complicated to solve due to the way that the software architecture was designed. In the end, we decided to scrap the idea because time was running out and there were still many essential features that had not been implemented.

7.2.2 Relations between characters

We wanted the NPCs to be able to build relations with each other. Relations were supposed to be the base for how an NPC would respond to another in social interaction. An NPC was supposed to be helpful towards friends and unhelpful towards enemies or NPCs that NPC did not particularly like.

Relations were also going to be used for family trees, the AI was supposed to know who its mother, father, brothers, sisters, sons and daughters were. Additionally relations were intended to allow for reproduction. If a man and a woman got to a specific grade of relation they would get engaged and eventually have children. Since social interaction never made it to the final implementation of the game, there was no good way of building relations. Thus they were never implemented.

7.2.3 Character skills

We wanted the NPCs to have certain skills that would make them more useful in certain areas than others. When performing a certain task, the NPC's skill in that task would increase. It would also decrease over time if the NPC did not perform the task regularly. Some intended skills were; fishing, farming, trading and exploring. If an NPC was particularly good at fishing, it would net more fish than other NPCs with a lower skill level. However, skills had low priority and were never implemented.

7.2.4 Long- and short-term goals of Characters

Another possible extension of the NPCs were long and short term goals. Achieving these goals would increase its overall happiness and would give more depth to the game. Short term goals could be: build a house, make a new friend or something along those lines. Long term goals would require more effort to fulfill and would not necessarily be something the AI could achieve in its lifetime. Examples of intended long term goals were: get married, have three children or become the richest person in the village.

7.2.5 Trading resources between characters

Trading was supposed to be a type of social interaction where one NPC would propose a trade with another, that NPC would then assess if it was a fair trade and choose to accept or decline it. The idea was that NPCs that had a lot of a particular resource could trade that resource for something else that it needed, removing the need to always gather everything. This would have played well together with the use of skills, as proffessions could arise naturally. For example: an NPC skilled in lumbering could trade wood for fish with an NPC that had a high fishing skill. Trades would have been influenced by relations were an NPC would be more likely to accept a trade from a friend. Trading was not implemented since social interaction was discarded.

7.2.6 Reproduction of characters

A vision was that the AI would be smart enough to sustain its society forever. Once the game had started running, the NPCs would never go extinct. A prerequisite for this is of course that the AI is able to reproduce, otherwise extinction is inevitable. Since relations were never implemented in the game, neither was reproduction, rendering this vision unachievable.

7.2.7 Lack of Test Data

With the purpose of implementing a realistic AI in a video game, it would have been helpful to define tests to verify that the result fulfilled this purpose. These tests would have been designed to prove that the AI we created fulfilled the definition of a realistic AI to a satisfactory degree. Unfortunately, such tests were never explicitly defined, and therefore never performed on the final product.

The reason such tests were never defined was simply that in the early stages of the project, when such tests should have been defined, we were focused on the literary study. Therefore, when we realised that such tests should have been defined, it was too late to implement or conduct them in any reasonable manner.

7.3 Are Our NPCs Realistic?

We lack the data to be able to confidently state that our NPCs are more believable or realistic than NPCs in games currently on the market. However, several strong points can be made as to why our NPC make a good proof of concept for a realistic NPC. In Maslow's hierarchy of needs the motivators of human behaviour are placed in a hierarchy. The idea is that the lowest unfulfilled step of the hierarchy drives the actions of the human. The lowest two steps of the hierarchy Maslow proposes is “physiological needs” followed by “safety” [34].

Our NPCs acts according to the bottom two of this hierarchy. When they have pressing hunger, thirst or energy (physiological needs) they will tend to these needs. If they are satisfied physiologically they will work on security, making sure that they have somewhere to live and that they will have food for future use. The further introduction of social interaction and skills could have further led to the inclusion of two more steps of Maslow's hierarchy: "Love and belonging" and "Esteem" [34].

One can argue that our NPCs achieve a high level of realism, relative to their simplicity (being incapable of doing most things a human can do). However we would argue that they make a good proof of concept, being realistic in performing the actions they are capable of. As such, they show that better realism for NPCs is achievable. Worth noting is that the AI in the NPCs is a version of an FSM, which then shows that realistic NPCs can be made using approaches already known to the industry.

7.4 Choosing Between Believable and Realistic

It can be a difficult task to make the decision whether one should have a realistic or believable AI in a video game. One has to take into account what the AI should accomplish and the amount of computational resources it should be allowed to use. An AI with focus on realism will require much more computational power than its believable counterpart; since a realistic AI has to do a lot more planning and decision making. Believable AI does not suffer from this to nearly the same extent, as its behaviour is often predetermined.

A well-implemented realistic AI will undoubtedly give the player a deeper sense of immersion, since the AI will act and behave as a regular person would. In some games however this might be an undesirable behaviour. In Role Playing Games (RPG), the player takes on a role of a character in a fictional setting. In these games, NPCs play a big part in the player experience and they are essential for the player's progression through the game. They give the player missions, serve as storytelling elements, they purchase and sell items; the list goes on.

If these NPCs were entirely realistic, meaning they would have to eat, drink and sleep, just like human beings, the game could quickly get tedious. Imagine if you needed a specific quest to progress through the game and the quest giver, a character which hands out missions, is not where you expect him to be. Perhaps he has gone hunting and won't be back for a couple of hours, or perhaps he has taken a stroll down to the local inn to get something to drink. What if the quest giver dies? This would certainly be a very immersive experience, but it would slow down the pace of the game tremendously and could leave the player stuck with nothing to do for some time. It might even break the game completely.

In order to have realistic AI in this type of game, a game designer would have to put much bigger thought into the overall design of the game. In these days where game developers are often limited by tight budgets and time schedules, realistic AI is rarely a prioritized design choice.

If we take a look at the other side of the coin, what if NPCs are not realistic; which is the case in most games. Here the player will always be able to progress, as he can always be sure to find the quest giver where he expects him to be. The game designer does not have to take into account the difficulties that comes along with realistic AI. The AI will always behave as the designers expect it to behave, there is no risk of the AI running into a situation that might not be accounted for (given that the developer has not made a mistake somewhere, of course). However, a stationary quest giver that stands in the same position every second of every day could arguably be an equally undesirable feature, as it takes away the immersion a realistic AI provides.

As we can see, the choice is anything but easy when it comes to realistic or believable. A lot of games commonly end up somewhere in between the two, with developers using both realistic and believable elements to shape their AI.

7.5 Ways of Overcoming Obstacles of Realistic NPCs in the Industry

Consider the situation where you have a character that is important to the story of the game. If the character is realistic it may die before playing its part in the story leading to one of two outcomes. Either the character is revived, which leads to disbelief, or the character remains dead and therefore a part of the game, maybe an important one, is lost to the player.

Immortal characters can be observed in the game Skyrim in which important NPCs will not die if damaged to zero health. Instead the NPC will collapse and remain still until it regains its health. In this case it not only causes the game to be unrealistic, but you can also use this to your advantage by sending your NPC allies to attack the enemy without approaching the enemy on your own. Since your allies cannot die and will regain health after having been knocked out, they will always win the fight eventually.

Another common tactic is counting a death of an essential NPC as a death of the player. This will keep the player defending the NPC and if it dies the game would usually return to a state before the essential NPC's, death. Another more immersive solution can be found in the game Morrowind, a prequel to Skyrim [36]. In Morrowind the essential NPCs are treated just as any other NPC concerning death, with one small difference; the game will warn you when a essential NPC have died (something that rarely happens thanks to the game's design) and advise you to reload a previous save in which the NPC is still alive. This way the game retains its realism and gives the player freer hands in how to play, while still keeping the option to play the game with all of its essential NPCs retained.

Considering this, we would suggest using realistic NPCs when NPC behaviour is a central part of the game. Games that often fall in this category are Role-Playing Games (RPG) and simulators. In other cases believable NPCs, imitating human behaviour should suffice.

7.6 Finite State Machines in Video Games

Finite State Machines is the most popular approach to AIs in video games [37]. There are two main reasons for this: First there is tradition. FSMs have been popular in the video game industry for a while and as such a lot of the tools used in the video game industry supports FSMs more than other AI approaches. Secondly FSM have many qualities appreciated by video game designers a few of which will be discussed further on in this section.

7.6.1 Implementing NPCs using FSM

How an FSM is used to make AI for NPCs varies from game to game. Sometimes the FSM is used in cooperation with other AI approaches. However, there are a few basic ideas that stick around.

An FSM decides the exact behaviour of a NPC from the states and transitions that have been programmed. Because of this, one usually implements one unique FSM per NPC type. For example all animals in a game could use the same FSM, if the behaviour of one species does not differentiate from another, though that same FSM would probably not suffice for the NPCs representing humans.

7.6.2 Advantages of Finite State Machines

There are good reasons why FSMs have become so popular in the video game industry. Since an FSM is, by its very nature, easy to divide into parts, this makes workload easy to divide as well. For example one programmer could program the state transitions while two other program the different states. Through their simplicity, FSMs are good for setting up a simple prototype that can be made more advanced by adding more states later on. As a lot of time went into making the game world, the fast development pace FSMs allow for proved very useful to us.

Since the behaviour in each state as well as the transitions follow a strict set of instructions, it is easy to implement an NPC with distinct behaviour. The deterministic nature of FSMs makes it easy to know how a piece of the program will affect the behaviour of the NPC, thus changes are easy to make. If one wants to make tiny changes to a certain part of the NPCs behaviour, that is easily achieved by redesigning the concerned states. For more advanced changes one could simply add more states to deal with the specific problem at hand. This aspect made it easy for us to make changes to the AI as more states were introduced, and existing states had to be reworked in order to accommodate for changes these new states brought along.

7.6.3 Disadvantages of Finite State Machines

As the name suggests, FSMs have finite states. Specifically they only have as many states as the designer implements. While this approach works great for simple NPCs in simple circumstances, it poses problematic for more complex situations.

Since each state depends on the circumstances of the NPC, the FSM might often lack states for the many different situations that can occur in a complex environment. Even if a state is meant to cover a given situation, the state may be too general and not always suit the situation given. You could theoretically expand an FSM until it encapsulates every improbable situation. However, in complex enough environments, this is an almost impossible task.

The other problem that arises is the growing complexity of the FSM. This is easily observed by considering the blueprint of an FSM. For each new state created a designer will have to consider which other states should be accessible from this one. For an FSM with two states there are two possible moves to be considered, for three states there are six moves, and for four states there are twelve, hence it grows quadratically. This is not something that ever became a problem for us, as we countered it by making our FSM hierarchical.

7.7 Our Finite-State Machine Implementation

Our final AI implementation is, as previously stated, an FSM. Though it is not a pure FSM, but rather a Stack-Based FSM (SFSM) which is a more common construct in video game AI [38]. What is referred to as FSM in video game development is rarely a pure FSM according to its mathematical description. More often the FSM used is a more complex design. Common designs are Hierarchical FSM or, like in our case, a SFSM [39].

We decided to use an FSM for our AI. There are a few reasons for our choice of using FSM rather than any other AI approach we had studied. One reason was time. Our game had been progressing slower than anticipated and we needed to have a working AI as fast as possible. We reasoned that FSM was the fastest way to get a working AI up and running. The lack of time also forced us to start working on the AI before the rest of the game was finished and the expandability of a FSM made it easy to add more complexity to the AI as the game was being developed.

The use of an SFSM gives the advantages of having an FSM but with the additional power of a stack memory. The NPCs could use the stack to push their current state when they decide to do something else. This way they remember what they were doing when they complete their new task. For example an AI could go into a build state because they need a house. In the build state they could realize that they do not have enough resources to build a house. The natural conclusion is to transition into a gather-resource state. In this state the AI gathers the resources it needs and when it is finished it needs to transition into a new state. With a normal FSM, the AI could not know why it had gathered resources, with an SFSM it can. The NPC, now finished with the gather state and with no reason to transition to a specific state, pops the last state pushed onto the stack. This way it returns to the build state and can finish building its house.

We chose this implementation over a regular FSM as we deemed it gave us greater control over how the AI behaves. It allowed us to work with sequences of states, rather than the traditional way of letting each state determine what state comes next. These sequences are good, because we can easily make sure that the AI does things in the order that we have intended. An ordinary FSM could give us this behaviour as well, but with the use of state sequences we could neglect a lot of the code that would have been necessary in a regular FSM to prevent it from doing things it was not supposed to. For example, if each state was to decide what the next state was, we could never be sure that the next state the AI entered would be the one we intended. Sequences removed some of the difficulty of having to figure out why the AI did something unexpected.

This implementation could be argued to be bad design, because it comes with a lot more hard coding and predetermined behaviour, but for the results we sought this was satisfactory.

7.8 Problem Solving In Video Games

Problem solving was a popular approach for creating game AI early in the field of Artificial Intelligence. Though because of the brute force nature of problem solvers the purely problem solving approach was soon abandoned for more efficient approaches [40].

7.8.1 Implementing NPCs Using Problem Solvers

What we know of problem solving comes from a mixture of studying the AI approach in literature, developing an early proof-of-concept AI and an, due to lack of time, unsuccessful attempt to make a full problem solver for the game. Despite this, or maybe because of it, we feel that we learned a lot of the advantages and disadvantages of using problem solver to create an AI in a video game.

For an NPC to be able to use problem solving requires a lot of structures in the game itself. There needs to be a way for the NPC to check what actions are, and what actions will become, available (or unavailable) by performing each of these actions. The NPC needs to know how to value its state and maybe even the state of the world. This requires structures and standards for what actions there are, what objects are interactable and what a given action would yield. The AI will be required to simulate the use of each possible action and be able to simulate even further what actions are possible after that one. There need to be a finite types of actions. For each action type there need to be a way for the AI to tell the outcome of this action to be able to search further down the tree of possible routes of actions.

In the game we implemented much of the framework required for a problem solving AI, as well as began work on the problem solver itself. We divided our actions into three types, not only for the problem solver but for keeping similar interfaces for the FSM as well. The three types are; character bound, item bound, and environmental. The interfaces differed, out of necessity, although it would have been possible to have an AI which interacted in the same way through each action.

A character bound action represents an action a character does with its body only interacting with itself. The example from our game is that a character can at any time sleep on the ground. It is an action that can be performed simply by deciding to do it.

An item bound action is an action that is bound to some item the character is carrying. They differ from the character bound in that they require the NPC to have the given item and in some cases using the item will consume it. An example from the game would in this case be eating a fish. A fish is something you must have to eat and eating it would hinder the NPC from eating it again.

Lastly an environmental action is a action bound to an object that is part of the environment or surroundings. This could be any action that needed or affected a part of the environment. An example from the game would be chopping wood from a tree.

The separation of item- and environmental actions was due to the fact that to perform an environmental action an NPC would need to move to the object they wish to interact with. Although similar actions, interacting with another instance of the world, the programming required differed heavily because of the need to move the character into position.

7.8.2 Advantages of Problem Solving

Problem solving can, given a good interface with the world and a good value function, calculate the best approach to a given situation. It frees the programmer from the task of figuring out how the NPC should behave in given situations and focus instead on the NPCs overall goal. A problem solver can take into account the small differences between slightly different states and take different actions accordingly.

A problem solver can plan far ahead and therefore think of long term goals. This could mean the AI decides to take several unbeneficial actions in a row because it will allow it to take a more beneficial action later on.

Personality and personal goals can be easily implemented into the NPCs as a change in the AIs value function. Different NPCs value the state of the world differently. For example an NPC with high greed may have a function that considers the value of money or gold higher whereas a gluttonous AI values food more. A more advanced value function may consider the state of the other NPCs. If an NPC like another they could value the state where the other AI is satisfied more and thus help that NPC reach that state. Conversely if an NPC has an enemy they may value a state higher if it is bad for their enemy.

7.8.3 Disadvantages of Problem Solving

Problem solving requires certain structures in the game to be viable. It is also difficult to implement and test a problem solver if the base of the game is not yet finished. A problem solver can also be rather heavy on processing if it needs to plan too often or if the game has such great complexity that very deep searches are needed to find satisfactory nodes.

The necessity of a value function is another drawback as it in some cases might be difficult to know how to value a given state and the sole pursuit of increasing that value might lead to the NPC neglecting everything else. Other problems might arise if it is difficult to model the changes of the world within the AI. Since it is extremely ineffective to model the entire world within the AI the states of a problem is often a more abstract version of the world. If the conversion of the world into a state in the problem is done poorly the AI is likely to make some bad decisions. However one could argue that this is how human works as well; we don't make perfect decisions because of our imperfect perception of the world.

Problem solving is also a rather difficult AI to predict or make small changes in. Whereas in an FSM you can modify the specific behaviour in a given state or change the exact transition between states a problem solver is much more limited when it comes to detailed control. The value function could be slightly changed as well as the perception of the world but the AI might still not behave satisfactory. This also makes the AI less predictable which is important in some games.

7.9 Machine Learning

The possibility of using Machine Learning algorithms in our AI have been discussed, however due to time restrictions we were unable to implement this in the final product. One of the implementations we considered was using an unsupervised learning method to model the relations between the NPCs. The idea was to use the "k-means" clustering method on the population, represented by their traits, to split the population into k different groups. By making this calculation for every NPC, every one of them would get their own idea of what characters belongs to which groups. This is due to the fact that the k-means algorithm starts off by randomizing starting values.

This method can be extended by adding an extra parameter, representing the impressions other NPCs have made on the character in question. This value should be able to change over time, as characters should be able to make new impressions. By using this parameter, along with the traits when defining clusters, the NPCs will get more differentiating ideas of the different groups. This would also allow for the clusters to change over time, giving the NPCs a dynamic idea of what group they want to associate themselves with.

This way of thinking can be considered realistic as it depends on what impressions you have of a person as well as what their personality traits are. The fact that the characters wants to associate with a certain group, rather than single individuals, makes it so that the opinion it has of others is affected by its current group of friends. The advantage of this model is that it is a more complex representation of relationships and as such is able to model more complex relations. The computational complexity of this model is naturally higher than that of representing each relation with a scalar, but as the relations would not change often this should not be a issue.

8 Conclusion

AI is a very broad scope and even in small projects like this it takes a lot of time to develop an AI that we consider is intelligent. We managed to develop a game world as well as an AI inspired by realism. Not only did we get working NPCs but also a behaviour that somewhat simulates a society in development. The final implementation of the game we created contained a hierarchical FSM with a state stack as the AI approach for controlling the NPCs.

We wanted to create the other approaches that are discussed throughout this report as well, in order to implement more advanced elements for the AI, such as relations and planning. Our notion is that the problem solver would have proved to be the better implementation and would have given a more interesting behaviour, as it allows for better planning and rational, dynamic decision making. However, the FSM implementation is still considered, by the group, to be the right choice as a first implementation, as this was the easiest to get running and still achieved a good base for us to assess the results on. In hindsight, the group agrees on the fact that developing a game world was something that took more time than we anticipated.

The main focus of this project was the AI, thus we think that using a pre-existing world or an engine, like Unity or Unreal Engine, where a world can be created fast, would have been a better option. The time saved could have been devoted to AI development, consequently leading to us being able to explore additional approaches and compare them.

Bibliography

[1] "Login | Deep Dream Generator", Deepdreamgenerator.com, 2016. [Online]. Available: http://deepdreamgenerator.com/generator. [Accessed: 16- May- 2016].

[2] “Global Games Market Revenues 2016 | Newzoo", Newzoo, 2016. [Online]. Available: https://newzoo.com/insights/articles/global-games-market-reaches-99-6-billion-2016-mobile-generating-37/. [Accessed: 26- Apr- 2016].

[3] T. Quandt and S. Kröger, Multiplayer: The Social Aspects of Digital Gaming. New York: Routledge, 2014.

[4] M. Buro, "Call for AI research in RTS Games", http://www.aaai.org/, 2016. [Online]. Available: http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-028.pdf. [Accessed: 31- May- 2016].

[5] J. McCarthy, M. Minsky, N. Rochester and C. Shannon, "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955", AI Magazine, vol. 27, no. 4, p. 12, 2006.

[6] M. Childs, "John McCarthy: Computer scientist known as the father of AI", Independent, 2011. [Online]. Available: http://www.independent.co.uk/news/obituaries/john-mccarthy-computer-scientist-known-as-the-father-of-ai-6255307.html. [Accessed: 28- Apr- 2016].

[7] "Machines Who Think", Pamelamc.com, 2016. [Online]. Available: http://www.pamelamc.com/html/machines_who_think.html. [Accessed: 16- May- 2016].

[8] R. Redheffer, "A Machine for Playing the Game Nim", The American Mathematical Monthly, vol. 55, no. 6, p. 343, 1948.

[9] W. Saletan, "The triumphant teamwork of humans and computers", Slate, 2007. [Online]. Available: http://www.slate.com/articles/health_and_science/ human_nature/2007/05/chess_bump.html [Accessed: 15- May- 2016].

[10] C. Metz, "Google’s AI Wins Fifth And Final Game Against

Go Genius Lee Sedol", Wired, 2016 [Online]. Available: http://www.wired.com/2016/03/googles-ai-wins-fifth-final-game-go-genius-lee-sedol/. [Accessed: 13- Apr- 2016].

[11] S. Gibbs, "Google's AI AlphaGo to take on world No 1 Lee Sedol in live broadcast", The Guardian, 2016. [Online]. Available: https://www.theguardian.com/technology/2016/feb/05/google-ai-alphago-world-no-1-lee-se-dol-live-broadcast/. [Accessed: 13 Apr 2016].

[12] D. Kehoe, "Designing Artificial Intelligence for Games (Part 1) | Intel® Developer Zone" ,Software.intel.com, 2009. [Online]. Available: https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1/. [Accessed: 22- Apr- 2016].

[13] S. Russell and P. Norvig, Artificial intelligence, 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, 1995.

[14] "Video Games and Artificial Intelligence - Microsoft Research", Research.microsoft.com, 2016. [Online]. Available: http://research.microsoft.com/en-us/projects/ijcaiigames/. [Accessed: 15- May- 2016].

[15] Demis Hassabis, CEO, DeepMind Technologies - The Theory of Everything", YouTube, 2016. [Online]. Available: https://www.youtube.com/watch?v=rbsqaJwpu6A/. [Accessed: 15- May- 2016].

[16] M. Olsson, "Behavior Trees for decision-making in Autonomous Driving", M. Sc, KTH Royal Institute of Technology, 2016. [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A907048&dswid=4279/. [Accessed: 15- May- 2016].

[17] A. Champandard, "Top 10 Most Influential AI Games | AiGameDev.com", Aigamedev.com, 2016. [Online]. Available: http://aigamedev.com/open/highlights/top-ai-games/. [Accessed: 15- May- 2016].

[18] "The Elder Scrolls Official Site", Elderscrolls.com, 2016. [Online]. Available: http://www.elderscrolls.com/skyrim. [Accessed: 22- Apr- 2016].

[19] Acadamey of Interactive Arts & Sciences, "Game Developer Details", Interactive.org, 2016. [Online]. Available: http://www.interactive.org/games/game_developer_details.asp?idAward=2012&idGameDeve
[Accessed: 31- May- 2016].

[20] "Best PC Video Games of All Time", Metacritic, 2016. [Online]. Available: http://www.metacritic.com/browse/games/score/metascore/all/pc/filtered?sort=desc.

[Accessed: 01- Jun- 2016].

[21] "Slick2D Wiki", Slick.ninjacave.com, 2016. [Online]. Available: http://slick.ninjacave.com/wiki/index.php?title=Main_Page. [Accessed: 15- May- 2016].

[22] T. Parr, "Enforcing Strict Model-View Separation in Template Engines", University of San Francisco. [Online]. Available: http://www.cs.usfca.edu/p̃arrt/papers/mvc.templates.pdf/. [Accessed: 15- May- 2016].

[23] T. Reenskaug, "The Model-View-Controller (MVC) Its Past and Present", University of Oslo, 2003. [Online]. Available: heim.ifi.uio.no/t̃rygver/2003/javazone-jaoo/MVC_pattern.pdf/. [Accessed: 15- May- 2016].

[24] "Introduction - Tiled", Doc.mapeditor.org. [Online]. Available: http://doc.mapeditor.org/manual/introduction/. [Accessed: 27- Apr- 2016].

[25] G. Booch, J. Rumbaugh and I. Jacobson, The unified modeling language user guide. Upper Saddle River, NJ: Addison-Wesley, 2005.

[26] UML Diagrams - Learn What They Are and How to Make Them", Smartdraw.com, 2016. [Online]. Available: https://www.smartdraw.com/uml-diagram/. [Accessed: 31- May- 2016].

[27] A. Botea, B. Bouzy, M. Buro, C. Bauckhage and D. Nau, "Pathfinding in Games", Dagstuhl Follow-Ups, vol. 6, 2013.

[28] "Introduction to A*", Red Blob Games, 2014. [Online]. Available: http://www.redblobgames.com/pathfinding/a-star/introduction.html/. [Accessed: 15- May- 2016].

[29] J. Brownlee, "Background", Ai-depot.com, 2016. [Online]. Available: http://ai-depot.com/FiniteStateMachines/FSM-Background.html. [Accessed: 15- May- 2016].

[30] A. Champandard, "On Finite State Machines and Reusability | AiGameDev.com", Aigamedev.com, 2016. [Online]. Available: http://aigamedev.com/open/article/fsm-reusable/. [Accessed: 16- May- 2016].

[31] A. Champandard, "The Gist of Hierarchical FSM | AiGameDev.com", Aigamedev.com, 2016. [Online]. Available: http://aigamedev.com/open/article/hfsm-gist/. [Accessed: 16- May- 2016].

[32] K. Murphy, Machine learning. Cambridge, MA: MIT Press, 2012.

[33] T. Hastie, R. Tibshirani and J. Friedman, The elements of statistical learning. New York, NY: Springer, 2016.

[34] A. Maslow, "A theory of human motivation.", Psychological Review, vol. 50, no. 4, pp. 370-396, 1943.

[35] I. Millington and J. Funge, Artificial intelligence for games. Burlington, MA: Morgan Kaufmann/Elsevier, 2009.

[36] "The Elder Scrolls Official Site", Elderscrolls.com, 2016. [Online]. Available: http://www.elderscrolls.com/morrowind. [Accessed: 25- Apr- 2016].

[37] P. Sweetser and J. Wiles, "Current AI in Games: A Review", Australian Journal of Intelligent Information Processing Systems, vol. 8, no. 1, pp. 24-42, 2002.

[38] S. Rabin, AI game programming wisdom 2. Hingham, Mass.: Charles River Media, 2006.

[39] S. Rabin, AI game programming wisdom. Hingham, Mass.: Charles River Media, 2002.

[40] M. Jones, Artificial intelligence. Sudbury, Mass.: Jones and Bartlett Publishers, 2009.

Appendices

A

Images from the Game

Distributed under Creative Commons 3.0 (CC-BY 3.0).
Created by Zabin, Hyptosis, and Danial Cook.
Available: http://opengameart.org/content/castle-tiles-for-rpgs

Distributed under Creative Commons 1.0 (CC0).
Created by Buch.

Distributed under Creative Commons 3.0 (CC-BY-SA 3.0).
Created by Casper Nilsson, Johann Charlot, Daniel Eddeland, Stephen Challener, Charles Sanchez, Manuel Riecke, Daniel Armstrong, Lanea Zimmerman and Skyler Robert Colladay.

Distributed under Creative Commons 3.0 (CC-BY-SA 3.0).
Created by Casper Nilsson, Johann Charlot, Barbara Rivera, Chris Phillips and Lanea Zimmerman.

B

Software Description

Background

We will create a game where the focus lies on creating characters with believable interaction and behavior and seeing how they affect the world they inhabit. The game will be seen from a top-down 2D-view and will be developed for PC. The world will be set in a medieval fantasy world and will be generated to contain resources like trees, water, stone and gold. When a world has been generated we will place one, or several, villages in the world, each with their own set of residents who will have to survive in the world they have been placed. The player will be able to either stay in the god view or enter the world as a resident, all other residents will be controlled by the AIs. The AI will decide what the character it controls will do depending on its current needs. The needs of a character will also depend on the characters traits and skills. An example would be that a character is hungry, then it will need to either buy food to eat, or, if it does not have enough money, gather the food itself. The game will also implement a relation system allowing characters to have a family, friends or enemies.

Characters

The characters will represent the villagers in the game world. They can be seen as a sprite (a 2D-animation) and this animation will show the player what the characters are currently doing. The character is without intelligence, so without a AI it would never act. In the game's most simple form, the amount of characters in the world will decide how fast the population will increase, that is if we don't implement a more complex reproduction system around the relationships (see Additional goals.).

The game will show the player the needs, traits and skills of characters, this will show the player why a character is performing a certain task. The needs of a character will represent what it requires to be healthy, and staying alive, such as eating, sleeping and socializing. Skills will decide how good a character is at a task, such as fishing or farming, and in this case being better would give more food in less time. Traits will affect the needs, for example a character might be gluttonous and therefore requires more food when hungry. These different stats will make up the personality of the character.

Every character will also have a inventory where it can store items such as resources and money. This will enable different characters to trade with one another. To perform a trade, the characters must be close to each other and both must agree to the proposed trade.

The characters should have a age as a way for the player to see how long they have existed in the world. Characters should also be able to be born, and die of old age.

AI and NPC

The AI will be the ‘brain’ of our character, without it they wouldn’t be able to survive, let alone act. Every character will be controlled by a AI and the AI will tell a character how to act depending on its current needs and wishes, this union is our NPC (Non player character). As the traits and skills of the character affect its needs and wishes they will also affect the AI’s decision of what the character should do. The AI will never be seen in the game, it’s purely code running in the background that makes the NPCs act.

The focus of the project is making this AI capable of making the characters act in a believable and rational way. This means that the characters must make decisions based on their environments and decide the best action to take based on this. The game will feature a number of different AI and thus the NPCs will perform differently in some situations. The NPCs will react on the actions of other NPC, and also affect them in return, and interact not make any difference between the player’s character and other NPCs. How this interaction precedes and what it entails will depend on the traits of the characters.

World

The world will be a top down 2D-view where all the things in the game plays out. The world will be relatively large and therefore the player will be able to zoom in and out on the map to get different levels of detail. The zoom will be implemented by having 3 different views the player can switch between. In the first, the player will be able to see the whole world, but smaller things like NPC’s and houses will be harder to see. The second view will be zoomed out enough to show an entire village, where the player can see what’s going on in the entire village and how it’s developing. The third will be the one with highest detail but shows the least of the map. This view will show the player’s character as well as other NPC’s.

The map will also contain mountains, forests and other places where NPC’s can gather resources like farms and mines. Regardless of where the player is the entire world will be updated equally.

The world will also feature a system for weather and seasons which affects what resources the NPCs, and player, can find. There will also be a chance for natural disasters occurring that would negatively affect the characters in some way or other. There will also be some kind of animal life in the world, that could potentially work as a resource as well as a threat to the characters.

Structures

The most basic structures, houses, will work as a capacity limit for how many villagers there can live in a town, there can't live more people than what the housing allows. If all the houses are occupied more houses must be built in the town for the population to be able to keep increasing. There will be additional structures for resources, like mills, lumberyard and mining camps.

Resources

Characters will be able to extract resources from nature on the sites in the world where the resource is available. The resources will exist in three categories:

- Unlimited resources – never runs out no matter how much the villagers extract from the source.
- Renewable resources – resource will spread while there is still some source that can spread it.
- Limited resources – will only exist in a fixed amount and when it runs out the world will have to manage without it.

Following are some short description of the resources in the game:

- Wood – trees will exist as a renewable resource since trees will grow up next to other trees so the forest will expand. NPC's can chop down trees to get wood to sell or work with, hence its categorized as a resource, but if all the trees in the world are cut down then the resource will not come back.
- Water – an infinite resource which will exists in different forms, such as oceans and rivers. NPC's can gather water in these locations or build wells which will be required to sate their thirst. NPC's can also catch fish to gather food and use water to improve the agriculture.
- Stone – stone will exist and can be gathered.
- Gold – gold will exist and be able to be extracted from mines. It can be used in trading with other NPC's.
- Food – food will be created in farms or caught from fishing and hunting. It will be used to keep NPC's from starving and can also be traded.

Player interaction

The player will be able to either actively interact with the world and the NPC's by taking control of a character in the same way that the AI's do. Or simply view the world and the NPC's without being able to influence them or having to worry about the needs of a character. In either mode of playing the player will be able to view the world in any of three different views that are listed under the world section.

Additional goals

We hope to implement relations between NPC's so they can make friends, family and enemies. The relations will depend on characters traits and somewhat on the skills as well. This is not a requirement since humans can make connections with people who have different traits and interests in the real world. If we do implement this, then the population in the village will depend on how many families living in the village, also housing will be restricted to family members and so the capacity of houses will not be maximized, a stranger can't move in just because there's an extra bed. Another goal is implement conflicts between NPC's, they might fight over resources or because two family has some kind of rivalry or simply personal issues. Interactions like these can result in death. We are also thinking about spawning more than one village in the world to see how residents in different villages interact with and behave to one another.

We also discussed to implement "fog of war", this will cover the map in darkness if no NPC is currently in that area. If this was the case, NPC's wouldn't be all knowing and therefore would have to explore to find resources and check areas where they had been before to know if resources were still there, someone could have taken it while the NPC was away. This would make it possible for NPC's to share information with each other to know where resources were most recently collected.

A last goal is to make a tool for the player to create a game world or generate a random one. Then the player could test scenarios they were interested in or simply fast get into a new unique world.

Functional Requirements

The following sections will state the planned functionality of the game in the form of functional requirements.

Characters

1. Characters should have needs.
2. The magnitude of the needs should increase over time.
3. Characters should have skills.
4. The skills should increase when used.
5. Characters should have traits.
6. The traits should affect the needs.
7. The traits should be constant and do not change over time.
8. Characters should have an inventory.
9. Characters should be born.
10. Characters should be able to die.
11. Characters should eventually die of old age.

Rationale Characters requires needs and traits so the AI can decide what the character must do. The needs must decrease over time to continuously present a challenge for the one controlling the character. The traits must somehow affect the needs in order to have a direct influence on the character. They also need skills to determine how good they are at different tasks which will affect the best way to fulfill their needs. Inventory will be needed to keep track of what resources a character has and thus required for trading. Character must be born and be able to die of age or other reasons so that the population, and state of the world, will change over time.

AI and NPC

12. All AI should be able to control one character each.
13. The AI should make decisions based on the current needs of the character.
14. The AI should make decisions based on the current goals of the character.
15. The effect on the needs and wishes of the character, that the traits and skills have, should affect the AI's decisions.
16. The AI should make decisions based on the environment around the character it controls.
17. The AI should take the actions of other NPCs into account when deciding on actions for the NPC.

Rationale If the AI cannot control a character it has no purpose there will be no NPCs in the game. For the AI to be 'believable' it must make decisions based on the needs, wishes, traits and skills of the controlled character. It must also take into account the environment around the character and the actions of others to appear 'believable'.

World

18. The player should be able to zoom in and out between three different levels.
19. The world should contain different resources.
20. The world should contain characters.
21. The world should contain structures.
22. The world should have a basic weather system.
23. The world should contain animal life.
24. The entire world should be constantly evolve and change regardless of where the player is.

Rationale The world needs to be possible to view, and as the world should be large different levels of detail are necessary. The world needs characters for the game to be possible to play. The world must also contain resources so that the characters can fulfill their needs and survive, less they all die out immediately. The world should contain structures as they set the max capacity for the villages population. A basic weather system will be implemented to make the lives of the villagers harder by randomly spawning harsh weather and natural disasters, this is to take some of the power of control from the AIs and make them need to adapt to change. There should also be some kind of animal life in the world, again to take some power from the AIs in that they may unexpectedly encounter wild animals or gather a resource that can move (i.e. hunting). As all characters should have equal importance in the world, it must progress even when the player is not present.

Structures

25. Characters should be able to construct structures.
26. There should be different kinds of structures filling different functions.
27. There should be structures that act as houses.
28. The number of houses in the village should set the max population of the village.
29. There should be structures that can act as stockpiles for resources.

Rationale For a village to be more than just a collection of characters it needs to have some kind of structures. For the village to behave in a believable way its maximum population must depend on the size of the village. The possibility to build structures gives the NPCs a possibility to make decisions that have an immediate cost but positive effects in in the long run.

Resources

30. There should be infinite resources in the world.
31. There should be renewable resources in the world.

32. There should be finite resources in the world.
33. Resources should be possible to be collected from set positions in the world.
34. The amount of a finite resource at a certain location should decrease if collected.
35. The amount of a finite resource at a certain location should be able to be depleted.
36. The amount of a renewable resource at a certain location should decrease if collected.
37. The amount of a renewable resource at a certain location should increase over time.
38. The amount of a renewable resource at a certain location should be able to be depleted.
39. If a renewable resource has been depleted at a certain location, then its amount at that location should no longer increase.

Rationale For the characters to be able to survive they must be able to collect resources to satisfy their needs. For the collection of resources to present a challenge for the NPCs, or the player, some of them must be able to be depleted, and some limited so that they NPCs must plan ahead to be successful.

Player interaction

40. The player should be able to control a character .
41. The NPC's should interact with the player controlled character in the same ways as with each other.
42. The player should be able to view the world without controlling a character.
43. The player should be able to view the world from the three different views stated above.

Rationale For the player to be able to test how the NPCs react to different actions taken by characters it is necessary for the player to be able to control a character, and for the NPCs to interact with this character in the same way as with each other. As controlling a character and taking care of its needs can distract the player from the development of the world, it is necessary for the player to be able to view the world even without controlling a character. As there are several different things one might want to observe at different times, the world needs to be able to be viewed from different views.

Non-Functional Requirements

The NPCs should appear believable as humans. The decisions made by the AIs should always be relevant to the needs, traits, goals and skills of the controlled character. The game should present information to the player in a clear way. The calculations made by the AIs should be efficient enough that the addition of NPCs should not drastically affect the running speed of the game.

C

States of Early AI Implementation

A list of the states that were in the early implementation of the AI.

- Idle - This state was used to plan the AIs next move, as well as popping the next state from the state stack making it the current state that would then be executed in update() of ArtificialBrain. The AI will enter idle state in between states. It was also used to check if the character was hungry, thirsty or tired. This was later moved to update() of ArtificalBrain for performance reasons.

- Hungry - The AI would enter this state if hunger fell below a certain threshold. The AI would first check if the character had food in its inventory. If it did it would consume it. If it didn't, the AI would push food onto the gather stack and look for that.

- Thirsty - The AI would enter this state if thirst fell below a certain threshold. The AI would first check if the character had water in its inventory. If it did it would consume it. If it didn't, the AI would push water onto the gather stack and look for that.

- LowEnergy - The AI would enter this state when energy fell below a certain threshold. If the AI had a house it would go home to sleep, if it didn't it would rest where it stood.

- Gather - This state is used to gather resources. The AI would check what the first resource on the gather stack was, it would then go look for that resource. If there was no resource on the gather stack, the AI would randomize a resource to look for.

  - Material - Specific state that would be entered once the AI reached a certain resource. This state was used to gather material and put it in the inventory. This was later split up into specific gathering states for each resource.

  - Food - Same functionality as Material, but for food.

  - Water - Same functionality as Material, but for water.

- Eat - In this state the AI will consume a food item from its inventory to replenish hunger.
- Drink - In this state the AI will consume water from its inventory to replenish thirst.
- Sleep - In this state the AI will sleep to replenish energy. This state could only be executed when inside a house.
- Resting - In this state the AI will rest to replenish energy. This was not as effective as the sleep state, but could be performed anywhere. It was used as a means to replenish energy before the AI had built a house.

D

States of Final AI Implementation

A complete list of all the states in the final AI implementation of this project.

- Idle - The AI enters idle state between states. It is used to pop the next state from the stack. The idle state contains the functionality to decide what the AI should do next. This is hardcoded, to make sure the AI does things in an appropriate order. It will start by building a house. If it has a house it will proceed to build a farm or a stockpile. If it has built a stockpile and a farm it will either hunt, work on a farm or gather the resource of which it has the lowest amount in its inventory.
- Build - In this state the AI will check what resources it needs in order to build a specific building. It will then push those resources onto the gather stack and find resource stack (if those resources are not present in the AI's memory).
- Drink - In this state the AI will consume water from its inventory to replenish thirst.
- Eat - In this state the AI will consume a food item from its inventory to replenish hunger.
- Sleep - In this state the AI will sleep to replenish energy. This state can only be entered when in a house.
- FindResource - In this state the AI will walk in a random direction until it finds what it is looking for. This state is used to find resources it does not currently have in memory.
- Follow - This state is used to follow moving objects around the world; such as animals. It is used to catch up to an animal the AI is hunting. It was earlier used to catch up to characters as well, but this functionality was removed when it was decided not to have social interaction in the game.
- Gather - This state is used to decide what it is to gather if there is nothing on the gather stack. It is used for transitioning to the specific gather states for each resource. It is used to determine which resource is the closest, if there is one. If there isn't, it is used to transition into Gather<Resource>.
- Gather<Resource> - Each resource has its own specific gather state. These states are used to actually gather the resource from the node and put it in its inventory, once the AI has reached the resource in question. For instance if it is to gather wood and has reached a tree, the AI would enter the GatherWood state.

- Hungry - The AI will enter this state if hunger falls below a certain threshold. The AI will first check if the character had food in its inventory. If it does it will consume it. If it doesn't, the AI will push food onto the gather stack and look for it.
- Thirsty - Same functionality as hunger, but instead of food it will consume or look for water.
- LowEnergy - The AI will enter this state when energy falls below a certain threshold. If the AI has a house it will go home to sleep, if it doesn't it will rest where it stands. Resting - In this state the AI will rest to replenish energy. Resting is not as effective as sleeping, but can be done anywhere.
- Moving - In this state the AI finds and follows a path to a destination in the world. The destination is set by a previous state which then transitions into the Moving state to move to that destination.
- Hunting - In this state the AI will hunt for animals. This state contains its own movement and works similar to FindResource but for animals. If an animal is spotted the AI goes into the Follow state.
- WorkFarm - In this state the AI will work on a farm. When an AI works on a farm, crops will grow that can be consumed to replenish hunger.