Southampton Solent University

SCHOOL OF MEDIA, ARTS AND TECHNOLOGY

BSc (Hons) Computer Games (Software Development)

2017

Yannick Pohl

"Real-Time Strategy Artificial Intelligence"

Supervisor : Mark Bennett
Date of presentation : February 2018

1 Abstract

The purpose of this project was to create a real-time strategy Artificial Intelligence (AI) which should function as an opponent, destroying all enemy units in the game. The first approach to the AI did not include a game but should use a set of test data, this lead to major structure changes since this project was not supposed to be a science paper about an AI algorithm.

This report describes what has been accomplished during the development and where the planning failed. It also focuses on the future development of this project and shows a possible way of an algorithm that could have been implemented if other important aspects were present.

Content

2 Progress Report

2.1 Background

2.1.1 Project

The goal of this project is to create an Artificial Intelligence (AI) to work as an opponent in a real-time strategy (RTS) game.

The AI is supposed to be used on a specific RTS game. This game is under development and is not usable during the project time. The AI will not be compatible with an already existing RTS game like Empire Earth (*Stainless Steel Studios 2001*). Input for the Ai will be handled separately and is rudimentary but optimized to the actual game.

To understand the concept of the game, it was inspired by Empire Earth: The Art of Conquest. The game will be following the basics of Empire Earth but to due to different features the AI cannot be used as a modification for it.

2.1.2 Overall Aims

The overall aim of the project is to create an Artificial Intelligence (AI) that will be used in a real-time strategy game.

The basic aim is to finish each control-layer of the AI at a time to insure its functionality and to provide at least a work ground for further development. The next step would be to connect the layers to a single working algorithm. For example, connecting strategic planning with military management and then with unit control.

An additional aim is to equip the AI with machine learning algorithms to bring up diversity over multiple games.

In the end, the AI should be able to generate strategies, manage resources and command units by using layers of abstraction (*see 2.3.1 Abstraction Layer for an in-depth description of abstract layering*).

2.2 Objectives

2.2.1 Main Goals

- Define suitable layer of abstraction[1] for the Artificial Intelligence to operate on.
- Research different algorithms that suits the problem of the given layer.
- Include the algorithm that solves the problem of the given layer with the desired result and with the least performance cost.
- Create test data for the algorithm to evaluate if the result is desired or no usable.
- Connect the layers to provide input data from the layer above.

2.2.2 Stretch Goals

- Create functionality which can alter test data to simulate a real-time strategy game.
  This should give a demonstration of the working Artificial Intelligence.
- Implement complex machine-learning algorithms instead of regular algorithms.

- Create test data for the algorithm to evaluate if the result is desired or no usable.

- Implement multithreading into the algorithms to relieve the main thread.

1, see 2.3.1 Abstraction Layer for an in-depth description of abstract layering.

2.3 Design

2.3.1 Abstraction Layer

To maintain a clean structure of the Artificial Intelligence (AI), the design will use abstraction.

To enhance the performance of certain behaviour, the design will use layers (level).

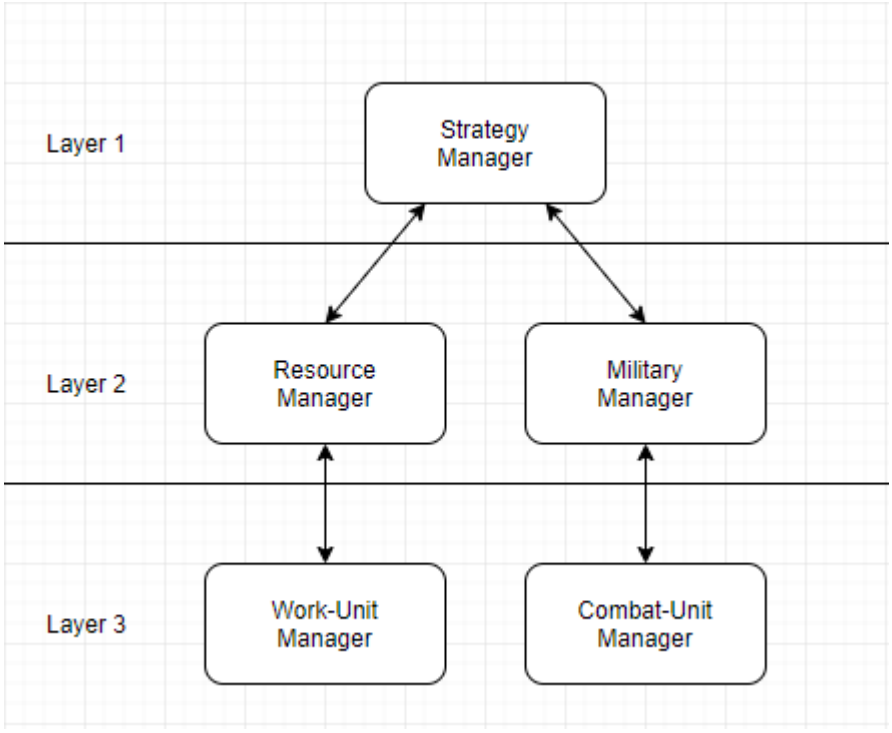

Figure 1. Abstraction Layer Diagram (F. Safadi, R. Fonteneau and D. Ernst n.d., p. 4)

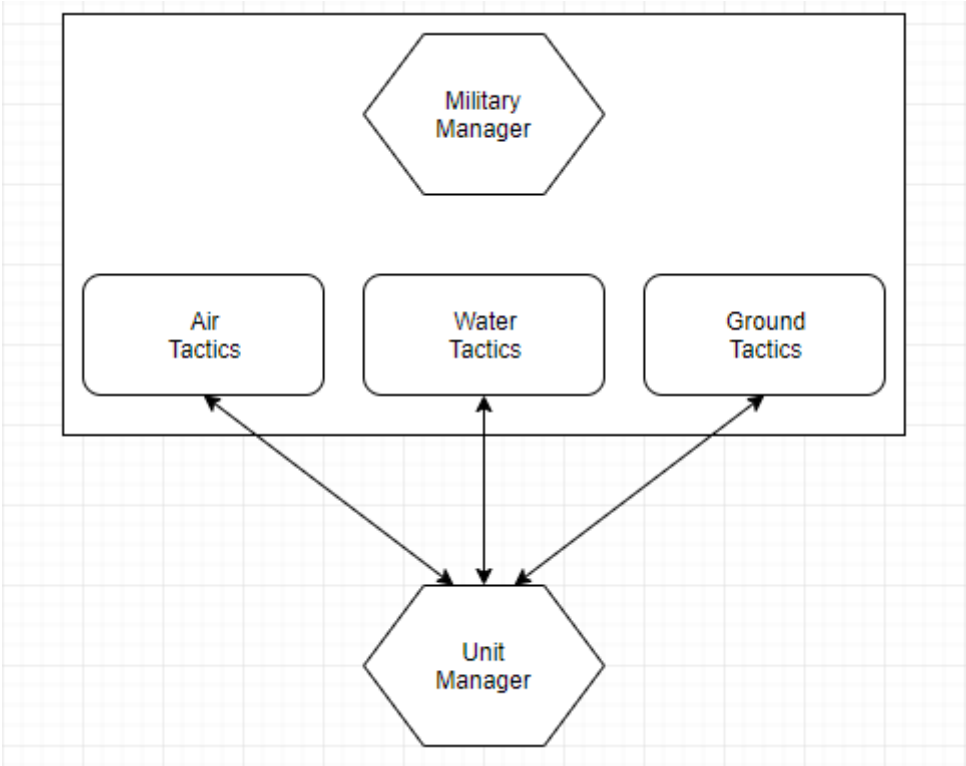

Figure 2. Abstraction Layer Diagram

2.3.1.1 Theory:

The more processes we separate and chunk down into smaller tasks, the better each algorithm can calculate a desired solution to its problem. This contributes to a clear structure and superior behaviour. This can be associated with the Divide and Conquer paradigm (*Anon. 2018*).

2.3.1.2 Practice:

This method has several issues that can occur. It can get messy if it is not supported with appropriate diagrams. It is harder to debug due to the amount of information that is calculated and transferred between each layer. The AI

may be better but that does not mean that it contributes to a good experience for the player.

, see Appendix A for an in-depth definition of abstract layering.

2.3.2 Structure

Often, but not always, real-time strategy (RTS) games have their focus on 2 aspects, economy and military. Economy is the simplest out of all important and less important aspects, because strictly speaking, placing buildings of any aspect would be falling under military management. An additional aspect of RTS games could be diplomacy.

As (*Figure 1. Abstraction Layer Diagram, p. 6*) shows, these two aspects are separated into manager. Their only functionality is to ensure that sub manager do their work, like a hierarchy in a company. They get information from the lower level, communicate with each other and provide information to the upper level.

A manager can be split into additional manager like (*Figure 2. Abstraction Layer Diagram, p. 6*) shows. If seen in the context of Figure 1. Abstraction Layer Diagram, the Strategy Manager could provide a game strategy and the Military Manager a more specific strategy for the next combat, while the sub manager training units and position them properly.

2.3.3 Timing

The design of this algorithm means that, since it is not rudimentary anymore, timing of each iteration is a very important aspect of this Artificial Intelligence. The larger the algorithm, the more time is needed for each frame to calculate the amount of information.

Which means, that reducing the iterations at higher levels more than at the lower levels result in more processing time, since higher level algorithms are more likely to work with resource intense pathfinding calculations; Machine-learning algorithms would be an example. While the whole game strategy can be calculated once, unit position updates must be processed almost at each frame.

Another way to control processing time for each algorithm is the Big O Notation (*Rob Bell 2009*).

2.3.4 Multithreading

Multithreading could become a problem at a later stage of the development. Since the development will be focused on the Artificial Intelligence algorithm only and not unit pathfinding, it should not be a major problem. It is, in regards of the project, only a stretch goal and is not relevant as an engine criterion. Whereas in the actual game, the option of multithreading must be considered because the performance drawback will be noticeable.

2.3.5 Algorithms

As the abstraction layer design of the Artificial Intelligence (AI) implies, higher level algorithms can be used for strategy planning. This makes it very appealing

to machine-learning algorithms. Lower level algorithms have a more specified or static field of problems to solve. Beneath are some interesting algorithms that might be useful for the project.

2.3.5.1 GOAP (variation)

Goal Orientated Action Planning (GOAP) solves a specific problem with given possible actions to do so. Each action has requirements and costs. The best solution would be to solve the problem with the least costs (*Mark Bennett 2017, Level 6*).

A possible variation of the GOAP, to plan a game strategy, would implement parts of the Monte Carlo Tree Search. For example, first it would choose actions at random and store a win/play ratio together with some other information, like map type, number of enemy player and more. In later games it would learn what strategy was successful and can still vary some actions due to randomness.

2.3.5.2 Monte Carlo Tree Search

The Monte Carlo Tree Search algorithm is used to generate legal board states after a player altered the current state, like in a game of chess or go. It chooses board states with the best win/played ratio, if applicable, to reach the best conclusion (*Mark Bennett 2017, Level 6*).

This algorithm could be used to analyse influence maps of the real-time strategy (RTS) game. As RTS games presume, the problem would be that the influence map is being updated several times per second and by both players at the same time. That and the huge amount of possible states would require

unobtainable processing power and time. In conclusion, this algorithm is not suited for this project.

2.3.5.3 GOB

Goal Orientated Behaviour (GOB) can be sometimes mixed up with Goal Orientated Action Planning (GOAP) because it seems they achieve their goals on the same way. The GOAP is, as the name says, a planning algorithm, whereas the GOB is not. The best example for a GOB algorithm is The Sims 4 (*Maxis 2014*); When a character has the need (goal) to clean, the GOB tries to fulfil this goal by using the shower (*Mark Bennett 2017, Level 6*). It does not plan.

The GOB algorithm can be used to send commands to the lower levels of the abstraction layer design. For example, the AI needs an army to stop an enemy attack. To satisfy this goal, it would build artillery and soldiers. If this is not possible because the attack is too early for the artillery to be finished in production, it would choose to build more soldiers instead. This would not satisfy its goal as much, but it would still be the better move.

2.3.5.4 Resource Tree

The Resource Tree is a variation of the Decision Tree and essentially it is not making its choice based of Booleans, instead it makes choices based on the difference of a real number and the goal percentage of a node (*Niklas Hansson 2010*). The Resource Tree tries to balance the real numbers to the given percent of the node, that's why the design is also called Auto Balancing Decision Tree.

Due to its design, the Resource Tree could work as the whole AI algorithm. The algorithm has the downside that, because of its basic design, it gets slower as the tree grows, because it would work down the same path, if the resource distribution has not changed. The best way would be, to keep the tree smaller and manage just a part of the resource distribution.

2.4 Project Management

2.4.1 Methodology

2.4.1.1 Predictive Waterfall

The main project management methodology will be the waterfall model. It is the prescriptive framework that must be used for the project (*David Cobb 2018, Level 6*). For a more detailed look about Waterfall, see Appendix E. Each phase requires a report of the current progress: Definition Report, Progress Report, Prototype and Final Report. Between each phase it is up to the developer what methodology is an appropriate choice for the development.

2.4.1.2 Adaptive Feature-Driven Development

Feature-Driven Development (FDD) will be used for the development cycle of the project. This adaptive methodology is a good approach because of the high uncertainty of time and probably also of scope. The five milestones provided by the FDD process are a good approach for the Artificial Intelligence (AI) requirements. Three of the milestones are especially interesting: Develop the overall model, an example of this can be seen in 2.3 Design; Build feature list, a good way to show the AI behaviours; Build by feature, to implement the features or behaviours. For a more detailed look about Feature-Driven Development, see Appendix E

2.4.2 Tools

2.4.2.1 Trello

Trello allows to manage tasks that are used for the Feature-Driven Development (FDD) process. FDD is described with more detail in 2.4.1

Methodology and Appendix E. Furthermore, Trello offers an automatic or manually created Gantt Chart and Burndown Chart.

2.4.3 Time Estimates

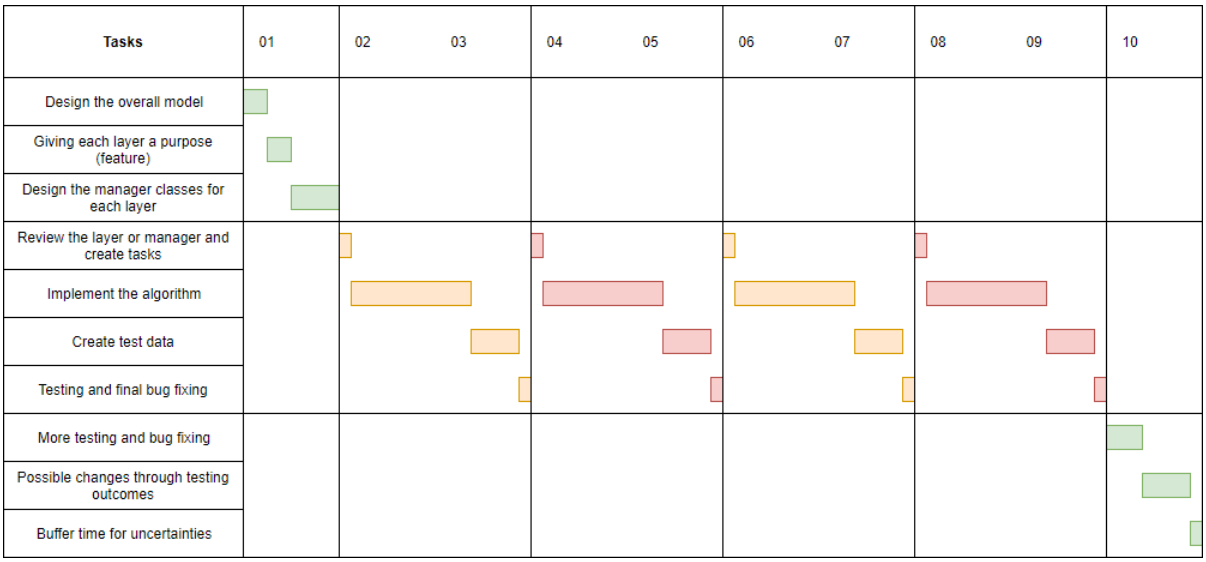

Figure 3. Gannt Chart

Below are some tables with a more detailed description of the tasks shown in the Gantt Chart above and with the estimated time for each task and phase.

Week 01			Appx. 40h
	Task	**Description**	**Time est.**
01	Design the overall model.	- Design the Artificial Intelligence structure (the abstraction of each layer), and make sure it is not too large for 10 weeks of work (appx. 400h).	10h
02	Giving each layer a purpose (feature).	- Ensure that there are no layers that do not fulfil a purpose.	10h
03	Design the manager classes for each layer.	- Creating a structure within a layer. Defining the input and output of the manager, how the manager is communicating with other layers or managers.	20h

Week 02-09	[Two Weeks per Iteration]	Appx. 80h
Task	**Description**	**Time est.**
01 Review the layer or manager and create tasks.	- Make sure that pervious work has no conflicts and plan the implementation of the algorithm.	4h
02 Implement the algorithm.	- Implement the algorithm. The manager should calculate the appropriate outputs based on the inputs given.	50h
03 Create test data.	- Create a test structure for the Artificial Intelligence to work with.	20h
04 Testing and final bug fixing.	- Test the algorithm on the testing structure to ensure that the manager contributes in the desired way to the overall product.	6h

Week 10			Appx. 40h
	Task	**Description**	**Time est.**
01	More testing and bug fixing.	- Ensure that everything works as intended by a larger testing session.	16h
02	Possible changes through testing outcomes.	- It might happen that the testing session revealed some issues that must be corrected.	20h
03	Buffer time for uncertainties.	- Maybe something went not right and needs more time. Can also be used to flesh out the final document report.	4h

2.4.4 Risk Analysis

	Risk	Chance	Weight	Description
01	Too much allocated Work per Sprint	High	Medium	Having too much work assigned to a sprint is a result of bad preparation, but it is almost impossible to negate.

Symptoms	Prevention Difficult
- Calculated work time is larger or equals the calculated sprint time.	- Calculate an estimation time for each task. - Assign no more tasks to a sprint if there is not enough buffer time left for that sprint.

	Risk	Chance	Weight	Description
02	Missing Sprint Deadlines	High	Medium	Missing a sprint deadline begins with having too much work planed for a period. It might happen that this is due to other risks that occurred during the sprint.

Symptoms	Prevention Normal
- Calculated work time is larger or equals the calculated sprint time.	- Make sure that the work, that has been assigned to the sprint, has its own time calculation.
- Realising that there is too much work left for the remaining sprint time.	- Always assign less work than possible to the sprint, to make sure that enough buffer time is left.

	Risk	Chance	Weight	Description
03	External Distraction	High	Low	External distraction will always persist, it can only be reduced to a minimum. Sometimes, external distraction is necessary to be able to concentrate longer on a task, like mini breaks. Balance is the key.

Symptoms	Prevention Simple
- Realising that your concentration is on something else.	- In case your attention is not on the task anymore, try to interrupt perceptible contact.
- Realising that you are not working anymore.	- If you find it hard to concentrate on the task, take a short break. Make sure to continue working.

	Risk	Chance	Weight	Description
04	Insufficient Testing	Medium	High	Artificial Intelligence (AI) is an area where testing is a very important but also very time-consuming task, due to randomness and the number of possible outcomes.

Symptoms	Prevention Normal
- The AI is behaving strange or if a testing plan exist, not like the desired result.	- Create a test plan for the AI with the feature, a desired result and the actual result. - Each part of the AI and each connected parts of the AI might behave different, so the test plan must be reiterated or extended.

	Risk	Chance	Weight	Description
05	Get Stuck at Problems	Medium	Medium	Getting stuck at tasks or problems is a common thing. Sometimes the problem or task is fundamental for further development. In that case some prevention advises may not work.

Symptoms	Prevention Simple
- Rewriting the same code repeatedly. - Having a blackout or not being happy with the current implementation.	- Start or continue other tasks to clear your mind. You may come up with a desired solution. - Make a short break; Grab something to eat or drink.

	Risk	Chance	Weight	Description
06	Feature Creep	Medium	Low	Feature creep is the risk of subconscious forcing yourself to implement an unnecessary feature. It is simple to prevent but the symptoms may be hard to notice.

Symptoms	Prevention Simple
- There are no clear defined symptoms. If you notice yourself working on something unnecessary it might be too late.	- Having a clear defined feature list and a task list with an additional description of each task. - Discard not finished or unnecessary code and mark the feature or task as completed. - Ask yourself if the feature or task is necessary at this stage. Maybe it is worth making it a stretch goal.

	Risk	Chance	Weight	Description
07	Missing Deadlines	Low	High	Missing a deadline often is a result from illness or system failures. It might happen that this is due to other risks that occurred during the sprint.

Symptoms	Prevention Normal
- Systems or services may indicate a maintenance downtime during a deadline date. - Getting illness shortly before a deadline submission.	- Avoid the risk of illness during the project time. - Avoid the risk of system or service failures during the project time or maintenance downtimes.

	Risk	Chance	Weight	Description
08	Losing Work	Low	High	Losing all progress of the project would be, if it would happen at a later stage, the end of the universe. The best option would be to reallocate to a backup plan.

Symptoms	Prevention Simple
- Realising that all work is gone.	- Store up-to-date backups on other HDDs or online services. - For partially loss of work, use version systems or services to cover the risk.

	Risk	Chance	Weight	Description
09	Getting ill	Low	Medium	Depending on how bad the illness is, it might be possible, that the chances of other risks increase. Commonly, illness has the habit to decrease the efficiency.

Symptoms	Prevention Difficult
- There is a wide range of symptoms depending on the disease.	- Drink enough water and eat healthy. - Stay away from persons that are already ill.

	Risk	Chance	Weight	Description
10	Hardware Failure	Low	Medium	It might happen that a hardware component break. Depending on the hardware, this risk is can be a serious problem. In very few cases this can be foreseen.

Symptoms	Prevention Simple
- There are a wide range of possible symptoms depending on the hardware, but in most cases, there are no symptoms.	- In case of memory loss, have up-to-date backups on other HDDs or online services. - Buy trustworthy hardware.

	Risk	Chance	Weight	Description
11	Working too much	Low	Low	Overworking can lead to bad results and should be avoided.

Symptoms	Prevention Simple
- Realising that your concentration is not stable anymore or on something else.	- Take breaks in between work phases. - Grab something to eat or drink.

2.5 Project Specification

2.5.1 Project Approach

2.5.1.1 Methodology

The project management will be done with the two frameworks, Waterfall and Feature-Driven Development (FDD), that are described in 2.4.1 Methodology and Appendix E. The section also describes why the choice narrowed down on these two methodologies and how they be used during the development process. In addition to FDD, Trello is used to manage tasks during the process phases.

2.5.1.2 Design

There is a detailed specification about the design for the Artificial Intelligence in 2.3 Design. It describes the abstraction layer and the possible advantages and disadvantages that comes along with some algorithms. The section also discusses the aspect of multithreading and timing, to maintain performance during the game.

2.5.1.3 Development

The section 2.4.3 Time Estimates gives an overview about the distribution of the tasks during the development and shows the estimated time that is needed to complete a task.

2.5.2 Software

2.5.2.1 Game Engine Unity

The project will be realised in the Unity game engine because the Artificial Intelligence (AI) will be used in a game that is written in Unity, it allows an easy implementation of GUIs and data for testing and previous knowledge about Unity is existing. For this reason, Unity was chosen against Unreal; For a more detailed look about Unity and Unreal, see Appendix B.

2.5.2.2 Editor Visual Studio

Visual Studio will be used to write the AI code for the project. It stands in no comparison with other Integrated Development Environments (IDE) because it offers the most advanced features that are currently available for IDEs; For a more detailed look about Visual Studio, see Appendix B.

2.5.2.3 Language C#

Since Unity is the editor used for the project, only JavaScript or C# are available as programming language. This choice is based more on preference, since more previous knowledge with C# that with JavaScript exist. Also, C# is more suited for the development due to its structure; For a more detailed look about JavaScript and C#, see Appendix C.

2.5.2.4 Version Control Git, GitHub & SourceTree

The version control for this project will be handled over Git. The service will be GitHub because it offers useful project management and documentation features besides the hosting of the project versions. Instead of the Git console, SourceTree will be used because it carries out the same functionality as the

console, but it is a lot quicker due to its graphical user interface. For a more detailed look about Git and SourceTree, see Appendix D and Appendix B

2.5.3 Alternatives

2.3 Design propose an approach to an Artificial Intelligence (AI) design that should handle the scale of a real-time strategy game. However, it may occur that the initial design is not suited to deal with the problem, is too large or difficulties during the development occur. In that case, having alternatives to fall back on is an important step to prevent failure.

2.5.3.1 Reducing Complexity

If the initial design is too large to be completed during the project time, it might be appropriate to take some features out of the AI. Reducing the amount of complexity is the first step to get back in the planned time structure.

2.5.3.2 Without Design

If the whole design should fail, the chances are high that it will be noticed too late. At a later stage during the development progress, the only option might be, to use a single algorithm or at least two algorithms.

3 Final Report

The Final Report is the documentation of the Final Major Project - Real-Time Strategy Game Artificial Intelligence and sets a focus on the development phase of the project. It describes the difficulties and achievements that were made.

The final product is a Unity Project with the Unity version 2017.3.0f3 and stored in /Archrival Empire. Supporting material is stored in /Documents and partly included in this report.

3.1 Implementation

3.1.1 Week 01 Plan Update

The first problem that had to be addressed was the AIs input and output system, described in 2.1.1 Project and in 2.2.1 Main Goals as “Create test data for the algorithm to evaluate if the result is desired or not usable.”.

While defining suitable layer for the AI and looking for algorithms that might produce the desired result, the problem was not to generate input or output for and of the AI, but more how the output-information is processed into logical input without breaking the constraints of the game-logic and therefore resulting in uncertain AI behaviour.

The game was the missing piece. To make the AI behave as supposed, the rudimentary input system had to be converted into an actual working game, a system that translates the AI output into accurate input.

Since the initial plan was to make an advanced Artificial Intelligence that does more than just running down a simple decision tree, changes were necessary. The game and the AI had to be cut down in functionality because both were too complex for the whole remaining project time.

A new time plan was created featuring the creation of a small game, covering half of the project time, as seen in the figures below. The first figure shows an overview of the whole project time while the second figure shows more precise allocated tasks for the first task of the project “Create the game”.

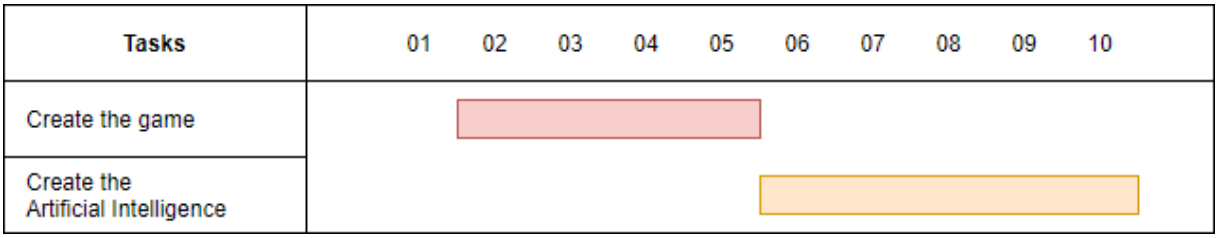

Figure 4. New Gantt Chart

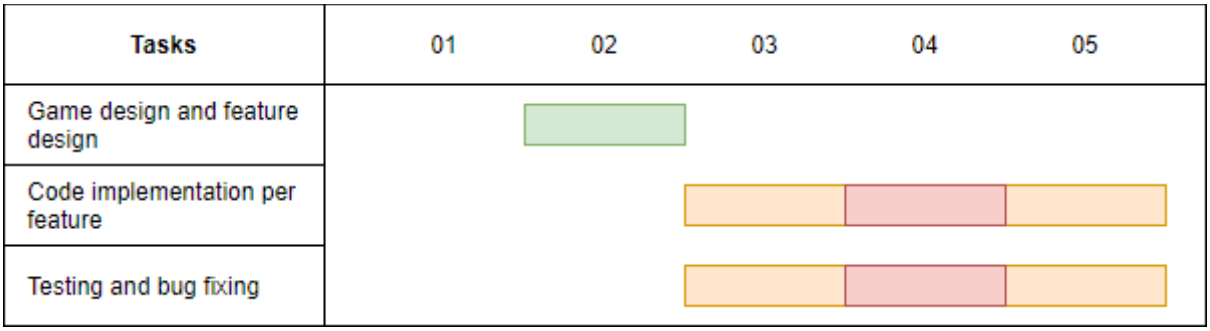

Figure 5. New Gantt Chart

3.1.2 Week 02 Game Planning

The goal was to make a game that resemble the actual game features but simultaneously reducing the amount of work that is necessary to implement them.

The first attempt was an AI vs AI game with only a graphical user interface that would show information about every AI player. This idea was discarded because the map design would have required advanced pathfinding to be traversable and the development of such a system would take too long. The result of the game design phase came very close to an actual real-time strategy game. Features like a procedural terrain generation, view spheres and unit diversity were cut out and unit pathfinding was solved with the use of a navigation mesh.

The figures below show a brief overview of implemented objects and their role in the game.

Game	Objective
- Player 1	Destroy all GameObjects from other Players.
- Player 2	Destroy all GameObjects from other Players.

Figure 6. Game Overview

Resources	Category	Function
- Food	World	Can be gathered by Citizen. Can be spent to buy Units and Buildings.
- Wood	World	Can be gathered by Citizen. Can be spent to buy Units and Buildings.
- Stone	World	Can be gathered by Citizen. Can be spent to buy Units and Buildings.
- Gold	World	Can be gathered by Citizen. Can be spent to buy Units and Buildings.
Units		
- Citizen	Civilian	Can gather Resources. Can build Buildings. Can fight Units. Can fight Buildings.
- Soldier	Military	Can fight Units. Can fight Buildings.
- Cavalier	Military	Can fight Units. Can fight Buildings.
- Artillery	Military	Can fight Units. Can fight Buildings.
Buildings		
- Capitol	Civilian	Can train Citizen.
- Mill	Civilian	Can store Resources.
- Sawmill	Civilian	Can store Resources.
- Mine	Civilian	Can store Resources.
- Barrack	Military	Can train Soldier.
- Stable	Military	Can train Cavalier.
- Foundry	Military	Can train Artillery.
- Tower	Military	Can fight Units.

Figure 7. Game Overview

The combat system was intended to function like rock paper scissors. Soldiers should be good against Cavaliers, Cavaliers against Artillery and Artillery having an advantage against Soldiers. This system was successfully used in the game Age of Empires II (*Ensemble Studios 1999*) and was used as an idea for this game. A simpler version of the same damage model as in Age of Empires was used to achieve this combat system. Each unit does a multiple set of damage and has a different armour value for each damage type. Bonus damage against unit types were left out because it was not necessary for this project.

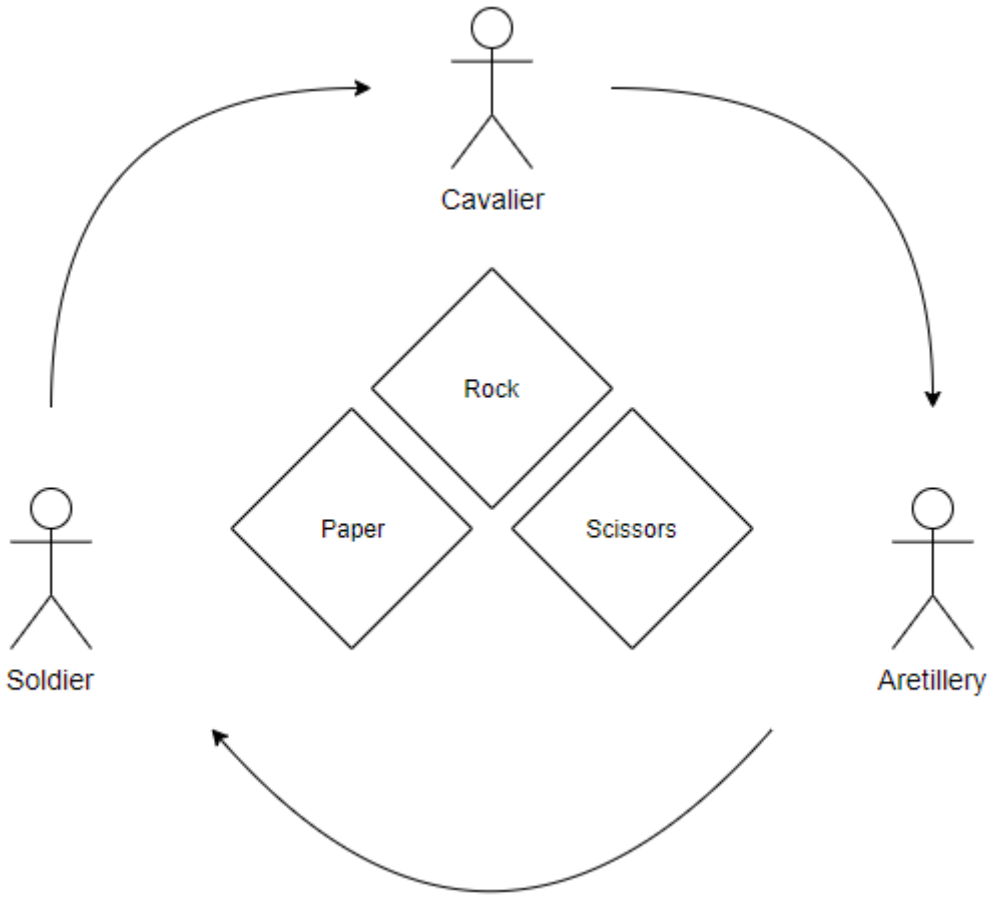

Figure 8. Combat System

Citizens can fight but are more useful while collecting resources or building new buildings. Units and buildings cost resources and can only be built if the player has collected enough resources. A player has won the game if all enemy units and buildings are destroyed.

Below is the feature list of the game functionality. The first list shows a simple overview of all features while the second list defines more precise objectives of the first list features. This list can also be seen in the “Feature List -

Game.xlsx" file, table 1-4 at /Documents. Each tab in the xlsx file shows the progress that was achieved each week of the development phase.

Feature Nr.	Category	Feature Name	Description
1.	Game	Map	*Create a Map for the game with all relevant features.*
2.	Game	Player	*Create two Player to battle each other.*
3.	Game	Resources	*Create Resources that can be gathered and spent to build Units and Buildings.*
4.	Unit	Citizen	*Create a Citizen that is responsible for Resource collection and building infrastructure.*
5.	Unit	Soldier	*Create a Soldier that can fight other Units and attack Buildings.*
6.	Unit	Cavalier	*Create a Cavalier that can fight other Units and attack Buildings.*
7.	Unit	Artillery	*Create an Artillery that can fight other Units and attack Buildings.*
8.	Building	Capitol	*Create Capitol that is able to buy and spawn more Citizen.*
9.	Building	Mill	*Create a Mill that is able to store the Resource Food.*
10.	Building	Sawmill	*Create a Sawmill that is able to store the Resource Wood.*
11.	Building	Mine	*Create a Mine that is able to store the Resources Stone and Gold.*
12.	Building	Barrack	*Create a Barrack that is able to buy and spawn more Soldiers.*
13.	Building	Stable	*Create a Stable that is able to buy and spawn more Cavaliers.*
14.	Building	Foundry	*Create a Foundry that is able to buy and spawn more Artillery.*
15.	Building	Tower	*Create a Tower that can fight only Units.*

Feature Nr.	Category	Feature Name	Description
1.	Map	Terrain	*A Terrain object for Units to walk over and Buildings to be build on.*
2.	Map	Navigation Mesh	*The NavMesh on the Terrain is used for Unit pathfinding.*
3.	Resources	Resources Prefab	*The Prefab allows the Resource to be placed multiple times on the Terrain.*
4.	Resources	Resource Type	*Resources have to be differentiated into: Food, Wood, Stone and Gold.*
5.	Resources	Resource Collection	*Resources should be collectable.*
6.	Resources	Resource Destruction	*Resources should be destroyed if there are no resources left in the object.*
7.	Player	Player Resource Account	*Players should be able to store Resources to buy Units and Buildings with it.*
8.	Player	Player Identification	*Players should be differentiated from each other with Tags and Colour.*
9.	Unit	Unit Class	*All Units should have a parent class.*
10.	Unit	Unit State Mashine	*Units should have a State Machine to simplify a broad range of actions.*
11.	Unit	Unit NavMesh Agent	*Units should be able to walk on the NavMesh with a NavMesh Agent.*
12.	Unit	Unit Perception	*Units should be able to perceive objects to trigger specific actions.*
13.	Unit	Unit Target Selection	*Units should have a automated Target Selection for their actions.*
14.	Unit	Unit Combat Stats (Defensive)	*Units should have defensive combat stats like armor and health.*
15.	Unit	Unit Combat Stats (Offensive)	*Units should have offensive combat stats like damage to be able to fight.*
16.	Unit	Unit Resource Cost	*Units should have Resource Costs because the Player should rely on Resource collection.*
17.	Unit	Unit Destruction	*Units should be destroyed if their health is zero.*
18.	Building	Building Class	*All Buildings should have a parent class.*
19.	Building	Unit Combat Stats (Defensive)	*Buildings should have defensive combat stats like armor and health.*
20.	Building	Building Resource Cost	*Buildings should have Resource Costs because the Player should rely on Resource collection.*
21.	Building	Building Resource Storage	*Buildings should be able to indicate if they can store Resources or not.*
22.	Building	Building Destruction	*Buildings should be destroyed if their health is zero.*
23.	Building	Building Size	*Buildings should indicate a building size to create an appropriate Construction.*
24.	Building	Building NavMesh Obstacle	*Buildings should be able to block Unit movement on the NavMesh.*
25.	Building	Construction Prefab	*The Prefab allows the Construction to be placed multiple times on the Terrain.*
26.	Building	Construction Combat Stats	*Constructions should have defensive combat stats like armor and health.*
27.	Building	Construction Destruction	*Constructions should be destroyed if their health is zero.*
28.	Building	Construction Building Information	*Constructions should store the Buildings Prefab to spawn it.*
29.	Building	Construction Building Functionality	*Constructions should be able to spawn the appropriate Building when the construction is finished.*
30.	Citizen	Citizen Work Information	*Citizens should be differentiated according to the actions they perform.*
31.	Citizen	Citizen Resource Account	*Citizens should be able to store Resources that can be delivered to an appropriate Building.*
32.	Citizen	Citizen Resource Collection	*Citizens should be able to collect Resources.*
33.	Citizen	Citizen Perception (Resources)	*Citizen should perceive surrounding Resources to cuntinue working.*
34.	Citizen	Citizen Resource Delivery	*Citizens should be able to deliver collected Resources to appropriate Buildings.*
35.	Citizen	Citizen Move	*Citizens should be able to move to an specific position.*
36.	Citizen	Citizen Attack	*Citizens should be able to attack a specific Unit and Building.*
37.	Soldier	Soldier Attack	*Soldiers should be able to attack a specific Unit or Building.*
38.	Soldier	Soldier Move	*Soldiers should be able to move to an specific position.*
39.	Cavalier	Cavalier Attack	*Cavaliers should be able to attack a specific Unit or Building.*
40.	Cavalier	Cavalier Move	*Cavaliers should be able to move to an specific position.*
41.	Artillery	Artillery Attack	*Artillery should be able to attack a specific Unit or Building.*
42.	Artillery	Artillery Move	*Artillery should be able to move to an specific position.*
43.	Capitol	Capitol Unit Production Pipeline	*Capitols should be able to spawn multiple Units respecting their build time.*
44.	Capitol	Capitol Citizen Production	*Capitols should be able to spawn Citizens.*
45.	Barrack	Barrack Unit Production Pipeline	*Barracks should be able to spawn multiple Units respecting their build time.*
46.	Barrack	Barrack Soldier Production	*Barracks should be able to spawn Soldiers.*
47.	Stable	Stable Unit Production Pipeline	*Stables should be able to spawn multiple Units respecting their build time.*
48.	Stable	Stable Cavalier Production	*Stables should be able to spawn Cavaliers.*
49.	Foundry	Foundry Unit Production Pipeline	*Foundries should be able to spawn multiple Units respecting their build time.*
50.	Foundry	Foundry Artillery Production	*Fourdries should be able to spawn Artillery.*
51.	Tower	Tower Perception (Unit)	*Tower should perceive surrounding Units.*
52.	Tower	Tower CombatStats (Offensive)	*Tower should have offensive combat stats like damage to be able to fight.*
53.	Tower	Tower Target Selection	*Tower should have a automated Target Selection for their actions.*
54.	Tower	Tower Attack	*Tower should be able to attack a specific Unit.*

Figure 9. Feature List - Game

3.1.3 Week 03 Base Structure

The focus was set on the creation of all base classes and the map in the beginning of the development phase because it speeds up the testing if all independent objects and classes already exists. The map consists of a terrain and the resource objects. This allowed to verify that the Nav Mesh, Nav Mesh Obstacles and the Nav Mesh Agent were working as intended for the map size.

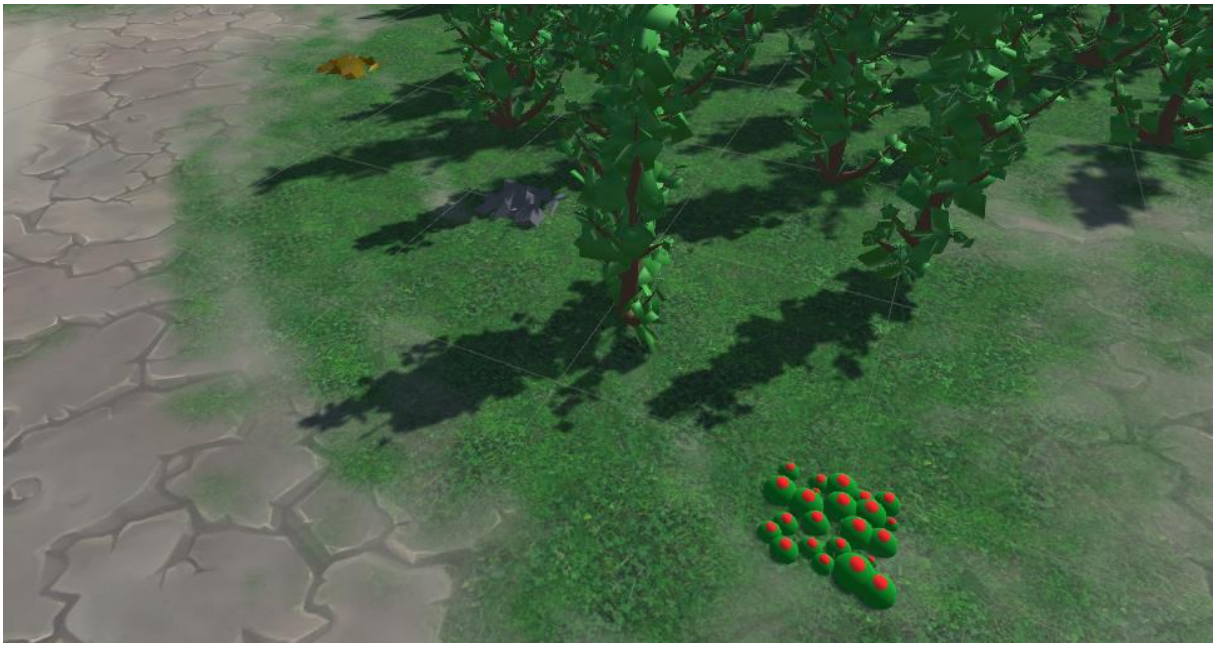

Figure 10. Map

The player class was also finished due to its few features that are necessary. One part of it, the Resource class dependency, were already implemented to fill the map with objects. As seen in Figure 7. Game Overview, there are four resource types (all resources can be seen in Figure 10. Map).

The Resource class is used on every resource object on the map and can be collected. The player got a different definition called PlayerResource since it must store all resources separately.

```
// The resource type to distinguish between resources.
public enum ResourceType
{
    Food,
    Wood,
    Stone,
    Gold
}
```

Figure 11. Resources

```
// The player tag to distinguish between players.
public enum PlayerTag
{
    Gaia,       // Gaia is the ancestral mother of all life: the primal Mother Earth goddess. Used for non player entities.
    Player1,
    Player2,
    Player3,
    Player4,
    Player5,
    Player6,
    Player7,
    Player8
}
```

Figure 12. Player Tag

```
public class Resource : MonoBehaviour
{
    public ResourceType resourceType;
    public int amount;

    /// <summary>
    /// Transfer an amount of resources from this object to another object by using this method.
    /// </summary>
    /// <param name="amount">The amount to be removed</param>
    /// <returns>the actual amount as int.</returns>
    public int GatherResource(int amount)...

    private void Update()
    {
        // If the resource object is empty, destroy it.
        if (amount <= 0)
        {
            Destroy(gameObject);
        }
    }
}
```

Figure 13. Resources

```
[System.Serializable]
public struct PlayerResource
{
    public int food;
    public int wood;
    public int stone;
    public int gold;

    /// <summary>
    /// Add a type of resource to the player resources.
    /// </summary>
    /// <param name="amount">The amount that gets stored.</param>
    /// <param name="resourceType">The type of the resource.</param>
    public void AddResource(ResourceTransferStruct resourceTransferStruct)...

    /// <summary>
    /// Removes the amount of the building/unit costs from the player resources.
    /// </summary>
    /// <param name="buildCost">The build costs of the building/unit</param>
    public void RemoveResources(BuildCost buildCost)...
}
```

Figure 14. Player Resources

```
public class Player : MonoBehaviour
{
    // The player tag is used to indentify a player.
    public PlayerTag playerTag;

    // The Color of the player is used to colour buildings and units.
    public Color playerColor;

    // The resources of a player.
    public PlayerResource resources;
}
```

Figure 15. Player Class

Parts of the Unit parent class and the Building parent class were implemented, like the Nav Mesh Agent and resource dependent fields, but not most of all required features. Most of the time was spent on the unit and building design to better distinguish between players, units and buildings.

14 / 54 features were implemented during the first week of the development.

3.1.4 Week 04 State Machine

Since all base classes and dependencies were implemented, of course without functionality, the Unit class and the Citizen class were prioritized. This was because the citizen has the highest complexity of all units. The citizen can do everything a normal unit can but has additional features like collecting resources or build buildings. If the citizen would be successfully implemented, the functionality of all other units would be done by only copy it from the citizen. Redundant functions for each unit, like the attack, was not used as a single function in the parent class because this way it can be re-designed for each unit if desired.

Each unit got a perception range to identify enemies in range. This is used to give units the ability to do certain actions without the interaction of the player. This will be discussed later. The citizen got an additional perception range to identify resources because the citizen is not a combat unit alone. This is also used to switch actions or continue actions without player interaction. To simplify the control of the maybe massive amount of citizen in the game, the WorkerGroup was created, showing in what action the citizen is engaged. This will be automatically handled by the state machine. To carry resources, the citizen got a Backpack class which allows the citizen to carry a maximum amount of a resource.

```
// The WorkerGroup is used to identify citizen actions.
public enum WorkerGroup
{
    Idle,           // The citizen does nothing.
    GatherFood,     // The citizen is collecting food.
    GatherWood,     // The citizen is collecting wood.
    GatherStone,    // The citizen is collecting stone.
    GatherGold,     // The citizen is collecting gold.
    Builder,        // The citizen is building.
    Fighter,        // The citizen is in combat.
    Other           // The citizen is just walking.
}
```

Figure 16. Citizen Worker Group

```
public class Backpack
{
    public int maxCarryCapacity;
    public int currentAmount;
    public ResourceType resourceType;

    /// <summary>
    /// Create a new backpack to store resources.
    /// </summary>
    /// <param name="maxCarryCapacity">The capacity of the backpack.</param>
    public Backpack(int maxCarryCapacity)...

    /// <summary>
    /// Create a new backpack to store resources.
    /// </summary>
    /// <param name="maxCarryCapacity">The capacity of the backpack.</param>
    /// <param name="currentAmount">The amount of resources in the backpack.</param>
    /// <param name="resourceType">The type of the current resource in the backpack.</param>
    public Backpack(int maxCarryCapacity, int currentAmount, ResourceType resourceType)...

    /// <summary>
    /// Returns if the backpack is full.
    /// </summary>
    /// <returns>true if the backpack is full.</returns>
    public bool IsFull()...

    /// <summary>
    /// Returns the available space in the backpack.
    /// </summary>
    /// <returns>the available space as int.</returns>
    public int GetFreeSpace()...

    /// <summary>
    /// Add resources to the backpack. If the new ResourceType is different, the current resources are lost.
    /// </summary>
    /// <param name="amount">The amount that gets stored. Normaly the citizen collection capacity.</param>
    /// <param name="resourceType">The type of the resource, decides if the backpack gets cleared.</param>
    public void AddResource(int amount, ResourceType resourceType)...

    /// <summary>
    /// Returns the amount of resources that are removed from the backpack.
    /// </summary>
    /// <returns>the amount of removed resources as int.</returns>
    public int RemoveResource()...

    /// <summary>
    /// Returns the amount of resources that are removed from the backpack with the coresponding ResourceType.
    /// </summary>
    /// <returns>the amount of removed resources and the ResourceType as struct.</returns>
    public ResourceTransferStruct RemoveResourceWithDefinition()...
}
```

Figure 17. Citizen Backpack

The state machine of each unit is a whole chapter in its own and offers many game enriching possibilities. To avoid the risk of feature creep, the focus was set on the following actions and states. Since all combat units have the same

state machine, the figures will only show the citizen state machine diagram; All combat related actions are structured the same way as in the citizen state machine diagram, all diagrams can be seen in /Documents/Diagrams:

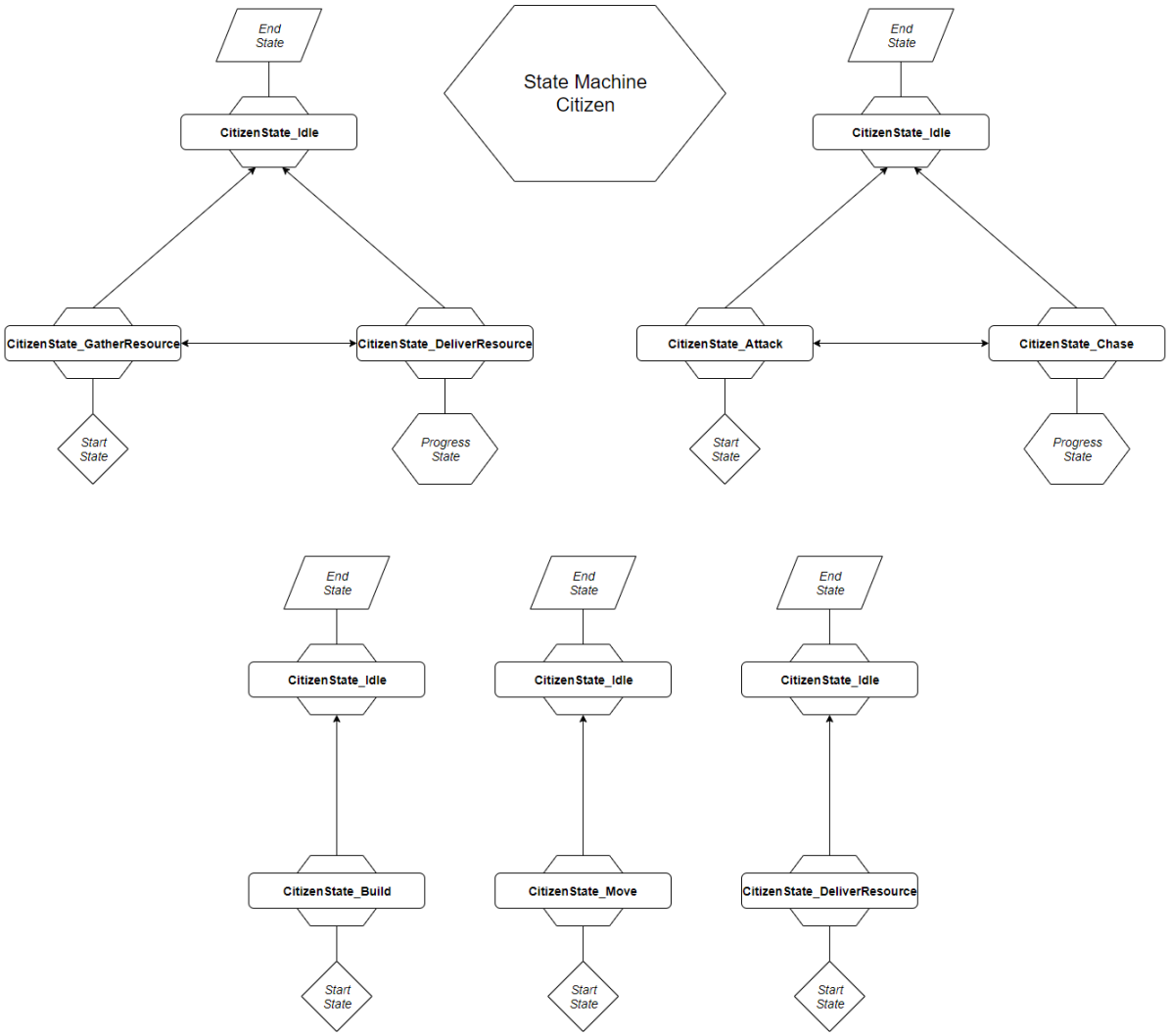

Figure 18. Citizen State Machine Diagram

The citizen state machine has two cycle-state and three single-state actions. The player calls a specific action and the state machine changes the states. As shown in the figure, one state appears in a cycle-state action and in a single-state action, this state has a different behaviour if it is called as a start state or

as a progress state. The cycle-state actions allow the citizen in relation to the earlier described perception ranges to continue working even if the original specified target gets destroyed.

Instead of just an attack order, the state machine could have been extended with states that allow the unit to patrol an area or to make an attack move.

Below is an example of the citizen state to collect resources:

```
public override void Enter(Citizen citizen)
{
    // Is the backpack full of the target resource?
    if (citizen.backpack.resourceType == citizen.targetResource.resourceType)
    {
        if (citizen.backpack.IsFull())
        {
            // Bring the resource to the nearest store.
            citizen.ChangeState(CitizenState_DeliverResource.Instance);
        }
    }
}

public override void Execute(Citizen citizen)
{
    // Is the backpack full?
    if (citizen.backpack.IsFull())
    {
        // Bring the resource to the nearest store.
        citizen.ChangeState(CitizenState_DeliverResource.Instance);
    }

    // Check if the resource disappeared because there are no resources in the object anymore.
    if (citizen.targetResource == null)
    {
        // Check for other resources of the same type in the area.
        if (!citizen.SelectNewResource())
        {
            // Deliver all current resources in the backpack if there are no other resources around.
            if (citizen.targetResource == null && citizen.backpack.GetFreeSpace() > 0)
            {
                citizen.ChangeState(CitizenState_DeliverResource.Instance);
            }
            else
            {
                citizen.ChangeState(CitizenState_Idle.Instance);
            }

        }
    }

    // Gather the resource if the target resource is in range
    if (citizen.targetResource != null && citizen.perceivedObjectsInRange.Contains(citizen.targetResource.gameObject))
    {
        citizen.navMeshAgent.ResetPath();
        GatherResource(citizen);
    }
    else
    {
        // Move to the target resource
        if (citizen.targetResource != null && citizen.navMeshAgent.destination != citizen.targetResource.transform.position)
        {
            citizen.navMeshAgent.SetDestination(citizen.targetResource.transform.position);
        }
    }
}
```

Figure 19. Citizen State Gather Resources

The build state required the construction object to be complete otherwise the building would not appear. Each building prefab has a specific size due to good design, so the creation of accurate construction sites was no problem.

```
// The size of the building.
public enum ConstructionSiteSize
{
    Size4x4,
    Size8x8
}
```

Figure 20. Building Size

```
public class ConstructionSite : Building
{
    [Header("Building Options")]
    // The name of the building it will become. Must be the same as the name of the prefab.
    public string building;

    [Header("Build Options")]
    // Build time in seconds
    public float buildTimeLeft;


    private void Start()...

    private void Update()...


    /// <summary>
    /// Builds the building.
    /// </summary>
    private void FinishConstruction()
    {
        // Create the new building.
        GameObject gameObject = (GameObject) Instantiate(Resources.Load(building), transform.position, transform.rotation);

        // Set the player of the new building.
        Building finishedBuilding = gameObject.GetComponent<Building>();
        finishedBuilding.player = player;
        finishedBuilding.playerTag = playerTag;

        // Destory this object.
        Destruct();
    }
}
```

Figure 21. Construction Site Class

20 / 40 features were implemented during the second week of the development.

3.1.5 Week 05 Production Pipe

The last allocated week for the task to create a game was used to implement the last states in the state machine, such as attack and chase. The states were copied to the three combat units. As mentioned, the at this time redundant

code can be used to make every unit unique. For testing purposes, each unit shoots a red ray during the damage calculations, different for each objects height. The time consumption for this was about 5 minutes. The last step was to implement the functionality for buildings to allow them to buy new units if the player has enough resources. This was done by the creation of the Production Pipe. The pipe would store the remaining build time for each bought unit and update it, then create a new object in front of the building.

```
public class ProductionPipe : MonoBehaviour
{
    // A object that hold the build time left for each unit in production.
    private class BuildTime
    {
        public Unit unit;
        public float buildTimeLeft;

        public BuildTime(Unit unit, float buildTime)
        {
            this.unit = unit;
            buildTimeLeft = buildTime;
        }
    }

    private Building building;
    private List<BuildTime> productionPipe = new List<BuildTime>();

    public ProductionPipe(Building building)
    {
        this.building = building;
    }

    /// <summary>
    /// Adds a unit to the production pipe using the BuildTime class.
    /// </summary>
    /// <param name="unit">The unit object that has to be cloned.</param>
    public void AddUnit(Unit unit)...

    /// <summary>
    /// Updates the build time left for each unit in the production pipe per frame.
    /// </summary>
    public void UpdatePipe()...

    /// <summary>
    /// Spawns an instance of the unit.
    /// </summary>
    /// <param name="unit">The unit object that has to be cloned.</param>
    private void SpawnUnit(Unit unit)...

    /// <summary>
    /// Returns if the player has enough resources to build a unit.
    /// </summary>
    /// <param name="player">The player who wants to build.</param>
    /// <param name="building">The unit that is about to be trained.</param>
    /// <returns>true if the player has enough resources.</returns>
    public bool EnoughResources(Unit unit)...
}
```

Figure 22. Building Production Pipe

The stats for each building and unit will be balanced during the AI testing because at the current stage it is very difficult to achieve realistic scenarios. For now, each unit and building have values that were reasonable for the current stage of the game.

The current class diagram of all game classes, structs and enumerations can be seen below or in /Documents/Diagrams:

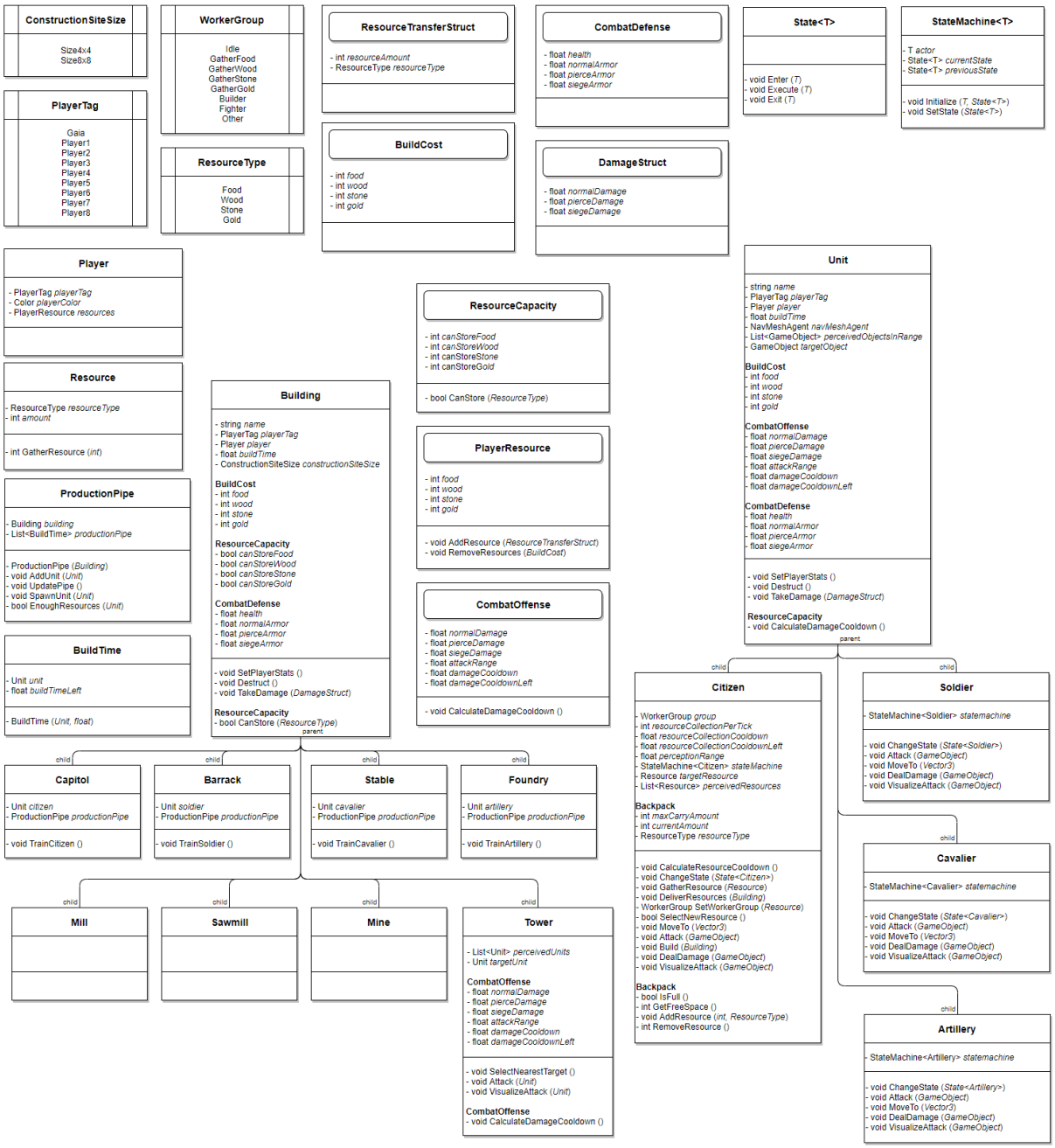

Figure 23. Game Class Diagram

20 / 20 features were implemented during the third week of the development.

3.1.6 Week 06 Plan Update Part 1

All features were implemented in the allocated time and the game structure was ready. Only the missing user interface, to interact with the game in any form and the AI opponent prevent the game from being playable.

The next task was to implement an Artificial Intelligence (AI) algorithm, so that two AI controlled player could fight against each other. Since half of the project time were already spent on an unexpected but predictable task, the initial AI design might not be implemented anymore and one of the alternatives 2.5.3 Alternatives had to be considered. That means, either reducing complexity of the abstraction, reducing the complexity of the algorithms or just using a single algorithm for the whole AI.

The decision that has been made, was to use the Hierarchical Task Network (HTN) algorithm. The HTN algorithm can satisfy both mentioned alternatives, reducing complexity and using a single algorithm. Also, the HTN algorithm provided a new challenge because it is not a simple algorithm that only reacts to the game state.

Hierarchical Task Network is a planner algorithm that uses a tree search approach to select tasks to create a plan. It uses abstract tasks that are decomposed into other abstract tasks or primitive tasks. Primitive tasks cannot be decomposed into smaller tasks. The algorithm traverses the tree and selects tasks that whose preconditions are satisfied by the current or predicted world state to that time. Each task effects the world state and thus generate through the predicted world state new possible plans.

An example of task decomposition:

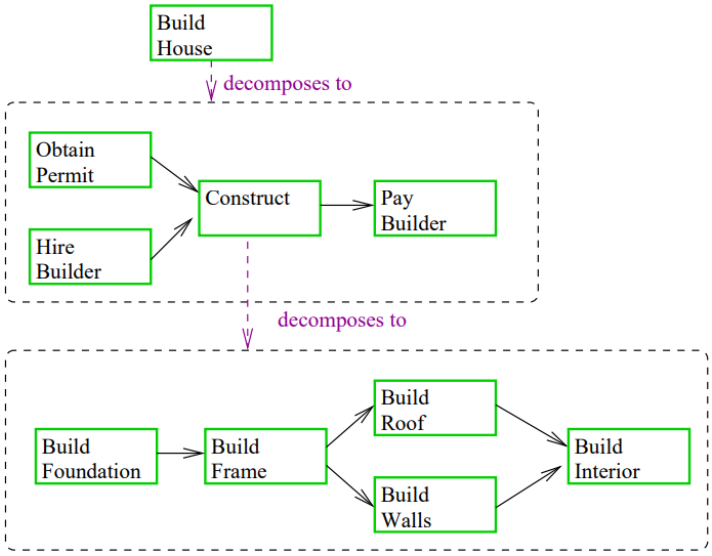

Figure 24. HTN Task Decomposition

An advantage of the HTN algorithm is, compared to Goal Orientated Action Planning (GOAP), that it uses a predefined tree to search for possible actions and can cut down entire branches if an abstract task fails, whereas GOAP uses a pathfinding algorithm to search all present tasks for the one with the least costs (*Anon. n.d.*).

The downside of HTNs are that they need a separate world state since a task influences the world state. To use this information for the next task's preconditions it must store all necessary information separately. This is a huge task to implement in a short amount of time; C# allows to deep copy objects with some functions (*Microsoft, 2018*) but it is still a slow process for the whole game objects.

To see a comparison to other algorithms, see Appendix F.

In the first week of the development phase a new Gantt Chart was presented to show the implementation of the new task “create the game”. It is shown again below alongside with an updated version giving a more detailed overview of the next week’s tasks for the AI development.

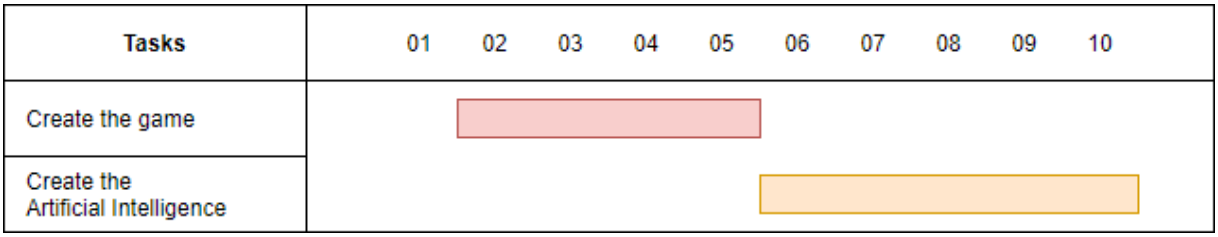

Figure 25. Updated Gantt Chart

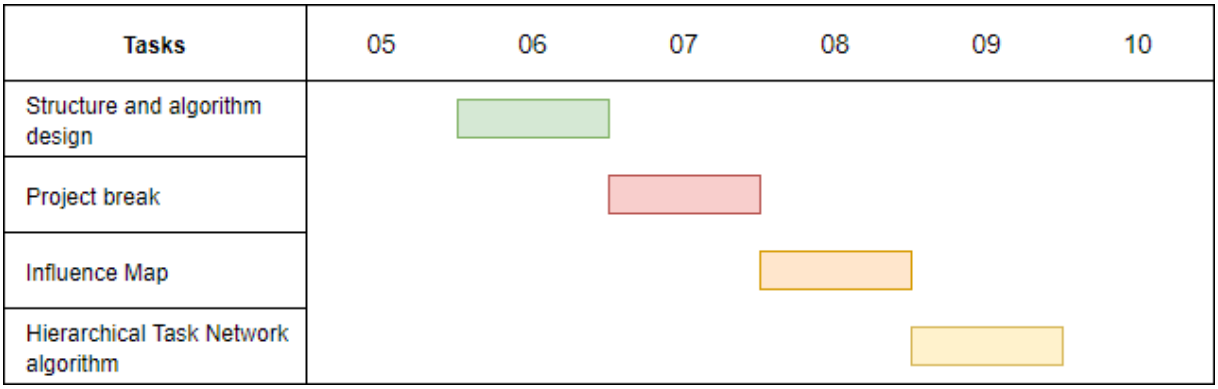

Figure 26. Updated Gantt Chart

Since other assignments are near their deadline, a week of the development is not available for the project and is listed as a project break. This, and the last week of the project, are not planned for code implementation because the project must be prepared for the hand in. This leaves very little time for the actual development.

3.1.7 Week 07 Plan Update Part 2

As described in the updated Gantt Chart for week 07, no significant progress was achieved. This section is used to show what progress was achieved during the structure and algorithm design phase.

As seen in the updated Gantt Chart, the development of an influence map was planned additionally to the Hierarchical Task Network (HTN). The influence map will be used to enable a better way of placing buildings or finding good spots to station combat units at vulnerable positions, rather than placing something random on the map without purpose.

The influence map is a tool that helps to read the map easier and allows to give a good amount of information what is going on around the map. For example, it allows to identify points of influences, tensions and vulnerability (*Hansson, 2010*).

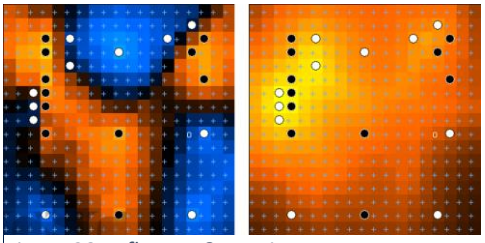

Figure 28. Influence & Tension

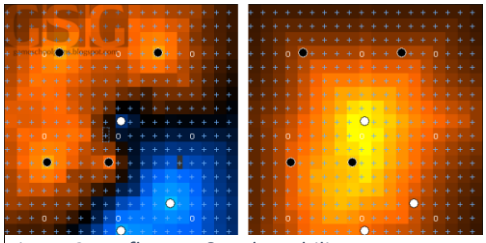

Figure 27. Influence & Vulnerability

This information can then be used to strategically place units and buildings on the map, to build a secure base or attack the enemy at vulnerable points.

Below is the feature list of the Artificial Intelligence functionality. The first list shows a simple overview of all features while the second list defines more precise objectives of the first list features. This list can also be seen in the

“Feature List - AI.xlsx” file, table 1-3 at /Documents. Each tab in the xlsx file shows the progress that was achieved each week of the development phase.

Feature Nr.	Category	Feature Name	Description
1.	Hierarchical Task Network	Tree Search	*Create an algorithm that travese a tree of tasksto build a plan.*
2.	Hierarchical Task Network	Abstract Task	*Create an abstract task that is used to indicate a set of actions.*
3.	Hierarchical Task Network	Primitive Task	*Create a primitive task that is used to hold actions for the plan.*
4.	Hierarchical Task Network	Action	*Create an action that stores methods for the AI to controll game units and buildings.*
5.	Hierarchical Task Network	Actions	*Create a library of methods the AI can use to controll units and buildings in the game.*
6.	Hierarchical Task Network	Preconditions	*Create a library of mothods for the algorithm to compare world states to add tasks to the plan.*
7.	Hierarchical Task Network	Effects	*Create a library of effects that can alter the predicted world state.*
8.	Hierarchical Task Network	Plan	*Create a plan that stores a set of actions that can be iterated to controll the game.*
9.	Hierarchical Task Network	World State	*Create a copy of the world state that can be influenced by the effects and is used to check preconditions.*
10.	Influence Map	Influence Map	*Create an influence map base class.*
11.	Influence Map	Grid Position	*Create a grid position object that stores information of each point in the influence map.*
12.	Influence Map	Player Object	*Create a new class for each building and unit to store influences.*

Feature Nr.	Category	Feature Name	Description
1.	Tree Search	Task Iteration	*Iterate through abstract tasks and check if preconditions are satisfied, search all subtasks for suitable primitive tasks, add the action to the pla and update the predicted world state.*
2.	Tree Search	Task Selection	*Keep the initial task tofall back on an action if no plan could be made.*
3.	Task	Task Differentiation	*Seperate between abstract and primitive tasks.*
4.	Abstract Task	Task Preconditions	*Tasks can iterate through all relevant preconditions.*
5.	Abstract Task	Task Subtasks	*Tasks hold a list of subtasks.*
6.	Primitive Task	Task Action	*Primitive Tasks hold actions the AI can use.*
7.	Action	Duration	*Actions can last a specific time before they are finished.*
8.	Action	Delegate	*Actions contain a delegate to the method the AI will call in the plan.*
9.	Actions	Library	*Create a library of methods that control the units and buildings behaviour in the game.*
10.	Preconditions	Library	*Create a library of methods that are used to compare the world state.*
11.	Effects	Library	*Create a library of methods that alter the predicted world state.*
12.	Plan	Action List	*The plan needs to hold a list of all actions that are executed.*
13.	World State	Citizen Count	*The amount of citizen a player has available.*
14.	World State	Soldier Count	*The amount of soldiers a player has available.*
15.	World State	Cavalier Count	*The amount of cavaliers a player has available.*
16.	World State	Artillery Count	*The amount of artillerie a player has available.*
17.	World State	Citizen Group	*The amount of citizen in a specific worker group.*
18.	World State	Capitol Count	*The amount of capitols a player has available.*
19.	World State	Mill Count	*The amount of mills a player has available.*
20.	World State	Sawmill Count	*The amount of sawmills a player has available.*
21.	World State	Mine Count	*The amount of mine sa player has available.*
22.	World State	Barrack Count	*The amount of barracks a player has available.*
23.	World State	Stable Count	*The amount of stables a player has available.*
24.	World State	Foundry Count	*The amount of foundries a player has available.*
25.	World State	Tower Count	*The amount of towers a player has available.*
26.	World State	Citizen Fields	*All information a citizen has, i.e. health, position, ...*
27.	World State	Soldier Fields	*All information a soldier has, i.e. health, position, ...*
28.	World State	Cavalier Fields	*All information a cavalier has, i.e. health, position, ...*
29.	World State	Artillery Fields	*All information a artillery has, i.e. health, position, ...*
30.	World State	Capitol Fields	*All information a capitol has, i.e. health, position, ...*
31.	World State	Mill Fields	*All information a mill has, i.e. health, position, ...*
32.	World State	Sawmill Fields	*All information a sawmill has, i.e. health, position, ...*
33.	World State	Mine Fields	*All information a mine has, i.e. health, position, ...*
34.	World State	Barrack Fields	*All information a barrack has, i.e. health, position, ...*
35.	World State	Stable Fields	*All information a stable has, i.e. health, position, ...*
36.	World State	Foundry Fields	*All information a foundry has, i.e. health, position, ...*
37.	World State	Tower Fields	*All information a tower has, i.e. health, position, ...*
38.	World State	Food	*The amount of food a player has available.*
39.	World State	Wood	*The amount of wood a player has available.*
40.	World State	Stone	*The amount of stone a player has available.*
41.	World State	Gold	*The amount of gold a player has available.*
42.	Influence Map	Size	*The size of the influence map.*
43.	Influence Map	Decay	*The decay of the influence over two fields.*
44.	Influence Map	Player Objects	*A list of all object that are influencing the map.*
45.	Influence Map	Array	*An array of two dimensions to simulate the map.*
46.	Influence Map	Create Map	*The map has to be created from the size and the array filled with the grid position information.*
47.	Influence Map	Update Map	*The map has to be updated, carrying influence over to other grid positions.*
48.	Influence Map	Update Player Objects	*The player object has to be updated to represent the correct positions on the map and setting the influence to that position.*
49.	Grid Position	Position	*The object needs to hold ist sample position.*
50.	Grid Position	Connections	*A list of all grid positions that are directly connected to this grid position.*
51.	Grid Position	Influence	*the influence value that is present at this position.*
52.	Player Object	Player Influence	*The amount of influence this object has.*
53.	Player Object	Influence Range	*The range of the influence of this object.*
54.	Player Object	Grid Position	*The position on the influence map grid.*

Figure 29. Feature List - AI

3.1.8 Week 08 Influence Map

To be able to implement the influence map in a more simpler way, the structure of some game classes had to be adjusted. The new class player object is used from the building and the unit class to inherit the player fields. Everything else remains the same.

```
public abstract class PlayerObject : MonoBehaviour
{
    // A reference to the influence map to add this object to it on creation.
    public NewInfluenceMap influenceMap;

    // The position on the influence map.
    [System.NonSerialized]
    public int gridPosition_x;
    [System.NonSerialized]
    public int gridPosition_z;

    // The strength of the influence. 0 - 1.
    public float influence;
    public float influenceRange;

    // The player reference.
    public PlayerTag playerTag;
    public Player player;

    /// <summary>
    /// Sets the playerColor of the object and the playerTag.
    /// </summary>
    public void SetPlayerStats()...

    /// <summary>
    /// Adds the object to the influence map.
    /// </summary>
    protected abstract void AddToPlayerList();
}
```

Figure 30. Player Object

The grid position is used to store the influences from each player object and all connected grid positions, to transfer the influence.

```
public class GridPosition
{
    // The position on the influence map.
    public int mapPosition_x;
    public int mapPosition_z;

    // All neighbor grid positions.
    public List<KeyValuePair<GridPosition, float>> connections;

    // Influences from all Players.
    public List<KeyValuePair<Player, float>> influences;

    public GridPosition(int x, int z)
    {
        mapPosition_x = x;
        mapPosition_z = z;

        connections = new List<KeyValuePair<GridPosition, float>>();
        influences = new List<KeyValuePair<Player, float>>();
    }
}
```

Figure 31. Grid Position

The influence map class has reached the following state at the end of the week:

```
public class NewInfluenceMap : MonoBehaviour
{
    // The size of all sides of the map.
    public int influenceMapSize;

    // The space between two grid positions.
    public int influenceMap_gridSize;

    // The width and height of the map.
    private int influenceMap_width;
    private int influenceMap_height;

    // The decay of the influence over two grid positions.
    public float influenceDecay;

    // The amount of updates per second.
    public float updateFrequency;
    private float updateFrequencyLeft = 0F;

    // A list of all objects that influences the map.
    private List<PlayerObject> playerObjects;

    // The influence map and the influence map buffer.
    private GridPosition[,] map;
    private GridPosition[,] mapBuffer;

    // The texture to visualise the influence spread over the map.
    private Texture2D influenceMap_texture;

    // The momentum is used for the influence spread calculation.
    float momentum = 5F;


    private void Start()...

    private void Update()...


    /// <summary>
    /// Set the grid positions and connect it to its neighbors.
    /// </summary>
    private void CreateMap()...

    /// <summary>
    /// Set the influence according to its strength and delay on all grid positions on the entire map.
    /// </summary>
    private void UpdateMap()...

    /// <summary>
    /// Set the buffer to the previous influence map state.
    /// </summary>
    private void UpdateBuffer()...

    /// <summary>
    /// Set the influence of all player objects to the current grid position.
    /// </summary>
    private void UpdatePlayerObjects()...

    /// <summary>
    /// Adds influence a player object has to a grid position.
    /// </summary>
    /// <param name="x">The array index of the X axis.</param>
    /// <param name="z">The array index of the Z axis.</param>
    /// <param name="player">The player that is influencing the grid position.</param>
    /// <param name="value">The intensity of the influence of the player object.</param>
    public void CreateInfluence(int x, int z, Player player, float value)...

    /// <summary>
    /// Adds a player object to the list of objects that can interact with the influence map.
    /// </summary>
    /// <param name="playerObject">The player object.</param>
    public void AddPlayerObject(PlayerObject playerObject)...

    /// <summary>
    /// Adds all player objects to the list of objects that can interact with the influence map.
    /// </summary>
    private void Fill()...

    // private void Draw() ...
}
```

Figure 32. Influence Map

While everything except the influence calculation is working, there is no time to keep working on the correction of this error. It is possible to add all existing objects to the influence map and create influence at their position. It is also possible to visualise the map in game but due to the fault in the calculations it is not visible; This feature is included as the comment section at the end of the class.

Since the time is up for this task and the fault could not be corrected, the influence map is not functional in the game and will therefore not be used by the Artificial Intelligence.

Below is the updated part of the class diagram with all new calluses that were added.

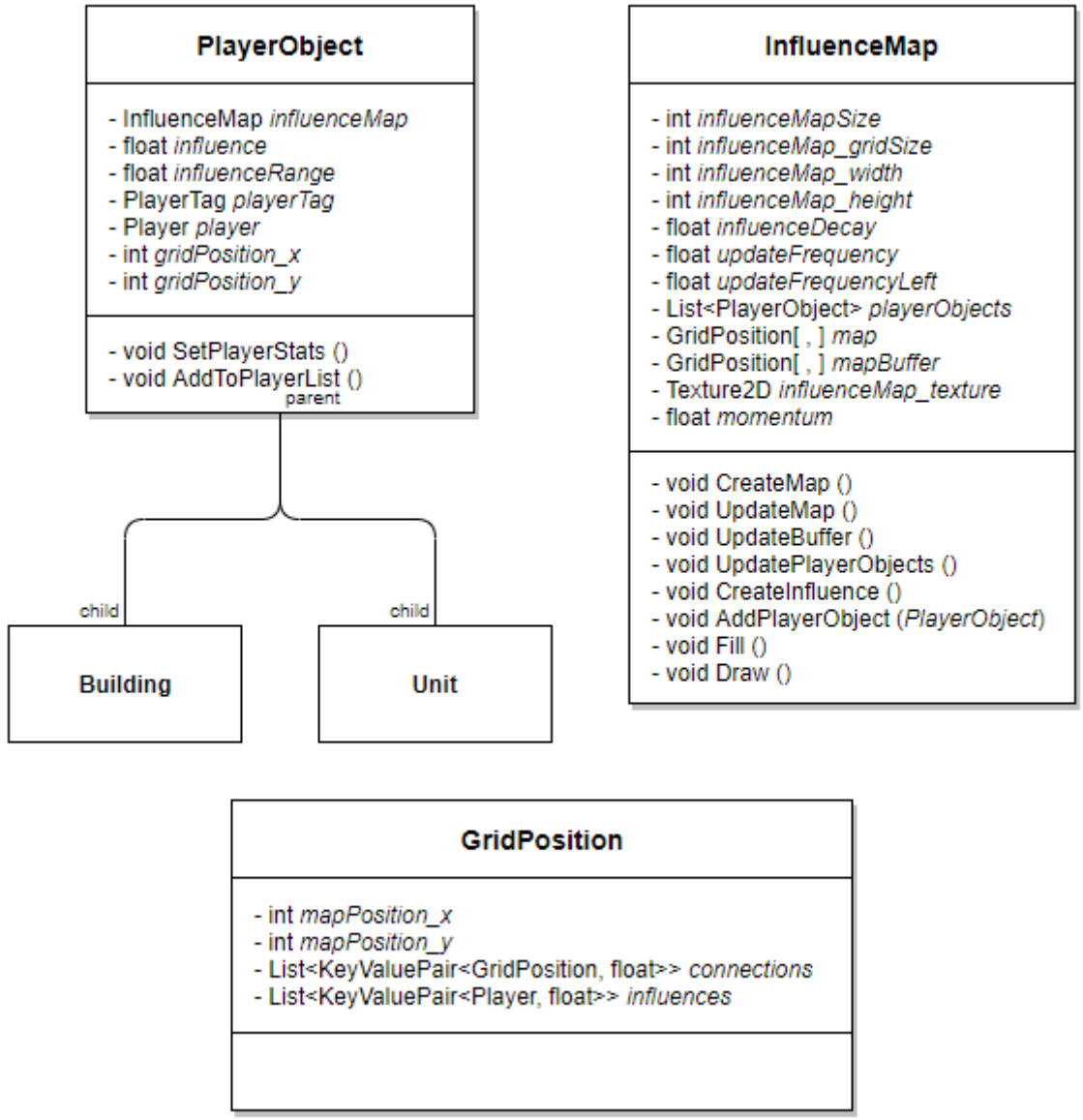

Figure 33. Updated Class Diagram

3.1.9 Week 09 Hierarchical Task Network

The Hierarchical Task Network (HTN) turned out to be even more trickier than expected. Some work on the functionality could not be finished because one week is not enough to fill up a library of methods while having trouble with the correct execution of the planed actions.

The algorithm can traverse tasks and check preconditions to add actions to the plan, but do not affect the predicted world state for other tasks. Creating all the necessary information to resemble a prediction of the current world state would take too much time, still it is an essential part of the planning process.

```
public class HierarchicalTaskNetwork
{
    Task initialTask;
    Task currentTask;
    Plan plan;

    public HierarchicalTaskNetwork(Task initialTask)
    {
        this.initialTask = initialTask;
        currentTask = initialTask;
        plan = new Plan();
    }

    /// <summary>
    /// Traverses the tree and creates a plan.
    /// </summary>
    /// <returns>The plan.</returns>
    public Plan Execute()
    {
        Perform(initialTask);
        return plan;
    }

    /// <summary>
    /// Traverse the tree and checks all preconditions and/or adds actions to the plan.
    /// </summary>
    /// <param name="currentTask">The tasks that is currently checked.</param>
    private void Perform(Task currentTask)
    {
        // Check if the task is primitive.
        if (!currentTask.isPrimitiveTask)
        {
            // Check the preconditions of the abstract task
            if (currentTask.CheckPreconditions())
            {
                // Update the predicted world state.

                // Traverse all subtasks.
                for (int i = 0; i < currentTask.subtasks.Count; i++)
                {
                    Perform(currentTask.subtasks[i]);
                }
            }
        }
        else
        {
            // Add the action of the primitive task to the plan.
            plan.AddActionToPlan(currentTask.action);
        }
    }
}
```

Figure 34. Hierarchical Task Network

```
public abstract class Task
{
    // The strategy manager holds all information.
    public StrategyManager strategyManager;

    public bool isPrimitiveTask;
    public Task parentTask;

    // Holds all preconditions of this task. Abstract only.
    public List<Precondition> preconditions;

    // Holds all subtasks of this task. Abstract only.
    public List<Task> subtasks;

    // Holds the action of this task. Primitive only.
    public Action action;

    public abstract bool CheckPreconditions();
    public abstract void AddPrecondition(Precondition precondition);
    public abstract void AddSubtask(Task subtask);
}
```

Figure 35. HTN Task

Another problem is the actual execution of each task. Some tasks can be instantly completed, some other tasks need a certain amount of time to be completed and other tasks could last an unpredictable amount of time. This would not be a problem if other tasks were not dependent from the outcome of these tasks.

For example, a citizen gets build in a capitol, this process lasts some seconds before the citizen can spawn in front of the building. Tasks that involve this citizen must wait until the citizen exist. When the capitol gets destroyed and the citizen could not spawn, the plan cannot be completed anymore.

```
public delegate object[] ActionDelegate(StrategyManager strategyManager, object[] data);

public class Action
{
    // The duration of this task until the world state is changed.
    public float duration;

    // The method that is called in the plan.
    public ActionDelegate actionDelegate;

    public Action(float duration, ActionDelegate actionDelegate)...

    /// <summary>
    /// Calls the store action method.
    /// </summary>
    /// <param name="strategyManager">The strategy manager of the player.</param>
    /// <param name="data">The necessary information to call this method.</param>
    /// <returns>Returns information that can be essential to other methods.</returns>
    public object[] Execute(StrategyManager strategyManager, object[] data)
    {
        return actionDelegate.Invoke(strategyManager, data);
    }
}
```

Figure 36. HTN Action

Another problem is the way to give a specific unit a task. As seen in figure 36. HTN Action, an array with the need information is given to the method; This is not an optimal solution because the parameters can change based on the task before. An approach would be, to let another algorithm handle this part.

There are plenty of problems that are causing problems with the HTN at this stage. Even if the HTN would work at its best performance, there are still missing parts of the program that are essential to the execution of the plan. The influence map, for example would help to place buildings at proper positions.

The feature list of the AI functionality shows how much progress was achieved in these two weeks. This list can be seen in the “Feature List - AI.xlsx” file, table 1-3 at /Documents. The first list shows a simple overview of all features while the second list defines more precise objectives of the first list features. Each tab in the xlsx file shows the progress that was achieved each week of the development phase. Approximately three to four weeks more would have been necessary for all missing features and bug fixing. If the first five weeks were not designated for game development, this might have been possible.

3.1.10 Week 10 Hand In

This week was left out because the documentation had to be finished.

It is also important to have some buffer time if things are delaying the hand in.

3.2 Reflection and Conclusion

The initial plan was it to make an Artificial Intelligence (AI) that can defeat an enemy in a real-time strategy (RTS) game.

The focus was set on the design of the algorithm but not much on the game itself. After the planning was completed and the development had started, it was noticeable that a system was needed that dynamically handled the AI's input and output. The game was created in five weeks, but it could have been a better choice to search a bit more for similar projects. Many academical papers for AI in RTS games uses StarCraft (*Blizzard Entertainment, 1998*) as testing ground to test their AIs. They are using the StarCraft Application Programming Interface (API) (*Blizzard Entertainment, 2017*) to use their AI in an already developed game. This would have saved a lot of time and allowed to focus more on the AI.

The problem was, that the idea of an own RTS game, which was around for some time, blocked the view to this time.

The second half of the project, the AI development, could have been better structured. An even more simpler algorithm, like a behaviour tree or state machine, might had resulted in a working opponent AI. The selected time frame was too short to experiment on new topics.

The influence map had also a time problem, but it was not too severe because it happened in the bug fixing phase. A not working software is, regardless of progress, is still of no use. It should work with one more week of development.

In conclusion, if the objectives were not set so ambiguous in the beginning, there might have been a working AI. A reason why the algorithm choice was set on the HTN was, that the ten weeks of the Final Major Project were a great opportunity to experiment with new topics that came to short in the lectures.

The overall project was still not a waste of time. The development of all these aspects have set a good working ground for future work.

The game development gave valuable information about the integration of a control system that should be able to control a very large number of units. Due to time constrains, this part was solved partially with an obnoxious number of lists, which worked well so far but there must be a better way.

The research to the influence map showed that it also is usable for pathfinding because of its ability to show tension or vulnerability in a convenient way.

Most of the research paper about HTN variations demonstrated the superior performance of HTNs over conventional algorithms in planning problems and First-Person Shooter unit behaviour. They gave a good insight about the usage of planning algorithms.

3.3 Future Work

The development of a simple real-time strategy game was completed during this project. Some ground work for an influence map and a Hierarchical Task Network (HTN).

As future work, the game gives a good foundation to develop the HTN to a functional point or to start with a new algorithm. To refine the game structure might also be a good task, since the interface to other software, the AI code for example, is not existing.

4 References

Anon., 2015. *Title Picture* [viewed 15 January 2018]. Available from: https://www.wallpaperup.com/655415/sci-fi_warrior_futuristic_art_artwork.html

Anon., 2015. *HTN Planner for Unity* [viewed 30 April 2018]. Available from: http://stagpoint.com/forums/pages/documentation/

Anon., 2018. *Divide and Conquer Algorithm* [viewed 17 February 2018]. Available from: https://en.wikipedia.org/wiki/Divide_and_conquer_algorithm

Anon., 2018. *Waterfall Model* [viewed 5 February 2018]. Available from: https://en.wikipedia.org/wiki/Waterfall_model

Anon., 2018. *Agile Software Development* [viewed 5 February 2018]. Available from: https://en.wikipedia.org/wiki/Agile_software_development

Anon., 2018. *Feature-Driven Development* [viewed 5 February 2018]. Available from: https://en.wikipedia.org/wiki/Feature-driven_development

Anon., *et al*, 2017. Influence Maps [viewed 15 April 2018]. Available from: https://forum.unity.com/threads/ai-influence-maps.145368/

BELL, R., 2009. ***A beginner's guide to Big O notation*** **[viewed 16 February 2018]. Available from:** https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/

BENNETT, P., 2017a. ***Goal Orientated Behaviour*****. Lecture delivered to Artificial Intelligence for Games, Level 6, n.d.**

BENNETT, P., 2017b. ***Goal Orientated Action Planning*****. Lecture delivered to Artificial Intelligence for Games, Level 6, n.d.**

BENNETT, P., 2017c. ***Monte Carlo Tree Search*****. Lecture delivered to Artificial Intelligence for Games, Level 6, n.d.**

BETHESDA GAME STUDIOS, 2006. ***The Elder Scrolls IV: Oblivion*****. Bethesda Softworks**

BLIZZARD ENTERTAINMENT, 2010. ***StarCraft II*****. Blizzard Entertainment**

BLIZZARD, 2017. ***StarCraft II API Has Arrived*** **[viewed 26 March 2018] Available from:** http://us.battle.net/sc2/en/blog/20944009/the-starcraft-ii-api-has-arrived-8-9-2017

COBB, D., 2018.*Final Major Project*. Lecture delivered to Game Development Project, Level 6, n.d.

GITHUB, 2018. *How Developers Work* [viewed 5 February 2018]. Available from: https://github.com/features#project-management

HANSFORD, B., 2015. *Influence Maps* [viewed 14 April 2018]. Available from: http://darkhorsegames.net/2015/11/23/influence-maps-unity/

MAXIS, 2014. *The Sims 4*. Electronic Arts

MICROSOFT, 2018. *MemberwiseClone* [viewed 05 April 2018]. Available from: https://msdn.microsoft.com/en-us/library/system.object.memberwiseclone(v=vs.110).aspx

SAFADI, F., FONTENEAU, R. and ERNST, D., n.d. *Artificial Intelligence Design for Real-time Strategy Games* [viewed 29 January 2018]. Available from: http://www.montefiore.ulg.ac.be/~fsafadi/nips2011.pdf

STAINLESS STEEL STUDIOS, 2001. *Empire Earth II*. Sierra Entertainment

5 Appendix

5.1 Appendix A - Abstraction Layer

It describes the way Humans deal with the complexity of the game: “Humans deal with the complexity using several abstraction layers, taking decisions on different abstract levels” (*F. Safadi, R. Fonteneau and D. Ernst n.d., p. 1*).

Artificial Intelligence (AI) always has resembled human behaviour to solve given problems. The simplest example would be the decision tree. It is a good example of the thought process visualised in code. Its fundamental flaw is the representation of structure once the thought process gets larger and larger. In terms of real-time strategy games, the consideration of every single unit would end in a code structure too large to maintain.

To reduce the scope of the situation helps to reduce the amount of code that is necessary to solve the situation. This is the concept of abstraction.

5.2 Appendix B - Software

5.2.1 Unity

Unity is an editor for game development. Created games run on the Unity Engine. It is a beginner friendly and Component based editor. Games can easily be converted to many other platforms. Supports 2D and 3D games.

Advantages	Disadvantages
previous knowledge	no multicore support
free for non-commercial projects	no C++ support
includes Visual Studio	
C# support	

5.2.2 Unreal

Unreal is an editor for game development. Games created in Unreal run on the Unreal Engine. The editor includes many features for a faster development that are mainly used by designer. Better for 3D games. Supported platforms are only PC and Console.

Advantages	Disadvantages
blueprint development	no previous knowledge
free for non-commercial projects	bad documentation
C++ support	

5.2.3 Visual Studio

Visual Studio is an integrated development environment. It features a debugger and a version control system. It supports a large amount of languages, with C++, C# and JavaScript among them. It is free of charge for Students.

Advantages	Disadvantages
version control	
intelligent code completion	
language support	
provided through university	

5.2.4 SourceTree

SourceTree is a program to manage git repositories. Unlike the git console it has a graphical user interface. It can still open the console if it is necessary. It offers many functions that, otherwise, must be installed separately.

Advantages	Disadvantages
graphical user interface	
fast development process	
clear version overview	

5.3 Appendix C - Languages

5.3.1 C++

C++ is an advanced high-level programming language. Unlike other high-level languages it provides low-level memory manipulation. Its focus on performance is one of the reasons why it is used in game development.

Advantages	Disadvantages
memory manipulation	poor standard library

5.3.2 C#

C# is a programming language that is mostly used in .NET programming. It has a strong focus on networking and therefore it is a good choice when multithreading is an essential part of the development.

Advantages	Disadvantages
rich standard library strong multicore support	

5.3.3 JavaScript

JavaScript is a programming language that is used in web applications and development. Unity also support JavaScript alongside with C#. It is a weakly typed programming language and therefore it is more likely to produce errors.

Advantages	Disadvantages
	weakly typed language

5.4 Appendix D - Services

5.4.1 Git

Git is a version control system. There are many providers that offer free hosting of git repositories. GitHub and GitLab are two of them. The functionality is the same and the only difference is how your repository is managed.

Advantages	Disadvantages
backup of files versions of projects	accessible data

5.4.2 Trello

Trello is a collaboration tool. It is mostly used to show information about who is working on what, what is done and what is pending. It also features a burndown chart. Trello is available on the web or as app and it is free for basic use.

Advantages	Disadvantages
visualized work progress visualized work performance	

5.5 Appendix E - Methodologies

5.5.1 Waterfall

The Waterfall model is a software development methodology. Its structure is like a waterfall. It consists of five milestones that are worked down one by one: Requirements, Design, Implementation, Verification and Maintenance. Waterfall is an easy methodology and is mostly used for smaller project without uncertain requirements. A predictive methodology often comes with a Task List, a Work Breakdown Structure and a Gantt Chart. (*Anon. 2018*).

Advantages	Disadvantages
simple framework	cannot deal with uncertainties
suitable for smaller projects	late working software

5.5.2 Feature-Driven Development

The Feature-Driven Development methodology is an Agile framework and consists of five short-iteration milestones. These are the development of the overall model, build a feature list, plan by feature, design by feature and build by feature. Agile frameworks are adaptive management methodologies and often comes with a Backlog, Iteration Planning, Retrospective and a Burndown Chart. (*Anon. 2018*).

Advantages	Disadvantages
suitable for larger projects	attempting to take too much work
can deal with uncertainties	
early working software	

5.6 Appendix F – Algorithm Comparison

Goal Orientated Action Planning (GOAP)

GOAP is an algorithm that is using pathfinding algorithms, mostly A*, to find actions that are contributing to the goal. It is very performance heavy but can plan and can come up with sometimes unusual behaviour (*Mark Bennett 2017, Level 6*).

Advantages	Disadvantages
capable of planning	bad performance
easy to expand	long development

Decision Tree

The decision tree is a simple algorithm that traverses a tree toe let a bool statement decide with of two branches it will follow. At the end of the tree is an action that will be performed. This algorithm is very reactive to the world state but cannot plan actions to satisfy a goal.

Advantages	Disadvantages
easy to develop	only reactive behaviour
bad scalability	predictable behaviour
good performance	

Hierarchical Task Network (HTN)

The HTN is an algorithm like the GOAP that builds a plan from available actions. The GOAP has a goal that it tries to satisfy and uses a pathfinding algorithm to select possible actions. The difference is that HTN uses a tree search for this. This has the advantage, if a branch cannot satisfy the goal, the whole branch is cut, so that subbranches are not considered anymore. This gives a performance advantage over GOAP (*Anon., 2015*).

Advantages	**Disadvantages**
capable of planning	long development
good performance	
good scalability	

State Machine

State Machines are a very simple method to express actions which can be replaced by other actions if some event in the game triggers. This algorithm is currently used to control the unit's behaviour. Citizens that deliver their resources to the nearest building if they cannot carry anymore. This help a lot to reduce the interaction between player and game. This algorithm could be used to give the player the same methodology to interact with the game, but this would be to simple for a project of this size. This algorithm also cannot plan actions to satisfy goals in the game.

Advantages	**Disadvantages**
Easy to develop	predicable behaviour
medium scalability	short development
good performance	

5.7 Appendix G - Literature Review

SAFADI, F., FONTENEAU, R. and ERNST, D., n.d. *Artificial Intelligence Design for Real-time Strategy Games* [viewed 29 January 2018]. Available from:
http://www.montefiore.ulg.ac.be/~fsafadi/nips2011.pdf

This paper discusses an approach to an Artificial Intelligence (AI) for StarCraft II (*Blizzard Entertainment 2010*). It uses a design that combines abstraction, modularity and hierarchy into a single model. Its results are showing that, whatever algorithm they were using, it was sometimes capable of defeating the StarCraft II AI without losing any units.

HANSSON, N., 2010. *Auto Balancing Decision Trees Aka Resource Trees I* [viewed 31 January 2018]. Available from:
http://gameschoolgems.blogspot.co.uk/2010/05/auto-balancing-decision-trees-aka.html

This Article is the first part about Auto Balancing Decision Trees Aka Resource Trees and discusses how they are working for real-time strategy games. Resource Trees are a way to implement goal orientated behaviour with a decision tree. However, this technique has the problem that it can only satisfy one node of the tree (every child node of it) at a time.

HANSSON, N., 2010. *Auto Balancing Decision Trees Aka Resource Trees II* [viewed 31 January 2018]. Available from:
http://gameschoolgems.blogspot.co.uk/2010/05/auto-balancing-decision-trees-aka_18.html

This Article is the second part about Auto Balancing Decision Trees Aka Resource Trees and discusses how they are working for real-time strategy games. Resource Trees are a way to implement goal orientated behaviour with a decision tree. However, this technique has the problem that it can only satisfy one node of the tree (every child node of it) at a time.

ANON., 2012. *Hierarchical Task Network (HTN) Planning* [viewed 03 April 2018]. Available from: http://pages.mtu.edu/~nilufer/classes/cs5811/2012-fall/lecture-slides/cs5811-ch11b-htn.pdf

This presentation shows a brief overview about Hierarchical Task Networks and how they are structured and why their tree structure has a performance advantage over pathfinding planner algorithms. A small example of a task decomposition is also displayed.

Kelly, J., Botea, A., Koenig, S., n.d. *Planning with Hierarchical Task Networks in Video Games* [viewed 05 April 2018]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.4477&rep=rep1&type=pdf

This paper discusses how Hierarchical Task Networks (HTNs) are used in Computer Games and sometimes combined with other algorithms, like in The Elder Scrolls IV: Oblivion (*Bethesda Game Studios 2006*). It gives a very detailed overview about the implementation for Non-Playable Characters (NPCs). This papers work focuses more on using HTNs to implement scripts as character behaviour.

J. CHAMPANDARD, A., 2011. *The Mechanics of Influence Mapping: Representation, Algorithm & Parameters* [viewed 30 January 2018]

Available from: http://aigamedev.com/open/tutorial/influence-map-mechanics/

An article about the power of influence maps. It shows several benefits and disadvantages of influence maps and provides a starting point of the algorithm that is necessary to program an influence map.

HANSSON, N., 2010. *Influence Maps I* [viewed 30 January 2018]. Available from: http://gameschoolgems.blogspot.co.uk/2009/12/influence-maps-i.html

Niklas Hansson describes in his 2-parted blog, what an influence map is and shows in detailed and simplified examples of how they work in real-time strategy games. The only downside is the lack of code.

5.7 Appendix H - Additional Literature

TEICH, T. and L. DAVIS, I., n.d. *AI Wall Building in Empire Earth II* [viewed 27 January 2018]. Available from: https://www.aaai.org/Papers/AIIDE/2006/AIIDE06-030.pdf

This paper explains how the Empire Earth II team for Artificial Intelligence (AI) solved the problem of efficient wall building. It shows how complex a simple thought can be. Ultimately this was the spark that brought the idea of abstraction to the current AI design.