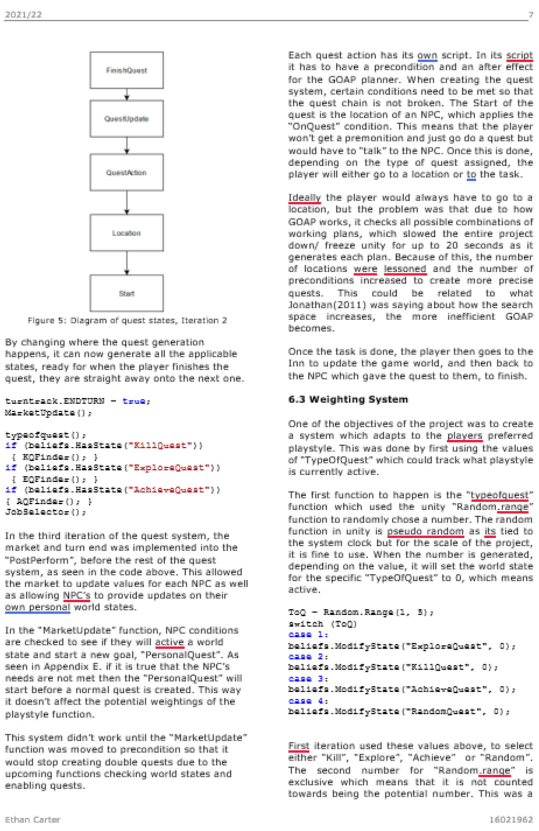
Comprehensive Creative Technologies Project: Procedurally Generated Quest System Using the Players Preferred Playstyle, World Simulation and G.O.A.P.

Ethan Carter
Ethan2.Carter@live.uwe.ac.uk
Supervisor: Luke Child

Department of Computer Science and Creative Technology
University of the West of England
Coldharbour Lane
Bristol BS16 1QY



2021/22 7

FinishQuest
QuestUpdate
QuestAction
Location
Start

Figure 5: Diagram of quest states, Iteration 2

By changing where the quest generation happens, it can now generate all the applicable states, ready for when the player finishes the quest, they are straight away onto the next one.

```
turntrack.ENDTURN = true;
MarketUpdate();

typeofquest();
if (beliefs.HasState("KillQuest"))
 { KQFinder(); }
if (beliefs.HasState("ExploreQuest"))
 { EQFinder(); }
if (beliefs.HasState("AchieveQuest"))
{ AQFinder(); }
JobSelector();
```

In the third iteration of the quest system, the market and turn end was implemented into the "PostPerform", before the rest of the quest system, as seen in the code above. This allowed the market to update values for each NPC as well as allowing NPC's to provide updates on their own personal world states.

In the "MarketUpdate" function, NPC conditions are checked to see if they will active a world state and start a new goal, "PersonalQuest". As seen in Appendix E. if it is true that the NPC's needs are not met then the "PersonalQuest" will start before a normal quest is created. This way it doesn't affect the potential weightings of the playstyle function.

This system didn't work until the "MarketUpdate" function was moved to precondition so that it would stop creating double quests due to the upcoming functions checking world states and enabling quests.

Each quest action has its own script. In its script it has to have a precondition and an after effect for the GOAP planner. When creating the quest system, certain conditions need to be met so that the quest chain is not broken. The Start of the quest is the location of an NPC, which applies the "OnQuest" condition. This means that the player won't get a premonition and just go do a quest but would have to "talk" to the NPC. Once this is done, depending on the type of quest assigned, the player will either go to a location or to the task.

Ideally the player would always have to go to a location, but the problem was that due to how GOAP works, it checks all possible combinations of working plans, which slowed the entire project down/ freeze unity for up to 20 seconds as it generates each plan. Because of this, the number of locations were lessoned and the number of preconditions increased to create more precise quests. This could be related to what Jonathan(2011) was saying about how the search space increases, the more inefficient GOAP becomes.

Once the task is done, the player then goes to the Inn to update the game world, and then back to the NPC which gave the quest to them, to finish.

6.3 Weighting System

One of the objectives of the project was to create a system which adapts to the players preferred playstyle. This was done by first using the values of "TypeOfQuest" which could track what playstyle is currently active.

The first function to happen is the "typeofquest" function which used the unity "Random.range" function to randomly chose a number. The random function in unity is pseudo random as its tied to the system clock but for the scale of the project, it is fine to use. When the number is generated, depending on the value, it will set the world state for the specific "TypeOfQuest" to 0, which means active.

```
ToQ = Random.Range(1, 5);
switch (ToQ)
case 1:
beliefs.ModifyState("ExploreQuest", 0);
case 2:
beliefs.ModifyState("KillQuest", 0);
case 3:
beliefs.ModifyState("AchieveQuest", 0);
case 4:
beliefs.ModifyState("RandomQuest", 0);
```

First iteration used these values above, to select either "Kill", "Explore", "Achieve" or "Random". The second number for "Random.range" is exclusive which means that it is not counted towards being the potential number. This was a

Ethan Carter 16021962

Abstract

This project is a procedural quest system that uses G.O.A.P. to create quests for the player using their preferred playstyle and an economy simulation that can affect world states. It aims to provide a system which can be used to generate quests that are targeted toward the player. The project is aimed at people interested in procedural content with a slight emphasis on emergent narrative.

Keywords: Simulation, Adaptive, Procedural Generation, G.O.A.P.

Brief biography

This project interests me because of the procedural story telling aspects of video gaming. Video games which have these types of systems offer an endless amount of enjoyment for me and to be able to try and replicate it for possible future use in my own game. This project helped me understand the difficulties of creating something procedural and balanced aswell as the task of creating an economy.

How to access the project

Video: https://www.youtube.com/watch?v=Bse7ZB8SU6E

Project URL: https://github.com/Ethan2CarterUWE/CTTP

Load project up and press the play button in unity.

Alternatively, open the "Build" folder and run the exe.

There are no controls as it is all AI-based, to demonstrate the system. Top left has current action + current target. Underneath that are 3 different values that record how much of each quest is completed.

All code is within the scripts folder. All code within the G.O.A.P. folder is from a tutorial by Penny de Byl. Every script outside of that folder is created by me, Ethan Carter. The main bulk of the code is within the "QuestGeneration" folder, and the script is called "QUESTSTATE".

This project is about generating quests for a player, depending on its various world states, the player's preferred playstyle and a simple economic simulation which can change world states. The world states will constantly be changing to create an interesting foundation for the quest generation to occur and use the research of playstyles to influence the quest generation to favour more enjoyable objectives than what they are playing.

This project hopes to show an example of the practicality of how to create a quest generator by using a G.O.A.P. system created by Penny de Byl. Hopefully, this will be able to help or inspire other people who are interested in this type of system as there are not many, if any, examples of a procedural quest generator.

The reason for focusing on this project was a personal enjoyment of video games where the world is dynamic; random events can occur, which helps tell a story. Rim world is a big inspiration in this regard as you can choose different storytellers to provide different procedural experiences.

Key Deliverables

- Unity build that showcases a player doing procedural quests.

Project Objectives

- A system to create quests
- A system that adapts to the players preferred playstyle which influences the generation of quests.
- A market economy that interacts with N.P.C. agents.

2. Literature review and Research findings

(Daniel C, 2011) states that generating content in a complex video game is time-consuming and costly, as the number of assets required and the detail needed is vast, requiring a team to create these. Daniel also says that procedural generation could help reduce the time and costs required to do these tasks. His research suggests that procedural generation is the next level in game development as it produces quicker results at less cost.

<u>Quest Generator</u>

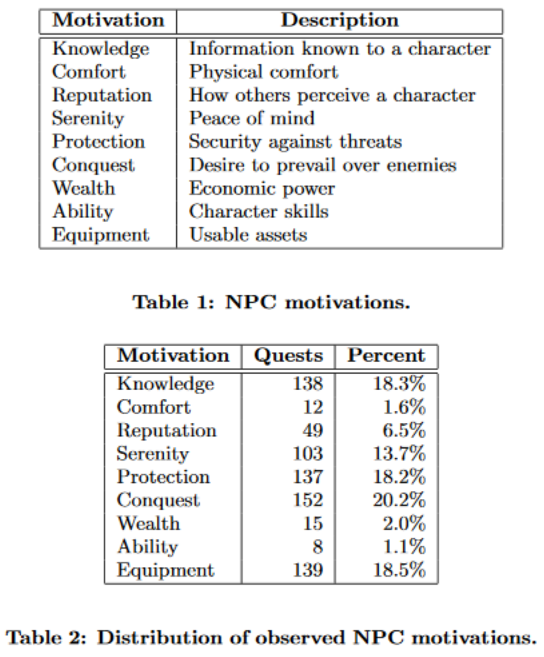
Quests in video games involve doing a particular activity to do a goal.
These activities can range from the following:

- Kill something
- Find something
- Go somewhere
- Talk to someone
- Collect something

(Jonathan D, 2011) explains that from a player's perspective, a quest is a narrative element that offers information about the world as well as gives the player power and knowledge. This information means that quests are an essential part of a role-playing game as it offers players a way to experience the world, the story, and get stronger. When creating a quest generator, these quests will be used as the foundation of the system for the player to experience.

(Jonathan D, 2011) mentions how quest generation requires planning, as it requires an initial state, a goal and then a set of actions that the player must do to finish the quest. This suggests that when creating a system to generate quests, three variables are needed which check the current state, goal and actions required.

Jonathan then mentions how 3000 quests from various games were examined and that each quest consisted of dialogue with an N.P.C. (Non-Player Character), the actions required by the player to complete the quest and the reward for finishing the quest. This seems to be the standard of how a quest is procured for the player and that a primary user interface must be created so the player understands whom they are talking to, what the objectives are the potential rewards.



Motivation	Description
Knowledge	Information known to a character
Comfort	Physical comfort
Reputation	How others perceive a character
Serenity	Peace of mind
Protection	Security against threats
Conquest	Desire to prevail over enemies
Wealth	Economic power
Ability	Character skills
Equipment	Usable assets

Table 1: NPC motivations.

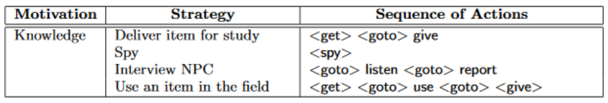
Motivation	Quests	Percent
Knowledge	138	18.3%
Comfort	12	1.6%
Reputation	49	6.5%
Serenity	103	13.7%
Protection	137	18.2%
Conquest	152	20.2%
Wealth	15	2.0%
Ability	8	1.1%
Equipment	139	18.5%

Table 2: Distribution of observed NPC motivations.

Figure 1: two tables about NPC motivation in Quests *(Jonathan D, 2011)*

Jonathan's research into quests has helped him categorise the quests underlying motivations for an N.P.C., as seen in Figure 1. Having motivations for an N.P.C. will help produce more engaging quests for the player because as they change the world state by doing quests for certain people, changes can happen which can change the quests and quest content produced. (Vincent B, 2018) also looks at the same system and proposes the creation of the C.O.N.A.N. system which follows the same research as (Jonathan D, 2011)

(Jonathan D, 2011) proceeds to then describe how when an N.P.C. generates a quest, it uses a specific strategy, as shown in figure 2.

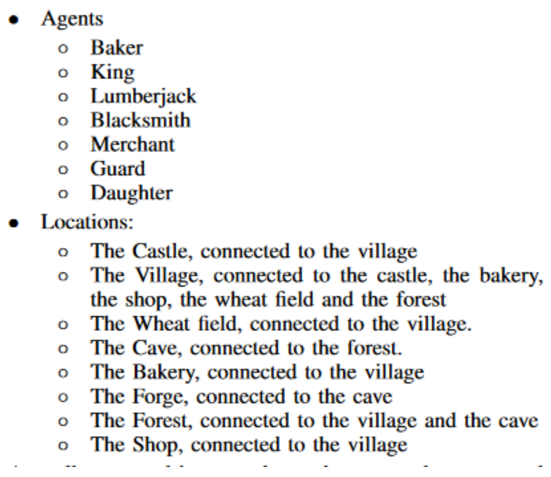


Motivation	Strategy	Sequence of Actions
Knowledge	Deliver item for study	<get> <goto> give
	Spy	<spy>
	Interview NPC	<goto> listen <goto> report
	Use an item in the field	<get> <goto> use <goto> <give>

Figure 2: *(Jonathan D, 2011)* Strategy and sequence of actions for specific motivation

Once a strategy is selected, the corresponding sequence of actions is then implemented, which are the goals/instructions for the player on what to do.

(Vincent B, 2018) implement a world state which uses agents (N.P.C.s) as quest givers or targets in a sequence of actions and locations which can store the quest givers, items, enemies, or quest objectives. The world state can be expanded and offers a more complex world for the player to explore.



- Agents
  - Baker
  - King
  - Lumberjack
  - Blacksmith
  - Merchant
  - Guard
  - Daughter
- Locations:
  - The Castle, connected to the village
  - The Village, connected to the castle, the bakery, the shop, the wheat field and the forest
  - The Wheat field, connected to the village.
  - The Cave, connected to the forest.
  - The Bakery, connected to the village
  - The Forge, connected to the cave
  - The Forest, connected to the village and the cave
  - The Shop, connected to the village

Figure 3: *(Vincent B, 2018)* World State manager of Agents and Locations

Once the agents have been generated with roles, such as a baker, they need to have certain preferences towards specific quests. (Vincent B, 2018) explains how they created a "total cost" for the agent's preference and the action of kill costs more for certain agents, such as the baker than the knight. For each agent, a preference list is created, which increases the preference for specific actions depending on the job. These preferences do not change the quest's outcome but change how the quest givers may approach to handling giving the player the task. For example, a beggar may offer a quest to steal bread from the baker, whereas a noble may offer a quest to purchase bread from the baker and give it to the beggar. This will create a semblance of personality for each agent in the game.

Vincent then proceeds to go on about how there are two algorithms which choose goals for the agent. The first algorithm will select a random goal to use as a baseline for each agent. Then the second algorithm will select another set of random goals but rank them in order depending on the agent's preferences then keep the best fitting one which is the goal. This is achieved by creating a list of goal states for each agent and then using a plan to reach the goal (Vincent B, 2018).

Player Types as a basis for the quest

To create a good player experience with a procedural quest system, a system needs to be created which figures out what type of playstyle the player enjoys and combine it with the quest system so that it will generate more quests in tune with their playstyle. (Bartle, 1996) created a model of player types for a game called M.U.D., after several hundred bulletin board postings, from other players; explaining likes, dislikes, why they play and what they want to see improved. He analysed the data and discovered that there were four main play styles. These styles are

- Killers (involve player vs player gameplay, go against other humans)
- Achievers (enjoys gameplay challenges)
- Explorers (explore the world and the game mechanics)
- Socialisers (enjoy talking to other humans/players)

However, Bartle's research is outdated and used out of context as this was specifically targeted towards the game M.U.D. (Dixon, 2011) has made further efforts to gather knowledge on player types by finding more up to date sources. (Dixon, 2011) cites the work of (Yee, 2005) who has updated the Bartle model with a more modern approach.

(Dixon, 2011) summarises (Yee, 2005) by listing the new three main components and ten subcomponents.

- "Achievement: Advancement, Mechanics, Competition"
- "Social: Socialising, Relationship, Teamwork"
- "Immersion: Discovery, Role-playing, Customization, Escapism"

(Dixon, 2011) also points out that the killer playstyle has been removed and instead it is found in correlation with players with the competition subcomponent. This means that players who enjoy a form of competition may also enjoy being a "killer" compared to (Bartle, 1996) research as killer was just its own category.

G.O.A.P.

G.O.A.P. stands for Goal Orientated Action Planning. Orkin (2006) describes the differences between goal-orientated action planning and finite state machines. He begins by explaining how finite state machines are strict and that they will always follow a specific set of actions, whereas goal-orientated action planning starts at the end, with the goal first. This means that once the goal has been set, a plan is then created to list a set of actions to achieve that goal. G.O.A.P. also has a cost attached to each possible action. This means that each action can be modified to have a different cost, allowing certain plans to be acted on. This could be useful when making a quest generator as the motivations of the quest giver could affect the cost of certain actions allowing "more realistic" quests.

World Simulation

Adams (2015) explains the simulation principles of Dwarf Fortress, a game known for its complexity and world simulation. He explains that you shouldn't over plan the model and that you should just keep building it up over time. This means that it's more likely to get the results required instead of creating something which does not line up with the goal.

He also states that the system should not be overcomplicated and that the world simulation should operate on a level that the player sees or one level below. This is to stop creating variables and other systems that might have an insignificant impact on the game, so stick to creating something that is useful for the quest generator.

3. Research questions 250w

After surveying relevant information for the project, a few questions have emerged.

- How would an economic simulation work that is balanced?
- In what ways can you keep track of a player's preferred playstyle
- Which method of weighting is more suitable?
- Is G.O.A.P. a viable candidate for a procedural quest system?

4. Research methods

The main research problem was how to create a system that procedurally generates quests which means that the research on this was primarily used to gain an understanding on potential methods.

Another research problem was finding out what sort of playstyles are there for people that play video games and how they can categorise players. Research for this was used to find out about different variables/characteristics of those playstyles that could be used to identify the player, to be used for quest generation.

The last research problem was looking at how to create a system of a living environment for games. The research here will provide information on what is required to create a baseline living world so that it can then interact with the quest system.

All research was done by using qualitative data of secondary resources, by people who have experienced either creating the systems or researched player types.
To collect the research, google scholar was used as the search engine and specific search terms to identify research such as "procedural quest generator" or "player types". The types of papers were created by people who have an interest or work in the video game industry, this is because the system being designed is for a game, so the information needs to be applicable. Some of these journals also had pictures of examples, such as an image with the system design of a quest generator. For the quest generator, a recurring idea was using a method called G.O.A.P., to create the quest generator.

A limitation of this research was that the topic of a procedural quest generator is not common, so there is only a limited number of resources. However, the information available is detailed and helpful. However, for creating a living world in video games, information is very limited to almost non-existent.

5. Ethical and professional principles

With the focus of this report on systems and algorithms relating to procedurally generated content, there is no human participation. However, this report's research could impact people with jobs regarding quest implementation. This could happen by using the systems created to help them do their job by automating a lot of the work as they only have to modify variables. This could also lead to a loss of jobs as people who are not specialised into those specialised work fields to create the same thing, thus reducing costs, or changing the requirements of their job to learn and adapt to the new software.

6. Practice

6.1 G.O.A.P.

At the beginning of the project, research was conducted on various aspects of quest generation and how to do it. Jonathan (2011) mentioned how a planner would be inefficient with creating a quest system as the search space grows but mentioned that it is suitable for the task. Vincent (2018) talked about using a planner to create a system to generate procedural quests. This meant that a planner was going to be required for a system to do with quest generation. The initial idea was a planner known as G.O.A.P. (Goal orientated Action Planning). G.O.A.P. uses regressive search to create a plan by starting at the desired goal and working its way back to actions to find the most suitable way of making a plan. Regressive search is good for a system where the goal is to complete certain quests, as it can limit the number of actions required if multiple world states of the same value are active.

The first iteration of G.O.A.P. implementation ended up not being G.O.A.P. at all. Instead of an actual planner, the fake version of G.O.A.P. searched through every action on the completion of finding a state for a string of its preconditions. This would not only take a while to run due to the amount of code being repeated, but the 'plan' was never actually correct. If an object had multiple preconditions, it would look for the first matching condition and set it as the active one.

The second iteration of G.O.A.P. implementation also ended was not G.O.A.P. but instead was a script which checked if the precondition of actions was the same as the aftereffects of actions. Again, this created some sort of 'plan', but again it searched through every action available for everything.

After that, the decision was to use a G.O.A.P. framework that already works, as, without it, this project is unable to proceed. Penny de Byl (2021) created a unity tutorial called "The G.O.A.P. Planner", which is a working G.O.A.P. system. After following the tutorial and implementing her system, work could begin on creating the actual system to create procedural quests.

6.2 Quest System

Once G.O.A.P. was implemented, work on the Quest System began. Quests in video games normally involve talking to an N.P.C., performing the required task, and then finishing the quest.

These quests are simple as they have a strict set of objectives for the player to do with little room for multiple choice. Using research from Bartle(1996) and Daniel (2011) on player types, the conclusion was that there are three types of play styles. This is important information as it will help categorise quests so that specific quests can be activated when needed. These types of conditions will be known as "TypeOfQuest".

Quests also have locations where the quest takes place or where the objective may be. This means that locations must be a part of the world states to see if the player is currently at the right place. Without this category, the player might be able to complete the quest without going to the area, and if the system had quest givers who give dialogue, this could break the immersion.

The goal of the player is to finish the quest, and the end world state is the goal that G.O.A.P. is trying to achieve.

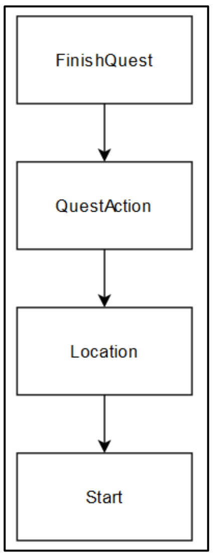


Figure 4: Diagram of quest states, Iteration 1

Figure 4 shows the example of the first iteration of G.O.A.P. solving a quest loop. When the player completes the quest(Arrives at the FinishQuest stage), the loop must restart so that the new quest can be activated.

```
typeofquest();
if (beliefs.HasState("KillQuest"))
{ KQFinder(); }
if (beliefs.HasState("ExploreQuest"))
{ EQFinder(); }
if (beliefs.HasState("AchieveQuest"))
{ AQFinder(); }
JobSelector();
```

The "FinishQuest" stage, as seen in figure 4, is where the quest generation happens, as seen in the code above. The quest generation happens in a child of the class "GAction", as it provides access to the "PrePerform" and "PostPerform" functions, which affect the planner, as well as access to easy access of world states to modify. First, it checks the "TypeOfQuest" that the player will be on. Then depending on that, it will assign an action for the quest. For example, if the "TypeOfQuest" is "Kill", then the "KQfinder" function activates where it will find what the player should kill. Once that is done, the job selector function activates, determining which N.P.C. is giving the quest.

```
if (beliefs.HasState("JobKing"))
   {
     beliefs.ModifyState("JobKing", -1);
   }
```

```
if (beliefs.HasState("ExploreQuest"))
    {
beliefs.RemoveState("ExploreQuest");
    }
```

On the first iteration ,the quest system would work for the static quest and then it would break, due to certain conditions not being turned off. To fix this, in the "PrePerform" of the "FinishQuest" stage 2 functions are added. These functions remove the world states for the active N.P.C. who has given a quest, and the world state for which ToQ(type of quest) is also active as seen in the code above. This allows for attribution of new states in the "PostPerform".

This change fixed those issues however a new quest was not being generated. The second iteration solved this issue by adding new step adding into the diagram , as seen in figure 5, which now does the quest generation in the "QuestUpdate" stage.

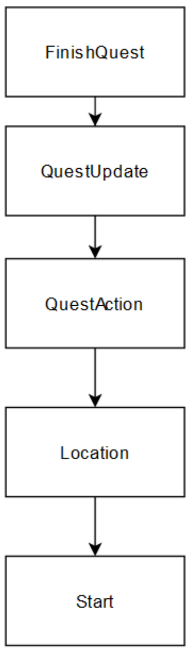


Figure 5: Diagram of quest states, Iteration 2

By changing where the quest generation happens, it can now generate all the applicable states, ready for when the player finishes the quest, they are straight away onto the next one.

```
turntrack.ENDTURN = true;
MarketUpdate();

typeofquest();
if (beliefs.HasState("KillQuest"))
 { KQFinder(); }
if (beliefs.HasState("ExploreQuest"))
 { EQFinder(); }
if (beliefs.HasState("AchieveQuest"))
{ AQFinder(); }
JobSelector();
```

In the third iteration of the quest system, the market and turn end was implemented into the "PostPerform", before the rest of the quest system, as seen in the code above. This allowed the market to update values for each N.P.C. as well as allowing N.P.C.'s to provide updates on their world states.

In the "MarketUpdate" function, N.P.C. conditions are checked to see if they will activate a world state and start a new goal, "PersonalQuest". As seen in Appendix E., if it is true that the N.P.C.'s needs are not met, then the "PersonalQuest" will start before a standard quest is created. This way, it does not affect the potential weightings of the playstyle function.

This system did not work until the "MarketUpdate" function was moved to precondition so that it would stop creating double quests due to the upcoming functions checking world states and enabling quests.

Each quest action has its own script. In its script it has to have a precondition and an after effect for the G.O.A.P. planner. When creating the quest system, certain conditions need to be met so that the quest chain is not broken. The Start of the quest is the location of an N.P.C., which applies the "OnQuest" condition. This means that the player will not get a premonition and just go do a quest but would have to "talk" to the N.P.C. Once this is done, the player will either go to a location or to the task depending on the type of quest assigned.

Ideally, the player would always have to go to a location. However, the problem was that due to how G.O.A.P. works, it checks all possible combinations of working plans, which slowed the entire project down/ freeze unity for up to 20 seconds as it generated each plan. Because of this, the number of locations was lessened, and the number of preconditions increased to create more specific quests. This could be related to what Jonathan(2011) said about how the search space increases, the more inefficient G.O.A.P. becomes.

Once the task is done, the player then goes to the Inn to update the game world and then back to the N.P.C., which gave the quest to them to finish.

6.3 Weighting System

One of the project's objectives was to create a system that adapts to the player's preferred playstyle. This was done by first using the values of "TypeOfQuest" which could track what playstyle is currently active.

The first function to happen is the "typeofquest" function which used the unity "Random.range" function to randomly chose a number. The random function in unity is pseudo-random as it's tied to the system clock, but for the scale of the project, it is fine to use. When the number is generated, depending on the value, it will set the world state for the specific "TypeOfQuest" to 0, which means active.

```
ToQ = Random.Range(1, 5);
switch (ToQ)
case 1:
beliefs.ModifyState("ExploreQuest", 0);
case 2:
beliefs.ModifyState("KillQuest", 0);
case 3:
beliefs.ModifyState("AchieveQuest", 0);
case 4:
beliefs.ModifyState("RandomQuest", 0);
```

The first iteration used these values above to select either "Kill", "Explore", "Achieve", or "Random". The second number for "Random.range" is exclusive, which means that it is not counted towards being the potential number. This was a start as it allowed a random chance to receive a quest.

```
ToQ = Random.Range(1, 101);
if(ToQ <= 30)
else if(ToQ <= 60)
else if (ToQ<=90)
else
```

The second iteration was to expand upon the simple values of "Random.Range" and include a weighting system. Above in the code is an example of what it currently looks like. In addition, ToQ can now be within a range of specific if statements which allows preference to be created if a designer was to use this.

```
comb1 =ExploreQmax  + KillQmax;
comb2 = comb1 + AchieveQmax;
```

The third iteration further expanded upon the previous attempt by adding a counter for each quest which is then used to adjust the weightings. Comb1 combines the amount of the first 2 types of quests, whereas comb2 combines comb1 with the third type of quest.

```
ToQ = Random.Range(1, totalQmax + 1);
```

The ToQ, ranges between 1 and the total complete quests + 1 because it is exclusive.

```
if (ToQ <= ExploreQmax)
else if (ToQ > ExploreQmax && ToQ <= comb1)
else if (ToQ > comb1 && ToQ <= comb2)
```

Once the values are totalled, the function checks to see which value it landed at to determine which type of quest is assigned. The problem with this system is that because there is no max value for the weighting if a player only completed 30 Kill quests, they would only have a 3.13% chance of receiving the other types of quest, which not only would be very rare at the point but almost statistically impossible to recover the weighting.

```
int[]actualValues = {1,1,1,2,2,2,3,3,3,0};
```

For the final iteration of the weighting system, an array was used as seen above. The array contains the initialising values of the array. In the "playstyleAIWeighting" function, as seen in appendix F, in a for loop the array shuffles all array values down by 1, and checks to see what the contents of the array member is. If the array value matches the indicated number and the corresponding quest number is not more than 6, it will increase it, which increases the weighting for that type of quest. In appendix G, the type of quest function then sees what the value of the random number is and modifies the last member of the array with a value of its own quest type. At the end of the function, the values of the "firstnumber", "Secondnumber" and "Thirdnumber" are reset back to 2. This is so that there is always a weighting for these types of , and in the "playstyleAIWeighting" function, there is a cap on how high each value can go so that there will never be 100% weightings.

6.4 Economy/LivingWorld

The initial idea was to create a marketplace where each N.P.C. can go so that they can purchase resources they specifically need and sell resources that they produce. This was the plan because depending on their access to these resources, the world state would change, and personalised quests would be produced to help restore balance. Appendix H has a diagram of N.P.C.s needs and desires.

However, upon creating the market, the N.P.C.s would be unhappy all the time due to supply chain issues. To try and solve this, the values of each N.P.C. were adjusted to be safer so that they would not be unhappy/hungry as often. This caused them never to be upset and it would never change the world state.

To deal with the constant unsatisfying values, they were given a high value of resources and have significantly increased output so they never get annoyed. This of course means that the world states are never changed due to the consequences of the N.P.C.s, and as such removed all code to do with messing with N.P.C. values from quest actions, such as bears increasing hunter fur output but lowering farmer food output.

7. Discussion of outcomes

The completed project could be used as a learning tool for people interested in Procedural quests/storytelling. This is because while procedural quest systems are rare to find, one that uses G.O.A.P. is almost non-existent. This provides a learning opportunity on how someone could possibly expand upon the project's idea. Other professionals may be able to understand why there is little to nonpractical examples on creating a system like this as the system is complicated to create.

This project had mixed outcomes regarding success. An own made version of G.O.A.P. failed to work but was slightly remedied by use of an already created version, from Penny de Byl(2021). The market also failed due to the vast underestimation of the difficulty it would be to not only build an economy which was balanced, but to provide enough chaos to create issues that would not overtake the main quest system. If the N.P.C.'s face a supply chain issue, the entire world is in jeopardy, but there is nothing to do if no one has a problem. However, the economy is a good idea but maybe try to do something even more straightforward than what was attempted. Maybe add a cooldown between N.P.C.s who are upset so they do not become annoyed every turn, or give the market a boost after completing a personal quest.

However, the success of the procedural quest generation system is good, and that not only is working, but its correct. This can easily be expanded upon to more complex quest systems.

Another success would be the weighting system used to give the player preferred quests. As seen in table 1, table 2, table 3 and figure 6, there is potential for preferred quests, but the player will always experience an even amount of quests. This is beneficial for the longevity of a game if the player has a variety of different quests. However, this system could be further utilised to provide a permanent increase in weighting if the player should choose, that way they can have even more control over what they want to experience.

Playstyle	values	Weighting %
ExploreNumber	5	45.46%
KillNumber	2	18.18%
AchieveNumber	4	36.36%

Table 1: Random value sample

Playstyle	values	Weighting %
ExploreNumber	4	25%
KillNumber	6	37.5%
AchieveNumber	6	37.5%

Table 2: Full values example

Playstyle	values	Weighting %
ExploreNumber	6	60%
KillNumber	2	20%
AchieveNumber	2	20%

Table 3: Most powerful weightings possible

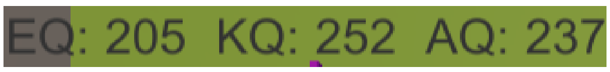

EQ: 205 KQ: 252 AQ: 237

Figure 6: an image of total quests completed after an hour of running.

A good thing with G.O.A.P. is that it did work with what was implemented. The quest system did create procedural quests, although essential, which means that the quest system is correct and that it can be expanded upon further.

A problem with G.O.A.P. was that it could not handle a large number of possible conditions. If there were too many possible solutions, the project would hang. Based on this, G.O.A.P. should not really be used as a planner for this kind of system. This is because if this system were used in a game, there would be many more possible world states, increasing the complexity of the plans required to complete goals. However, this is based on just this system of G.O.A.P. which Penny de Byl created. It is possible that there are better versions of G.O.A.P. but reflecting on what Jonathan(2011) said about it, it seems to be a widespread issue with G.O.A.P.

Another issue with G.O.A.P. is that it can be effortless to break. A misspelt or missing condition can just stop the system outright as it causes the player to be unable to find a plan. When this happens, it is hard to understand where the problem lies, so manual work of checking which action is missing a condition is time-consuming and frustrating.

Additionally, G.O.A.P. may be flexible in creating plans, but it does not allow on the fly changing of the actual currently activated plan. This means that if the player wanted to approach a quest differently than assigned, they would not be able to do so as the plan is set in stone. This takes freedom away from the player which is the opposite of what the system is about.

Using G.O.A.P. to create actions with preconditions and aftereffects is a simple task as it's just adding text to different attributes. However, when trying to make a world with lots of other world states, it can be confusing what states need what conditions. A concept that might help would be to create a visualised form of doing this. For example, Unreal Engine has blueprints which allow for visual scripting. The same could be done with G.O.A.P. The visualisation could look like a bubble graph, and each new action could be dragged and dropped onto previous connections to gain the target's aftereffects as a precondition. This would ensure that not only are all the actions visible in a straightforward manner but that they could be easily modified instead of manually hooking up values.

In hindsight, G.O.A.P. was the wrong way to approach a quest system due to the already listed flaws, but a different planner may provide a better experience.

9. Conclusion and recommendations

To conclude, this project successfully met the goals of creating a procedural quest system and creating a weighting system for the players' preferred playstyle. It struggled with the economy and failed with its own implementation of G.O.A.P.

To further improve upon this work, the quest system could handle more advanced quests, longer with more outcomes. The weighting system could be controlled by the player to have complete control over what per cent of quests they receive. The market could be reduced to a more simple form, with limitations on how often each N.P.C. can be affected by adverse effects. Finally, the G.O.A.P. system could be created and used instead of someone else's planner.

Beyond U.W.E., this project could become a way to help people interested in procedurally generated quests to create their system or inspire them to create something similar.

Because the code uses Penny de Byl's (2021) code which helped complete the project's goals, it should become a creative commons resource to help others who are doing a similar thing.

Overall the project has its failures and successes, but slightly more on the success side.

References

Bartle, R., 1996. Hearts, clubs, diamonds, spades: Players who suit M.U.D.s, s.l.: Richard Bartle. [last access: 23 April 2022]

Byl, P. d., 2021. The G.O.A.P. Planner. [Online] Available at: https://learn.unity.com/tutorial/the-goap-planner#
[Last access 15 March 2022].

Daniel C, F. B. C. P. M. C., 2011. A Survey of Procedural Content Generation Techniques Suitable to Game Development, Salvador: Brazilian Symposium on Games and Digital Entertainment (S.B.G.A.M.E.S.).

Dixon, D., 2011. Player Types and Gamification, Bristol: University of the West of England.
Jonathan D, I. P., 2011. A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs, Denton: University of North Texas.

Vincent B, S. O. J. D., 2018. Let C.O.N.A.N. tell you a story: Procedural quest generation, s.l.: Arxiv.

Yee, N., 2005. Motivations of Play in MMORPGs Results from a Factor Analytic Approach. DiGRA '05 - Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play, Volume 3, pp. 1-46.

Bibliography

Adams, T., 2015. *Simulation Principles from Dwarf Fortress,* s.l.: s.n.

Bartle, R., 1996. *Hearts, clubs, diamonds, spades: Players who suit M.U.D.s,* s.l.: Richard Bartle.

Byl, P. d., 2021. *The G.O.A.P. Planner.* [Online] Available at: https://learn.unity.com/tutorial/the-goap-planner#
[Accessed 15 03 2022].

Daniel C, F. B. C. P. M. C., 2011. *A Survey of Procedural Content Generation Techniques Suitable to Game Development,* Salvador: Brazilian Symposium on Games and Digital Entertainment (S.B.G.A.M.E.S.) .

Dixon, D., 2011. *Player Types and Gamification,* Bristol: University of the West of England.

John G, J. B., (2011). *Procedural Quests: A Focus for Agent Interaction inRole-Playing-Games,* Bath: University of Bath.

Jonathan D, I. P., 2011. *A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs,* Denton: University of North Texas.

Mendonca, S. (2020) DYNAMIC QUEST GENERATION USING IN SESSION PLAYER BEHAVIORS IN MMORPG[online], Masters, Faculty of California Polyrechnic State University Available from: https://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=3647&context=theses [Accessed 24 April 2022]

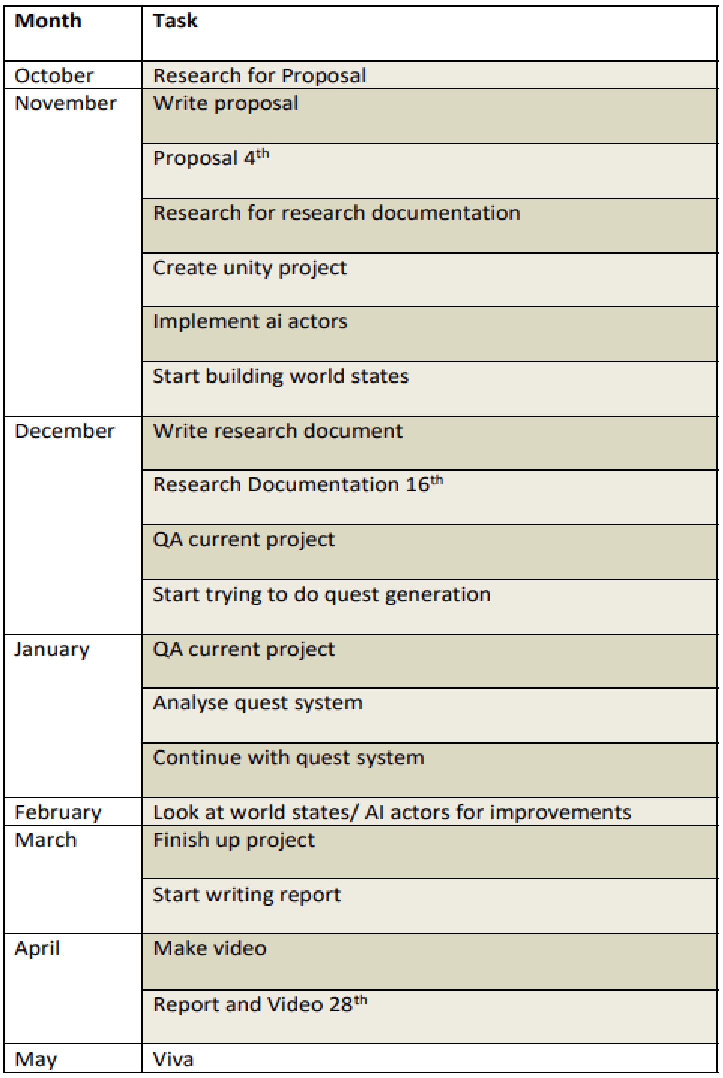
Orkin, J., 2006. *Three States and a Plan: The A.I. of F.E.A.R..* s.l., Game Developers Conference.
R, T., 2015. *Agent-based procedural quest generation for a,* Copenhagen: I.T. University of Copenhagen.
S, M., 2020. *DYNAMIC QUEST GENERATION USING IN.* s.l.:s.n.
Vincent B, S. O. J. D., 2018. *Let C.O.N.A.N. tell you a story: Procedural quest generation,* s.l.: Arxiv.
Yee, N., 2005. Motivations of Play in MMORPGs Results from a Factor Analytic Approach. *DiGRA '05 - Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play,* Volume 3, pp. 1-46.

Appendix A: Project Log (not included in word count)

Date	Task	Description
28/10/2021	Started Proposal outline	I looked at proposal and started to edit it, including new sections and removing old ones.
07/11/2021	Handed in Proposal	Handed in pdf of the proposal
20/11/2021	Created new unity project	Created new project for trying out some new ideas for the dissertation, mainly focusing on the N.P.C. actors.
02/12/2021	Started Research documentation	Edited research documentation and removed documentation of a procedural world and instead replaced it with information and G.O.A.P. and World Simulation. Fleshed out Research methodology.
20/12/2021	Handed in Research report	Handed in research report
05/01/2022	Started building quest generator	Improved upon the npc actors created previously, fleshing out potential procedural elements such as job and mentatlity. Added in a few simpler waypoints to simulate a potential goap system.
24/01/2022	Handed in Demo URL and Done presentation	Handed in Demo URL and presented glimpse of project
11/02/2022	Tried to create a G.O.A.P. system	Started to create a goap system but came across multiple issues regarding creating a plan to get to the goal.
20/02/2022	Prototyped different ways of weighting a quest	
27/02/2022	Restarted project, downgrading from unity 2020 to unity 2019.	Restarted project on an older version of unity due to personal annoyances with 2020, such as constant rebuilding/initialising of scripts when tabbing back to unity.
03/03/2022	Readded npcs and fake goap like thing using waypoints	Caught project back up to where it was before.
14/03/2022	Researched other peoples versions of goap	Researched for potential candidates to use in project.
16/03/2022	Implemented multiple different versions of G.O.A.P.	Tried out 3 different systems of goap.
22/03/2022	Implemented Penny de Byl's version of G.O.A.P. from a unity advanced tutorial	Implemented a version of goap so that the project can get built.
26/03/2022	Created a quest system that assigns quests to player	Created a quest system that assigns quests to player
28/03/2022	Changed way quest system works to incorprate weightings	Changed way quest system works to incorprate weightings
02/04/2022	Implemented npc behaviours	Implemented different behaviours depending on world state
06/04/2022	Added resources to each npc	Each npc now has inputs and outputs regarding materials, so that each of them can interact with eachother

07/04/2022	Added ways for npc states, to add new states to the world if certain conditions are met	If certain conditions are met then new world states are enabled to allow for quests based on npc state.
10/04/2022	removed the ability for npc states to affect world and quests.	Constant fluctuations in different resources kepts creating very similar quests, ignoring all other aspects of the quest gen. disabled so that it doesn't ruin the project
14/04/2022	Started Writing Dissertation	Inserted previously written content into templated. Also added project timeline and carried writing the project log
30/04/2022	Handed dissertation + artefact + video in.	fin

Appendix B: Project Timeline Created at the time of writing research proposal.



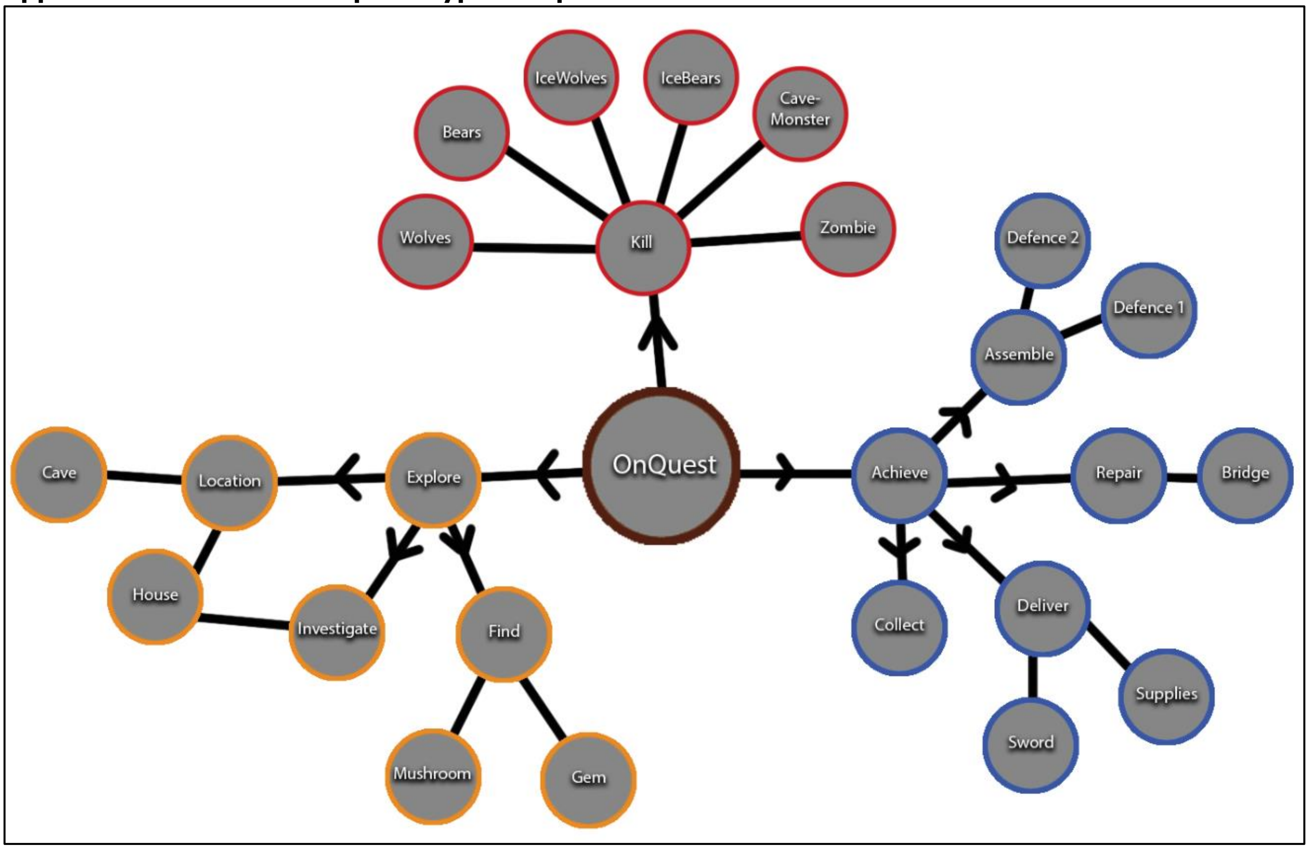
Month	Task
October	Research for Proposal
November	Write proposal
	Proposal 4th
	Research for research documentation
	Create unity project
	Implement ai actors
	Start building world states
December	Write research document
	Research Documentation 16th
	QA current project
	Start trying to do quest generation
January	QA current project
	Analyse quest system
	Continue with quest system
February	Look at world states/ AI actors for improvements
March	Finish up project
	Start writing report
April	Make video
	Report and Video 28th
May	Viva

C

Appendix C: Assets used in the project (not included in word count)

- G.O.A.P. Planner – Unity tutorial created by Penny de Byl - https://learn.unity.com/tutorial/the-goap-planner
- Low Poly Village Buildings –Unity user called 3dJeebs - https://assetstore.unity.com/packages/3d/simple-low-poly-village-buildings-99370
- Low Poly Pack – Unity user called Andrey Graphics - https://assetstore.unity.com/packages/3d/environments/low-poly-pack-94605

Appendixes D: Bubble Graph of types of quest



Appendixes E: Check to see if the king if hungry and needs food

```
    if (markeet.king.foodCONSUMED > markeet.king.foodSTORED)
        {
            if (markeet.king.HungryKing == true)
            {
                beliefs.ModifyState("OnPersonalQuest", 0);

                if (markeet.king.money >= 10)
                {
                    beliefs.ModifyState("SpawnDeliver", 0);
                    beliefs.ModifyState("AchieveQuest", 0);
                    beliefs.ModifyState("OnQuest", 0);
                    markeet.king.foodSTORED = 15;
                }
                else
                {
                    beliefs.ModifyState("SpawnWolves", 0);
                    beliefs.ModifyState("KillQuest", 0);
                    beliefs.ModifyState("OnQuest", 0);
                    markeet.king.foodSTORED = 15;
                }
            }
        }
```

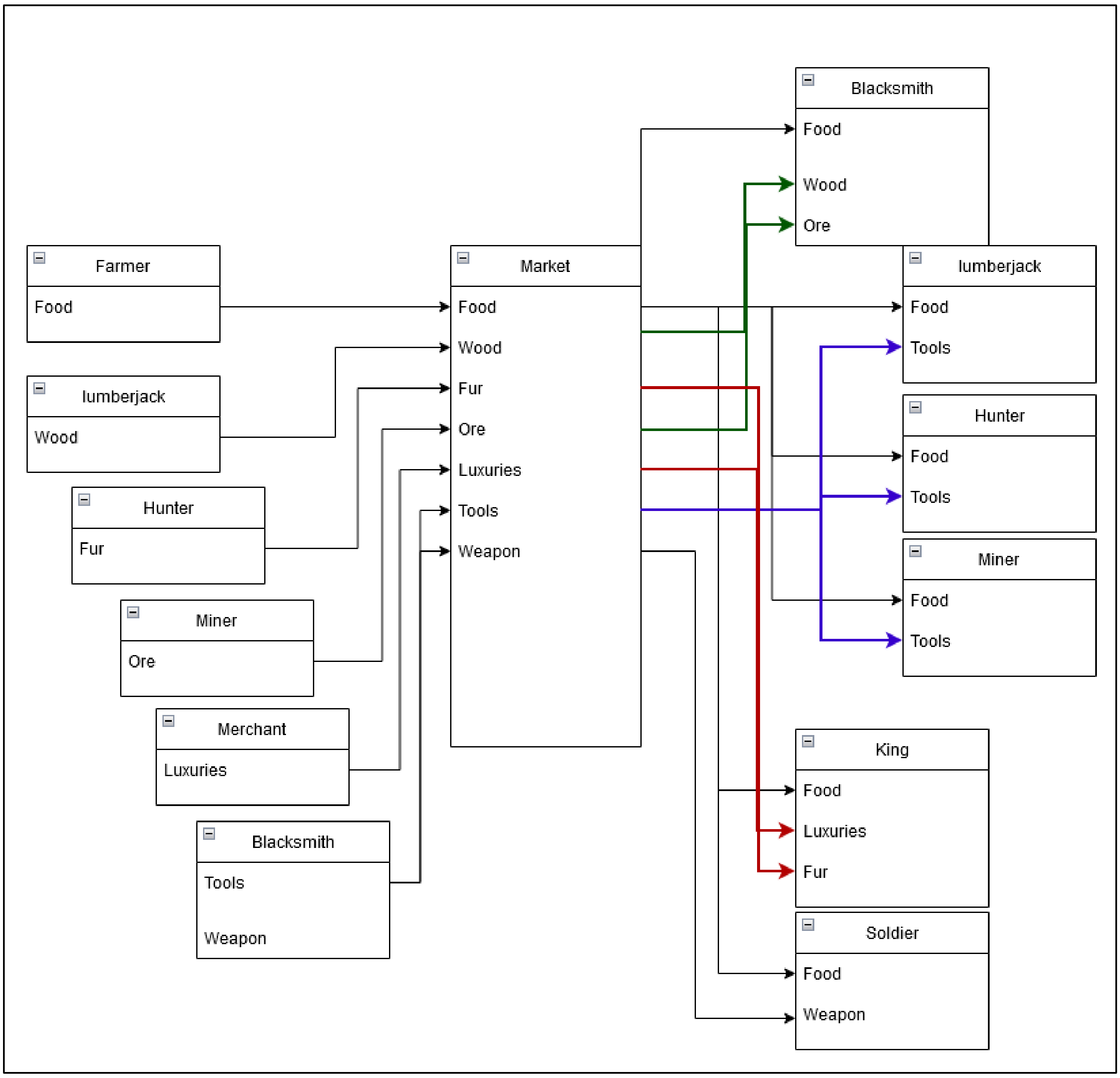
Appendixes F: Playstyle Weighting

```
for (int i = 0; i < 9; i++)
        {
            actualValues[i] = arraytest[i + 1];
            actualValues[i] = actualValues[i + 1];

            if (actualValues[i] == 1 && FirstNumber != 7)
            {
                FirstNumber++;
            }
            else if (actualValues[i] == 2 && SecondNumber != 7)
            {
                SecondNumber++;
            }
            else if (actualValues[i] == 3 && ThirdNumber != 7)
            {
                ThirdNumber++;
            }
        }
```

Appendixes G: Type of quest final iteration

```
playstyleAIWeighting();

        int comb1 = FirstNumber + SecondNumber;
        int comb2 = comb1 + ThirdNumber;
        ToQ = Random.Range(1, comb2 + 1);

        if (ToQ <= FirstNumber)
        {
            actualValues[9] = 1;
            ExploreQmax++;
            beliefs.ModifyState("ExploreQuest", 0);
        }
        else if (ToQ > FirstNumber && ToQ <= comb1)

        {
            actualValues[9] = 2;
            KillQmax++;
            beliefs.ModifyState("KillQuest", 0);
        }
        else if (ToQ > comb1 && ToQ <= comb2)
        {
            actualValues[9] = 3;
            AchieveQmax++;
            beliefs.ModifyState("AchieveQuest", 0);

        }

        //resets the values and keeps a minmium which means that the option is always
there
        FirstNumber = 2;
        SecondNumber = 2;
        ThirdNumber = 2;
```

Appendixes H: N.P.C.s, their needs and desires from the market



*only insert meaningful materials here, please don't just bulk this report up. Your main text should be able to stand on its own, without relying on information contained in appendixes. Check with your supervisor beforehand