Aalborg University Copenhagen

Semester: MED10

Title: Randomness in Games

Aalborg University Copenhagen
Frederikskaj 12,
2450 København S, Denmark

Semester Coordinator: Stefania Serafin

Secretary: Lisbeth Nykjær

Project Period: Spring 2017

Semester Theme:

Master Thesis

Supervisor(s):

Lars Reng

Project group no: N/A

Members:

Camilla Grønbjerg Jakobsen

Pages: 65

Finished: 2 June 2017

Abstract:

The thesis seeks to explore the possibility of constructing a procedural quest generator to create versions of the same quest structure which on each repeated encounter of such a quest using that structure would feel different to the player.

This problem was coined with the following problem statement:

"Can a procedural quest generator be constructed to create quests based on the same quest structure which feel different each time a player encounter them"

A system for a procedural quest generator using a structure based on having separate NPC motivations for giving the quest along with possible strategies for completing quests with that motivation. The variation for each separate instance of a similar quest were done using randomised spawn points, objective amounts and objective items, enemies and associated NPC.

The system was incorporated into a RPG type game made in Unity and tested by 67 participants. The test proved that the project was a success as that the participants found the generated quests different and varied and attributing this to the variation on randomly selected amounts, items and enemies primarily.

Table of Content

1 Introduction

When creating a game there are many things to consider, and during any game production certain choices or compromises are made in order to realistically produce the game within the given timeframe. This goes for small as well as big production groups, but especially if involved in a game production which does not employ people specialised for all aspects of the game. In these situations the production team will often create or find tools to help alleviate these shortcomings. (Chandler, 2014)

However, what about when it is not a question about the competencies or roles of the production group but the requests of the consumers? Currently players of all types of games want content, content and more content, be it in a big MMORPG, a small dungeon crawler or even sandbox games. This type of consumers want quantities of new things to do in their game of an adequate or high quality, not just replaying 'old' content; when these consumers buy a full-feature premium game they expect updates after its launch to supply them with a stream of content and if this does not happen bad reviews or forum complaints generally ensure. All of this is evident from looking at game services such as Steam (Valve, 2017) and GOG.com (CD Projekt, 2017) who allow user reviews on the games they supply, where many bad reviews often reflect the length of the game, lack of updates or lack of content. These types of complaints are also frequent on the games' own forum sites and Reddit (Reddit inc., 2017).

Many game companies try to pass this demand one way or another, and recently it has become quite popular to use procedurally generated content to accomplish this. By procedurally generated content is meant using an algorithm to randomly generate some form of content in a game, in essence it is a random generator with rules for how random. Similar to how Noor Shaker, Julian Togelius and Mark Nelson described it in their book 'Procedural Content Generation in Games' stating procedural content generation (PCG) as *"… the algorithmical creation of game content with limited or indirect user input"* (Shaker, Togelius, & Nelson, 2016)

Currently procedurally generated content have been done in many different aspects of game productions such as creating terrain, cities/villages, vegetation, maps/levels, sounds, quests/objectives, textures, interior outfitting and clutter just to name a few. Essentially, if you can imagine something as procedurally generated it can probably be done or have already been

done. There is so much being done in procedurally generated content currently that game jams around the world even use it as a theme; there even exists a sub-Reddit dedicated to procedurally generated content of all sorts (Reddit inc., 2017).

In addition to all the procedural generation booming on online sites such as Reddit, just as much have been done on the topic in the academic world. The book 'Procedural Content Generation in Games' (Shaker, Togelius, & Nelson, 2016) is a newer example of a collection of the more recent research done into the subject looking at the many different types of content which have been created by procedural generators showcasing algorithms, diagrams, factual basis and references for all the previewed content. However, much more research have been done in the field even though the book was published in 2016, since as stated in the preface of the book much of it was written around the year 2013.

As procedurally generated content have become such an expansive field covering most everything software can produce in a game as long as someone can imagine it, there is one problem that comes to mind. The thing with procedurally generated content is that most consumers will after some time of playing notice the patterns created by the generator for whatever content it is creating; an example of this is how high-ranking ladder players in Diablo 3 (Blizzard Entertainment Inc., 2017) can recognise the different pieces or map fragments the generator use to create the tiered dungeon challenge called Greater Rifts, they know where key elements (pylons) will be placed and from looking at the current map they can reason to which way the exit is. Many other such examples exist for other types of game content, and as such there seem to be one flaw with the current state of procedural generated content: what the generator creates is not significantly different in such a way that they can be mistaken for a real designer's or artist's work.

This lack of difference in the created content most likely stems from the limitations from the algorithm itself, often put in place to make sure the content is playable and will not reach cases where say a map does not allow the player to reach an exit or areas being boxed in with vegetation and being unreachable. As the complexity of the algorithm rises, the harder it is for the generator to find suitable executions within the limitations, and as such patterns will arise more often. This means less amount of different contents created from the generator, or just less variation on them. A way to combat this is to supply the algorithm with more pieces for the

generator, so it has more options to take from. However, this seems to defeat the purpose of having a tool to create content for you, with the added benefit of cutting down on designer and artist work if the only way to keep it going is to supply it with more; which is why the key problematic of procedural content generators are to have a self-sufficient generator that can create additional elements for parts or all of the content being generated.

In continuation of this it is interesting why so few games utilise a procedural content generation system for repeatable content, such as repeatable or daily quests in massive multiplayer online games (MMOs); where these types of quests are still being designer created and placed into a pool of quests with a handful selected each day and shuffled. The core feature of many games is quests, also called missions, challenges or just objectives (Rogers, 2010). Many types of games utilise quests of some sort and loads of research have been done as to why games use quests: some to guide the players through the game, give a feel of progression, replayability with choice-based quests, and even endless gameplay through procedural generated quests or similar systems. Generally it is giving the player a clear motive to play by giving them small clear goals, whether it is a goal right now or on a larger scale as a whole. As such it is interesting why so few games utilise a procedural generation approach to keep the repeatable quests from being too repetitive and alike, which one would assume would keep the users playing for longer and thereby supporting the game.

Furthermore much research has been done into creating procedural quest generators for a multitude of games, though primarily exemplified for MMOs or RPGs. However, very little have been done to ascertain whether or not it is actually possible to generate quests that seem significantly different for the player, even if using the same quest structure and how this can be incorporated into a procedural quest generator to make endless quest related gameplay in a game. The hard aspect of this problem is to create diverse quests procedurally while ascertaining the feel of the game and give the feeling that it is created by a designer and not a computer algorithm, as is the general downfall for procedural generators; but also to keep the quests generated from feeling repetitive and alike.

1.1 Problem Statement

To sum it all up, the above mentioned points could be coined into the following problem statement:

“Can a procedural quest generator be constructed to create quests based on the same quest structure which feel different each time a player encounter them”

2 Analysis

The following chapter will go further in-depth with quest structures and procedural generation in relation to quests, to narrow down the requirements for a procedural quest generator prototype fitting the problem statement; starting by districting procedural content generators from one another in general terms, and then moving on to the more specific topics of quest structure and quest generators.

2.1 Properties of procedural content generators

When creating procedural content generators it is usually to combat a problem with the game production, production team shortcomings, or as a solution for consumer needs and requests. Depending on what the procedural content generators needs to be a solution to there are generally properties that explain their limitations and usability. In the book 'Procedural Content Generation in Games', Noor Shaker, Julian Togelius and Mark Nelson (2016) describe properties that are used to describe the generators from research they have collected, these being:

- Speed
- Reliability
- Controllability
- Expressivity and diversity
- Creativity and believability

All of these properties are important for any procedural generators, however, generally speaking the procedural content generators will focus on one or more of these properties as they are required for the solution, and will usually give trade-offs in some way depending on what property is in focus. As only in the ideal world, and quite possibly only with a supercomputer, can you have a procedural content generator create the perfect content in no time with no failure and always at high quality; but realistically some of the properties would be lowered to allow others to remain at high quality e.g. generate content with no failure in a very short time with a trade-off on quality.

For this project the main focus of the procedural content generator is on 'Expressivity and diversity' and to some degree 'Creativity and believability'. The quests should be expressive of the

game areas they are placed in and feel connected to the game as a whole, while staying different as we want it to look like it is a lot of diverse content made by humans and not an algorithm. Meaning the quests generated by the generator should sample as much of the game world as possible, including locations and game elements.

Furthermore, it needs to be playable so the reliability is also a factor, so it should not generate quests that cannot be completed due to the fact that there are not any of those enemies or objects which are needed for the objectives. The speed at which these quests are generated is inconsequential, if the quests are pre-generated beforehand the generation speed can be quite long and if its generated during play then seconds or even a couple of minutes would be acceptable if there were already a couple of other quests to choose from in the meantime. However, if all the quests made available can be completed in one fell swoop there needs to be something ready to be placed before this total completion happen.

2.2 Taxonomy of the procedural content generator

In addition to the properties mentioned in the previous section (2.1), Shaker, Togelius and Nelson (2016) also expanded on the previously established taxonomy for procedural content generators, making it easier to categorise the different solutions based on several crucial aspects. Their taxonomy also puts limits on the properties of the solution, if they are to fall within certain categories. With so many procedural solutions being done currently, having a look at these taxonomical categories can help describe how one solution differ from another. In relation to this project this will help establish further the limitations of the prototype and explain how it will differ taxonomical. The expanded categories are as follow:

- Online vs offline
- Necessary vs optional
- Degree and dimensions of control
- Generic vs adaptive
- Stochastic vs deterministic
- Constructive vs Generate-and-test
- Automatic generation vs mixed authorship

For general procedural generators being either necessary or optional would depend on how vital the quests are for a certain game, and essentially how one defines quests. For this project quests are deemed necessary for the completion of the game, and as such fall under the category of 'Necessary'. The quests created by the generator will be 'Generic' and 'Deterministic' as they will not adapt to player actions and are allowed to create the same content if the same parameters or conditions are met, respectively. For the last categories it all depends on when the generation is intended to take place. For 'Offline' generation means generation of content before the game or system is started up, here the dimension of control is normally done through what is called seeds, numbered elements created beforehand and loaded on start-up. For 'Online' generation is referred to generation while the game is running usually done through parameters or variables. Lastly whether or not the generation is done through a constructive method or generate-and-test method depends on the amount of certainty there needs to be that the solution generated satisfactory content. For necessary content a generate-and-test method would be better, but the generation time will likely increase as well, and should be taken into consideration.

2.3 Quests Structures

Quests are generally speaking used to guide the player through a game, give them a clear road to follow and even optional roads they can pursue if they so desire, but overall give them objectives to do and things to strive towards getting done. It ensures the players have something obvious to do in the game and often gets the player started and introduced to the game and environment or even mechanics (Rogers, 2010). However, quests are also the main source of gameplay that keeps the players playing the game and financially supporting the creator, this especially goes when talking about MMOs such as World of Warcraft and Final Fantasy XXIV which are subscription based games[1] (Doran & Parberry, 2010).

Furthermore quests are also the primary device in many games to deliver stories and create narratives through the objectives themselves, the actions the player needs to take and the reasoning behind setting out on the quest in the first place. Much of this story can be contributed to the initial introduction to the quest often given by a quest giver or non-playable character

[1] Subscription based games: A pay-to-play model where users pay monthly in order to be allowed to log into the game and onto the online servers containing the game.

(NPC). These NPCs will often tell the player they need their help or that it is somehow vital to go explore an area for whatever reason. These enlistments or suggestions leads to the main tasks of the quest and the completion of these will reward the player with some sort of prize or payment.

In his dissertation Jonathon Doran (2014) goes into depth about procedural content generation for online role playing games (RPGs). In addition to explaining several uses of procedural content generation and exploring its advantages and drawbacks, he goes into different uses of procedural content generation; one use being for quests. He differentiates quests as either narrative devices or as functional game elements. For narratives he notes that consistency is important, if they are meant to be believable, as such occurrences of the same NPCs in the same town in or around the same places is needed. Whereas, when seen as purely game elements they require an objective, actions to do, places to go, characters to encounter and a possible reward. Dialogue can be limited to the initial and end encounters with NPCs; the initial encounter leads to the actual objectives and actions the player needs to do which are usually tracked until completion leads the player to the ending NPC or location.

There are many studies into the structures of quests can be broken down into smaller fragments and essential pieces which makes a quest, especially studies made with the prospect of using these structural parts in creating procedural generated quests. By identifying the underlying structures of previously designed quests, templates of or arbitrary quests can potentially be designed and even generated and filled with fitting game content pieces by algorithms to fit into a game and possibly any game out there.

Jonathon Doran and Ian Parberry (2010) collected previous research into these quest structures and expanded on these by collecting quests from online fan sites of several MMOs including World of Warcraft. From these quests they did a structural analysis of these quests in relation to what was previously know, but ended up separating quests into parts; two of these parts are NPC motivation and Player Action and these will be explained in the following subsections.

2.3.1 Non-playable character (NPC) motivations

The first thing a player is given when encountering a quest, is the motivation of the NPC. These motivations represent the things the NPC want to accomplish by completing the task at hand, and why they are asking for the player's assistance. An example of such motivations could be

knowledge or protection among others, such as asking a player to kill some creature foreign to the NPC so that it can study it or escort and provide protection to the NPC while it travel an area to investigate something. In their study, Doran and Parberry (2010) looked at a sample of over 3000 quests over four different games and determined rough categories of NPC motivations from which they created 600 arbitrary quests which fitted every of the 3000 quests sampled. These arbitrary quests lead to categorising the following motivations and sub-motivations (as seen in Figure 1).

One thing to remember is that these motivations is purely a NPCs self-interest, and although it is used to give the players the reason for doing the tasks, there are many instances of the opposite; where the NPCs motivation seemingly makes no sense, is concealed or even where the NPC lies.

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

Figure 1 – NPC motivation (Doran & Parberry, 2011)

2.3.2 Player actions

After the player has gotten the NPC motivation this leads to the actual objectives or task of the quests. The way the player accomplishes these objectives are by Doran and Parberry (2010) represented by actions the player have to take during a quest (show in Figure 2).

Section	Action	Meaning
1	[Attack [xN] {NPC, Item}]	Damage an entity
2	[Talk NPC]	Talk to an NPC
3	[Assemble Item]	Assemble a new item from parts
4	[Give Item to Entity] [Take Item [from Entity]] [Trade Item for Item with Entity]	Trade items with an entity.
5	[Defend Entity]	Defend an entity against attacks.
6	[Goto Entity]	Visit an NPC, item or world location
7	[Use {Item,Skill} on Entity]	Use an item or skill on a world entity.

Figure 2 – Table showing possible player actions during quests (Doran & Parberry, 2010)

From Figure 2 Figure 2 – Table showing possible player actions during quests are seen arbitrary actions which could easily be made into objectives which could in turn easily be filled by a procedural generator. As an example [Attack [xN] {NPC, Item}] would become [Attack X amount of

{NPC} or {Item}] where {Creature} and {Item} could be substituted for any enemy or item already found in the game world. The reason for this translation is that it is not necessarily only a creature that should be attacked but can also be inanimate objects such as urns or crates.

Quests can be constructed of one or more actions in collaboration with a motive, an example of such as quests could be: Go to a location X and search crates for missing pieces of Y, assemble Y and bring it back to NPC.

2.4 Procedural Quest Generators

In relation to procedural quest generators in games it is widely used in many types of games from singleplayer to massive multiplayer online games (MMOs) and a wide array of genres role-playing games (RPGs), adventure and even action games. If we look at games such as World of Warcraft and Final Fantasy XIV they each have quests that are picked from a pool of pre-created quests/objectives done by designers, called World Quests and Fates respectively. These World Quests and Fates are quests picked by an algorithm to decide which and how many quests out of a pool shall be active based on parameters. As such they are procedurally populating an area with quests from a pool of designer made quests, which lacks the variety of procedurally generated quests seeing as when the parameters are the same quests are very likely to repeat. However, the systems do share some of the same traits, and is basically the type of system this project would be able to improve or replace if it is shown to be possible.

There exist many different algorithms for procedural content generators among others search-based and planning algorithms. A search-based approach to the algorithms generally speaking means an algorithm which search for the best possible content based on variables given by creating some content and grading the fitness of each attempt until it reason no better solution can be found; each consecutive attempt tries to beat the fitness of the previous and when no solution is found with better fitness the best solution is taken, usually constricted by a timeframe. Planning algorithms on the other hand search for a solution using an order of actions that needs to be taken to complete the task; often used in artificial intelligence such as creating pathfinding tools for robots or virtual entities. (Shaker, Togelius, & Nelson, 2016)

Looking at the different research done into procedural quests generators in the book by Shaker, Togelius and Nelson (2016) along with other research found such as papers by Young-Seol Lee and Sung-Bae Cho (2012) and Calvin Ashmore and Michael Nitsche (2007), it becomes apparent that many different solutions and algorithms work in relation to generating quests procedurally. The one thing standing out seems to be the type of quests each study tries to create. Story or narrative based quests or quests which need connections between them and other quests are often done using a planning algorithm. Whereas quests which main purpose is supplying additional gameplay or objectives after ending the initial storyline or even as an additional activity to the main storyline, search-based algorithms in some degree seems to be the favoured approach.

Lee and Cho (2012) created a quest generator using planning, where the algorithm created a quest by defining a start and end point and a middle criterion (the actions) which in turn creates the sequence of the quest. The start and end point were used to connect quests to each other, meaning when you finished a quest you would end in a place where another quest started.

Ashmore and Nitsche (2007) study explore puzzle like quests using a lock-and-key methods, where lock-and-key is defined as something that stands in the players way which the player needs to find some item to allow him to pass the obstacle, not necessarily an actual door and a key. They explain a Java search-based method for creating and extending the world space which allow for generating what they define as quests, through placing of items which can be used in several ways one being to allow access to other parts of the game space by e.g. breaking parts of the terrain. The quest is seen as the objective of progressing through the area into the next and these lock-and-key puzzles are what allows the player to complete and find said puzzle and enter next area.

In their other studies, Jonathon Doran and Ian Parberry (2011) (2015) use the motivations and actions as described in section 2.3 to construct procedural quest generators. Their first study, Doran and Parberry (2011) created a procedural quest generator based on their structural analysis of quests from four MMORPGs. They derived that each motivation of a NPC can lead to a certain amount of sequences of possible actions to complete said motivation seen in the light of the analysis they had performed; these sequences of actions were named as strategies and each motivation could have several. Therefor they created an algorithm, which chose a motivation,

picked a derived strategy with the subsequent actions which in turn created the basis for a quest. The full list of strategies and actions in relation to motivations can be seen in Figure 3.

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>
Comfort	Obtain luxuries	<get> <goto> <give>
	Kill pests	<goto> damage <goto> report
Reputation	Obtain rare items	<get> <goto> <give>
	Kill enemies	<goto> <kill> <goto> report
	Visit a dangerous place	<goto> <goto> report
Serenity	Revenge, Justice	<goto> damage
	Capture Criminal(1)	<get> <goto> use <goto> <give>
	Capture Criminal(2)	<get> <goto> use capture <goto> <give>
	Check on NPC(1)	<goto> listen <goto> report
	Check on NPC(2)	<goto> take <goto> give
	Recover lost/stolen item	<get> <goto> <give>
	Rescue captured NPC	<goto> damage escort <goto> report
Protection	Attack threatening entities	<goto> damage <goto> report
	Treat or repair (1)	<get> <goto> use
	Treat or repair (2)	<goto> repair
	Create Diversion	<get> <goto> use
	Create Diversion	<goto> damage
	Assemble fortification	<goto> repair
	Guard Entity	<goto> defend
Conquest	Attack enemy	<goto> damage
	Steal stuff	<goto> <steal> <goto> give
Wealth	Gather raw materials	<goto> <get>
	Steal valuables for resale	<goto> <steal>
	Make valuables for resale	repair
Ability	Assemble tool for new skill	repair use
	Obtain training materials	<get> use
	Use existing tools	use
	Practice combat	damage
	Practice skill	use
	Research a skill(1)	<get> use
	Research a skill(2)	<get> experiment
Equipment	Assemble	repair
	Deliver supplies	<get> <goto> <give>
	Steal supplies	<steal>
	Trade for supplies	<goto> exchange

Figure 3 – The different motivations, strategies and sequence of actions (Doran & Parberry, 2011)

From these strategies and action sequences, they created rules the generator had to follow when creating quests. The generator created for the paper was tested and created a quest with a knowledge motivation leading to <spy>, the quest had several possible outcomes depending on the actions of the player.

In their second study, Doran and Parberry (2015) created a framework for executing procedural generated quests based on event triggers based on changes in the game state such as a player entering a location. When the event was triggered, the quest handler would check the associated event with its list of quest triggers and if matched it would fire the event to run the script associated with the selected event and quest.

As the intention of the project is to supply the users with additional gameplay elements to allow for extending gameplay time, the focus will not be on narratives and as such a search-based approach will be most ideal according to previous research. For a basis for the general quest structures to build upon in the procedural generator, the scheme of NPC motivation, strategies and action sequences by Doran and Parberry could be useful as it allow for easy extension and diversifying of quests using a general quest structure.

2.5 Analysis Conclusions

From this chapter the following conclusions can be made for creating a procedural quest generator which fit with the overall intention of the project and the problem statement.

The quests the generator will create be functional game elements, as described in section 2.3, since the intention is to create quests which can be seen as a procedural replacement of game elements such as World Quests and Fates in World of Warcraft and Final Fantasy XIV respectively. The focus will be on the content the quest supply and not the possible narrative elements which it may include.

1. Generated quest will be functional game elements.
2. Focus on content, not narrative.

As explained by the problem statement (section 1.1), the main focus is to create diverse quests from the same structure upon repeated encounters. Meaning the focus of the generator is on 'Expressivity and diversity' (see section 2.1). If the intention is to create a quest generator derived from the structure of motivation, strategy and sequence of actions as formulated by Doran and Parberry (2011) a minimum of three motivations with their strategies and actions used as structures (see section 2.4) should ideally be enough to show if the prototype generator can create enough difference in the quests; which can later be expanded on if shown not to be enough.

3. Use a minimum of three different motivations with underlying strategies and sequences of actions as quest structure, as theorised by Doran and Parberry (2011).

To allow the quests to be more diverse and expressive of the game world each quest should have objects, creatures and characters from the game world should be present in the objectives of the

quests. However the quests should not all concern themselves with the same part of the world, meaning the quests should be spread out over the world space supplying a wide range of possible quest areas.

4. Generated quest contain different elements from game world.
5. Generated quests should be located in a wide array of locations.

To ensure that there are always quests to complete, and that the quests can be completed it is important that the generator is reliable; meaning the generator should ensure that the proposed solution can be completed and fit with each requirement before allowing it into the game. For the algorithm to always ensure that the quests can be completed and to ensure the highest quality of the generated quest in relation to the game world a search-based approach to the procedural algorithm would allow grading each attempt at a solution until an adequate solution is found which ensure the best possible result within the restrictions. (Section 2.4)

6. Generator should use a search-based algorithm approach.

The quests generated between each playthrough of the game should be allowed to be the same, so that you can encounter the same objective more than once whoever these repeated occurrences should not be common. (Section 2.2)

7. Same quest can repeat if the conditions repeat.

Furthermore, as it is not the intention of the project to create a procedural quest generator which adapt to player actions; meaning that the generator will not take into account that the player has encountered the NPC and objective before such as stating that it is a re-investigation of an area for example. (Section 2.2)

8. Generator will not adapt quest to player input.

The generator should generator new quests during play which fit into the location which have opened up after a quest was completed, keeping within the parameters and restrictions of the original placement of quests. However, the new quest should be a new one, and not identical to the one it replaces to retain diversity and avoid repetition.

9. Generator work online (during play) to find new quest once one have been completed.
10. Replaced quests cannot be a repetition of the one it replaced.

2.5.1 Initial Design Requirements

All in all this gives the following ten initial requirements for a prototype procedural quest generator.

1. Generated quest will be functional game elements.
2. Focus on content, not narrative.
3. Use a minimum of three different motivations with underlying strategies and sequences of actions as quest structure, as theorised by Doran and Parberry (2011).
4. Generated quest contain different elements from game world.
5. Generated quests should be located in a wide array of locations.
6. Generator should use a search-based algorithm approach.
7. Same quest can repeat if the conditions repeat.
8. Generator will not adapt quest to player input.
9. Generator work online (during play) to find new quest once one have been completed.
10. Replaced quests cannot be a repetition of the one it replaced.

3 Methods

The following chapter explains what methods which were used in order to develop the best prototype possible to solve the described problem and create a working prototype solution, along with testing and evaluation methods.

3.1 Iterative Design Method

Many ideas are gained during the research phase of a project but not all ends up being the best working solution or even the most well liked one by the users. When working on something where there exists similar solutions but the project have a different take on the concept it is not possible to just go out and develop a prototype and expect it to be perfect in one go. As such an iterative design process seemed the most ideal solution to how to slowly get the prototype to be moulded into something potential users would like.

The iterative design process is a type of usability testing which intends to develop a design idea into its full potential using user input; it is normally used for user-interfaces but can be expanded to work for anything which users need to interact with. Jakob Nielsen (1993) wrote a paper on Interactive User-Interface Design in which he describes the iterative design method as *"Iterative development of user interfaces involves steady design refinement based on user testing and other evaluation methods"*. This should be understood as creating an initial design and slowly refining and improving the design until it is perfected. To do this the users who are expected to use the prototype will be used as sources of information through testing and evaluation while staying true to the initial design premise.

However, Nielsen notes that it is not valid for the iterative process to change any part of a prototype even small aspects without it being rooted in findings from testing or evaluations of other iterations which he explains as *"Iterative design aims specifically at refinement based on lessons learned from previous iteration"* (Nielsen, 1993). This means that the requirements set forth in the previous chapter (2.5.1) will only be changed based on the input gotten during the iterations made during the project, and no other changes can be made.

Furthermore, Nielsen explains that initially the improvements made to the prototype will be quite big while over time and iterations the improvements will be minor. This is contributed to the fact

that in the beginning of the process it will be major design flaws you will be fixing and later it will only be small errors or tweaks. He visualise this relation between the change in usability over the course of iterations in Figure 4 (Nielsen, 1993), where the first bump in usability is contributed to the first few iterations with larger flaws, while the second bump in usability is when its merely from tweaks when trying to perfect the prototype.

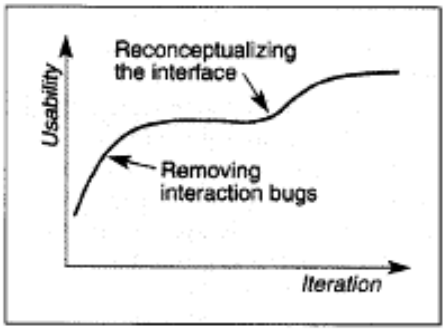


Figure 4 – The relation between interface usability and number of design iterations as conceptualised by Nielsen (1993)

Lastly Nielsen describes the necessary steps a designer needs to follow to successfully employ the iterative design method as follow:

"Interface designers complete a design and note the problems several test users have using it. They then fix these problems in a new iteration, which they test again to ensure that the 'fixes' did indeed solve the problems and to find any new usability problems introduced by the changed design" (Nielsen, 1993).

Put in perspective of this project it could be interpreted into the following six steps:

1. Decide the initial requirements for the prototype
2. Design and create design prototype based on requirements
3. Run a user test on the prototype using users intended to use the prototype
4. Note any problems had and improvement suggestions by the test users
5. Redefine requirements from user input to fix problems
6. Repeat steps 2-5

These steps will be followed during the iterative process of the project, which will be explained further on (see chapter 4).

Generally this process will be repeated until the changes are hypothetically at the second bump (see Figure 4 (Nielsen, 1993)) meaning when there are not any clear changes needed or the users cannot come with definitive problems. However due to time restrictions the prototype are most likely not going to have as many iterations as would generally be seen when working with user-interfaces, probably landing in usability between the first and second bump.

3.1.1 Testing method for iterative steps

When it is vital to gain knowledge from user feedback there are several ways to go about asking the users, each method having their strengths and weaknesses. If large numbers are needed surveys can be very helpful, however open-ended questions or questions in general which asks for longer answers usually go unanswered. This makes a survey quite limiting as it will be quite board but not very deep, and depth is very important in terms of an iterative design process especially for the iterative steps. This leaves interviews and focus group based testing as more viable choices for user testing. With focus groups allowing for a quicker gathering of a broad section of user opinions which still gives significant depth in the answers, it seems like the best method. Especially when each opinion or input can be related to other and even commented about by other focus group participants; it essentially gives a pro and con for each opinion and can be gathered into a more unifying input overall to easily adapt into new requirements.

In regards to focus groups, Jonathan Lazar, Jinjuan H. Feng and Harry Hochheiser (2009) explains that there are no general consensus about the size and amount of focus groups. However if the following applies it will increase the chances of success in terms of gaining usable data from focus groups:

- Around 5-12 people in each group
- At least two group

This testing method will be done for each iterative step to redefine the design requirements and lead to further iterations of the prototype. The people in each focus group will be chosen based on the target group for the prototype (explained in section 3.2).

To get a broad section of feedback without going off topic a semi-structured focus group setup will be preferred, whether done physically in person or online. The same setup structure should be

usable in all instances of testing, meaning if both testing in-person and online the setup should be the same for both. As such the following structure should be utilised to keep the participants on topic.

The time the participants should play the prototype might is quite short, as daily or repeatable quest types are generally very short activities which will generally not be continued for extended periods of time but rather in short bursts. This is due to the fact that most MMOs such as World of Warcraft (Blizzard Entertainment Inc., 2017), Rift (Trion Worlds, Inc, 2017) and Final Fantasy XIV (SQUARE ENIX CO., LTD, 2017) all have limits to how many quest you can complete throughout a day by having limited quests available or having quests on timed durations. From personal experience it is quite possible for a player to complete around 20 daily quests with simple objectives in as little as 10 minutes, so for a playtime around 10 minutes should give the players plenty of time to experience the quest generator in action.

As such the following structure will be used when conducting the focus groups:

- Participants try the prototype for a short time, around 5-10 minutes.
- Each participant is given six post-it notes to describe thoughts on two topics, the first things that come to mind good or bad.
    - Three to explain their general feeling for the prototype
    - Three about the quests in the prototype
- Talk about each topic of post-it notes. Get them to explain, let them comment on other participants' notes, and generalise the notes.
- Ask about the quests. Are there enough quests to choose from? Did they feel generated? Did the quest feel different or were they repetitive? What would they suggest be changed?

Furthermore, observations should be taken during the whole focus group process and noted down. The focus of these observations should be on the prototype noting any:

- Problems with the prototype?
- Bugs?
- Player comments during play.
- The quests generated.

- Preferences in quests completed.

For the observations, the point about problems with the prototype should only be needed for the first couple of iterations to iron out major design flaws; afterward the focus should be entirely on the quest generator.

3.2 Target group

Whenever testing is required it is important to outline the people who are intended to use the prototype to get usable feedback. Even more so when using a design process which is user-centered and require testing of core users in order to further develop or refine the initial design and prototype such as during an iterative method.

The group of users intended to use a product or prototype are called a target group, and the group we will be interested in receiving input from when testing the prototype. There are many ways to define a target group for a project, in his book Thomas Bjørner explains the four most used groupings of target groups for defining users or consumers for a specific product; these four groupings being demographic, geographic, psychographic and behavioural. (Bjørner, 2015, p. 57) Bjørner also explains other ways of defining users, which partly require a user-base beforehand.

Since the prototype is not expected to be a standalone system there is not an inherent user-base; it therefore seemed smart to look at the games wherein we expect to find our system. If going for a quest scheme similar to those found in World of Warcraft and Final Fantasy XIV in a procedural fashion then the research done by Nick Yee (2006) and Mark D. Griffith and Mark Davies (2003) could help narrow down the players of such games as in these studies the demographic of players in multiplayer games such as MMOs are enlightened. These studies show sociodemographic representation of said players is more than 80% male. The ages of the players in the studies show two very different pictures, with the study by Yee study showing ages ranging from 11 to 68 years old while the study by Griffith and Davies ranges from 14 to 29, with only Yee explaining the median of the study, being 25 years old.

Taking all the above into account and adding to it what the project tries to accomplish , it leads to the fact that the players would need a certain age and level of reading to have the comprehensive abilities and reasoning to fulfil what the system require of them, getting an objective and reason

their way into completing it. With these requirements for the players, the user target group would be primarily males of 15 years old or older. As the product is intended to solve the desires of video game consumers, these players would in addition to that also play video games of some description.

3.3 Final Testing Method

For the final test it was important to find a method of distribution which could reach a large segment of the target group, while retaining easily distinguishable data in a given timeframe. As such it was decided that a method different from the iterative test method, the focus groups, was needed since the downfall of focus groups is reaching large amount of people in a relatively short time. Surveys and questionnaires on the other hand can reach a wide section of potential users in a very short time; they are very broad but with much less depth in the questions. (Lazar, Feng, & Hochheiser, 2009)

As such a survey method was selected for the final test. The test would be given to segment of the target group (explained in section 3.3.13.2) who will be free to play the prototype and then answer a survey provided along with the prototype. It was decided that the distribution would be done online, giving potential testers a link to the prototype and survey; as such this would allow the users to complete the test from the comforts of their own home and on their premise. This seemed to provide more up-sides than down-sides. As the testers would not be monitored or interrupted when asked to test, this would provide the testers with enough time to comfortably complete the test and not being monitored generally means more truthful answers to a survey. The down-side is that you cannot ensure that the users who are sent the prototype and survey will actually play the prototype and answer the survey; as such I can be advisable to contact the testers if not enough respond back.

When providing the testers with the prototype and accompanying survey the following should be made clear:

- Purpose of the test - project for a master thesis.
- Suggested play-time of 10-15 minutes.
- Lead the testers to survey after end play session.

For the final testing of the prototype the following sub-sections describes the procedure for finding the testers and the run-down of the survey itself.

3.3.1 Sampling the testers

For the final test the testers would be found from the target group (explained in section 3.2) through online and physical communities related to video games. As not all online and physical communities would be representative of the target group some guidelines were set up to find suitable communities. These were as follows:

- Online game communities, such as guilds[2] and forums for games such as World of Warcraft and Final Fantasy XIV.
- Game and technical discussions forums such as Reddit.
- Network connections that play or develop games.

As such these testers were found using a quota sampling method. Quota sampling is a non-probability sampling methods where the sample of participants are based on the pre-specified characteristics (Bjørner, 2015, p. 62). In this instance it means selecting people of 15 years old or older who plays video games, signified through the above mentioned communities.

3.3.2 The survey

For the final test a survey data gathering method was chosen, as explained in the above section (3.3). Surveys is a method which can easily gather quantity of data concerning the topic in the prototype during the final test, reaching a broad selection of testers who could be potential users if delivered and sampled correctly. In addition to the broad selection of target group testers, a wide range of questions can be asked in a survey, however, these questions should be pre-determined and correctly worded, and should avoid being open-ended or long answers as these type of questions rarely provides useful information. (Bjørner, 2015) (Lazar, Feng, & Hochheiser, 2009)

Regarding the questions in the survey there a many different ways of structuring them, some using statements, multiple-choices, visual representations or even some sort of scales are all

[2] Guilds: In-game community of online games such as World of Warcraft, allowing character connection and benefits between players.

highly used methods. However, they should be clear and not leave much to be interpreted by the tester. Yes and no questions, and questions regarding scales are generally very useful in getting precise meaningful data, which can easily be quantified. Open-ended questions rarely provide useful data or are left unanswered, and as such will not be used in the survey.

The survey will start out with questions regarding age, genders and whether or not they normally play video/computer games. This information will be used to place the testers as part of the intended target group (as explained in section 3.2). The rest of the questions will concern the quests the testers encountered during their playthrough of the prototype. A full list of questions can be seen in the appendix section 14.

The survey used the following types of questions:

- Yes/no questions.
- Likert scales to degree of agreement to statements.
- Semantic differential scale.

The yes/no questions should be used to ascertain if the generator worked as it should such as spreading out the quests far enough and only giving objectives which could be completed with the objects available in the game world.

Statements should be given to the users asking about the essential parts of the quests that were generated in order to ascertain if the project fulfilled the problem statement, whether the quests felt different enough. These quests should be accompanied by a five point Likert scales going from agree to disagree, allowing the testers to mark their agreement with each statement. These statements should concern the differences between the quests such as placement, NPCs/creatures/items involved and tasks required, as well as the amount of choice given to the user and if the quests fit with the area they were placed in.

Lastly the testers will be asked for the general feel of the quests in the prototype, using a semantic differential scale showing to ascertain if the quests were understandable, varied and felt different overall or if they were confusing, repetitive and too alike.

4 The Iterative Process

The following chapter explains the different steps of the iterative process the prototype went through; covering initial design and implementation of the prototype, and then goes on to the testing of the prototype leading to further iterations of the prototype including testing results, revised design requirements, and changes to the implementation of the prototype for each iteration. The structure of the chapters follows the guidelines set up in the methods chapter section 3.1, starting with an initial design and implementation based on the requirements from section 2.5.1. Each test conducted during the iterations follow the method described in 3.1.1.

4.1 First iteration – Initial Prototype

The following section describes the first iteration using the initial design requirements from section 2.5.1 to design and implement an initial prototype to conduct the first iteration test on and thereby revise the requirements.

Based on the initial design requirements the following things needed to be designed and implemented to have a prototype which could be used for the first iteration of testing. These were as follow:

- A search-based procedural generator
- Using three motivations and corresponding actions as quest structure
- Using enemies and items from the game as objectives
- Place quests in locations where enemies and items already exists

Furthermore, as this is not a standalone system a game in which to place the system needs to be selected. As such the following sections will go into depth with each of these aspects and explain the design choices made for each of them, how they were implemented. Afterwards the results from the iterative test for this step of the process are explained and the requirements revised.

4.1.1 The chosen game – an RPG

For the prototype a game was chosen for which the system should be placed in. The intention was to find a smaller game which could provide some of the same game mechanics a player would find in games such as World of Warcraft (Blizzard Entertainment Inc., 2017) and Final Fantasy XIV

(SQUARE ENIX CO., LTD, 2017), and as such the aim was to find a smaller role-playing type game (RPG). The game which was chosen was a personal hobby game attempt at an RPG done in Unity, using both owned paid Unity assets as well as free Unity assets, containing the following gameplay elements and mechanics:

- Player character with 3rd person movement and object interaction
- Player inventory and equipment through InventoryPro unity asset
- Three different NPC characters
- Six different enemies
- Several item and environment objects
- Game world with vegetation and structures through Gaia unity asset, including day and night cycle

The game was found to be easily extendable and mendable for the purpose of the project; as such it was deemed a viable candidate for a system such as the proposed one meant for this project.

The following Figure 5 show the game world environment found in the game, here showing the central location of the world (a small village surrounded by forest).

Figure 5 – Picture of part of the pre-created world present in the game, showing the village

With the game found creating the pieces the generator should be using was the next step of the design and implementation; starting with the quest structure.

4.1.2 The initial quest structure – three motivations

Three out of the nine possible motivations, from Doran and Parberry (2011), had to be chosen to build a structure upon and all of them seemed equally usable in terms of creating different quests from an abstraction of these motivations and derived strategies. As such the motivations using strategies which required actions which were already present in the game were picked, these being:

- Comfort
- Conquest
- Reputation

These motivations required a total of five actions to be created as an abstract, these being:

- Get/Steal (renamed 'Collect') - pick up one or more items within the game
- 'Give' – Hand over items to NPC
- 'Report' – Report findings
- Kill/Damage (named 'Kill') – Wound or eliminate an entity e.g. enemy
- 'Goto' – Move to some place

'Goto' was decided to be unnecessary due to the fact that the game world is not very big, and therefore having a step telling you to go to a specific location seemed excessive when each location would be relatively close to the village. The above mentioned actions were implemented under the name quest. These quests were made up of two parts, the class itself and the quest definition which it extended e.g. CollectQuest and CollectQuestDefinition for the action Collect.

The definition for each quest is the underlying requirements for a quest, such as what item and amount of that item a player would need to collect for a CollectQuest, as such defining the quest parameters. The quest definition is also responsible for creating the quest. This can be seen from Figure 6, showing how the quest parameters are defined in the constructor of the CollectQuestDefinition along with the title and description of the quest.

```
using ...

public class CollectQuestDefinition : QuestDefinition
{
    // Use Motivation's Strategy to define the quest context.
    public int count;
    public string itemName;

    /// <summary> Define the quest's context in the constructor.
    public CollectQuestDefinition(string itemName, int count, string title, string description)
    {
        this.count = count;
        this.itemName = itemName;
        this.title = title;
        this.description = description;
    }

    /// <summary> Create a new quest using this definition.
    override public Quest Create()
    {
        return new CollectQuest(this);
    }
}
```

Figure 6 – Code snippet of CollectQuestDefinition

The quest itself, which is the actual quest, is in charge of tracking when the quest is active (started), completed and any progress made towards the objective of the quest, such as when an item of the right sort is collected. The _Start() and _Complete() methods from CollectQuest seen in marks the quests as respectively started or completed and adds or remove a listener for any items being collected; this can be seen in Figure 7.

```
    /// <summary> Assign the quest as STARTED and start tracking collection of this item.
    override protected void _Start()
    {
        base._Start();
        Item.onAnyItemCollect += OnAnyItemCollect;
    }

    /// <summary> Assign the quest as COMPLETED and stop tracking collection of this item.
    override protected void _Complete()
    {
        base._Complete();
        Item.onAnyItemCollect -= OnAnyItemCollect;
    }
```

Figure 7 – Code snippet of CollectQuest's _Start() and _Complete() methods.

Whenever an item is then collected the methods OnAnyItemCollect() will run and compare the name of the item collected to the item required for the quest. If it is the correct item collected the current collected item count will increase, and the quest will complete if the count reaches the amount needed for the quest. Otherwise nothing happens. This can be seen from the snippet in Figure 8.

```
30      /// <summary> Attempts to collect an item for the Quest. If item counts exceeds requirements, complete the quest.
34      private void OnAnyItemCollect(InventoryItemBase item, ItemCollectionBase inventory)
35      {
36          // If the item collected does not match the desired item, return.
37          if (!item.name.Equals(definition.itemName))
38          {
39              return;
40          }
41
42          // Get how many of an item the player has in its inventory.
43          int itemCount = inventory.FindAll(item.ID).Length;
44
45          // If a sufficient amount has been acquired, complete the quest.
46          if (itemCount >= definition.count)
47          {
48              Complete();
49          }
50      }
```

Figure 8 – Code snippet of OnAnyItemCollect method from CollectQuest script.

Similar scripts were created for ‘Give’, ‘Report’ and ‘Kill’, these can be found through the directions laid out in the appendix section 9.5.

In addition to this, there existed a set of strategies which would be able to complete the motivation of the NPC, each of which was showed through a string of actions, here quests. These actions where intended to be the steps within the quest the player needed to take in order for them to complete the objective of the quest. As such depending on the strategy the actions a player needs to take to complete the quest would differ e.g. a strategy for “Obtain luxuries” would require the actions ‘Get’, ‘Goto’ and ‘Give’. With the removal of the ‘Goto’ action, this also meant the removal of the strategy in Comfort about visiting dangerous places as this strategy was primarily based upon touring the world to specified places. As such the following abstract strategies were derived (see full list in appendix 9.4):

- Deliver – ‘Collect’ and ‘Give’
- Hunt – ‘Kill’
- Bounty – ‘Kill’ and ‘Report’

Each strategy had the purpose of creating a complete quest sequence using the given quest fragments, while also ensuring that each quest have the needed variables to successfully track its completion. All this is done in the Setup() method part of any strategy. In Setup() objects is created in the game scene hierarchy for each quest fragment belonging to a strategy, in the order they need to be completed; meaning in a Deliver strategy a player cannot complete the ‘Give’ quest before the ‘Collect’ quest part of the strategy is completed should it happen the player have the

required objects. This was done by putting the fragments in serial using a small separate script, SerialQuestDefinition, which ensure any objects which is a child of this SerialQuestDefinition object have to be completed in the order they appear as children, as seen in Figure 9.

```
21          // Get the main object containing the list of quest objects.
22          GameObject questContainer = GameObject.FindGameObjectWithTag("Quests");
23
24          // Create a new gameobject that holds the new quest and make it a child of the quest list.
25          GameObject newKillQuestSeries = new GameObject();
26          newKillQuestSeries.transform.parent = questContainer.transform;
27          var serialDefinition = newKillQuestSeries.AddComponent<SerialQuestDefinition>();
28          serialDefinition.title = questTitle;
29
30          // New quest objects must be children of the serial quest definition.
31          GameObject newKillQuest = new GameObject();
32          newKillQuest.transform.parent = newKillQuestSeries.transform;
33          var killDefinition = newKillQuest.AddComponent<KillQuestDefinition>();
```

Figure 9 – Setup of quests as object in the scene hierarchy applying them as children to the SerialQuestDefinition

After the creation of the quest fragment object in the scene, the strategy assign the parameters to the quest definition. In Figure 10 can be seen how the Hunt strategy assign a title, name of the enemy target, and the amount a player needs to kill to the KillQuestDefinition which is part of the strategy.

```
35          // Set up kill quest details.
36          if (killAmount == 1)
37          {
38              killDefinition.title = "Kill a " + killTarget;
39          }
40          else
41          {
42              killDefinition.title = "Kill " + killAmount + " " + killTarget + "s";
43          }
44          killDefinition.targetName = killTarget;
45          killDefinition.count = killAmount;
```

Figure 10 – Defining parameters for a KillQuest with title, target and count.

With the abstract structure of quests and strategy completed all the only thing missing was implementing the motivations, with a separate script for each motivation which should extend a basic motivation class; an example of such a script for the conquest motivation which can be seen in Figure 11. The motivation itself contains lists of strategies and the strategies' descriptions, initially just used to signify what quest was currently active; these were later changed to signify the written text used for the quest log to show the primary quest objective (see the second iteration section 4.2.1 regarding the addition of the quest log). In addition to this the motivation itself also had the just of assigning itself an id depending on the given motivation which it is, in Figure 11 assigning itself as a Conquest motivation in the SetUP() method. This id is later used in

the generator (for more info see the later section on the creating of the quest generation in section 4.1.5).

```
using ...

public class ConquestMotivation : QuestMotivation
{
    public ConquestMotivation()
    {
        availableStrategies = new List<string>
        {
            "Hunt",     //kill
            "Deliver"   //get give
        };
        strategyDescriptions = new List<string>
        {
            "Achieve conquest by killing AMOUNT ENEMY",
            "Steal AMOUNT valuable ITEM for the NPC in the village"
        };
    }

    public override void SetUp()
    {
        motivation = (int)Motivations.Conquest;
    }
}
```

Figure 11 – ConquestMotivation with strategies and strategies' descriptions

4.1.3 Entities – NPCs, Enemies and Items

With the complete setup of the underlying structure of motivations, strategies and quests the next missing pieces was filling these quests with objects and entities which the player would need to interact with. The game came with a selection of NPCs, enemies and items and the following section describes the decision made in relation to these entities and objects.

For the NPCs there were only three available: a male wood elf, a female villager and a small unarmoured creature. These were decided to be static entities which the player would usually need to interact with at the end of the quest in relation to the 'Give' and 'Report' quest actions. Therefore all NPCs were placed within the confines of the small village enclosure found in the center of the map which came pre-generated with the game (see Figure 12).

Figure 12 – Picture showing the three NPCs in the village enclosure

For the enemies there were six possible enemies which came with the game. However, since there were only three motivations using three different abstract strategies, with only two strategies requiring an enemy, it seemed excessive to use all of them, and two out of the six were chosen these being: an imp and a skeleton (see Figure 13 and Figure 14).

Figure 13 & Figure 14 – Imp and Skeleton

For the items usable there were an extensive amount of possible items which could easily be added into the Item Database provided by InventoryPro. For the purpose of the project three categories of items were created in InventoryPro, these being:

- Tools
- Valuables
- Supplies

These categories contained a range of items of different varieties. Tools were created to contained equipment and usable tools such a shovels and hammers. Valuables were made to contain artefact type object which might be considered valuable such as vases. Supplies were intended to contain materials which could be gathered such as logs, iron bars and sacks of grain.

An example of these items within the game can be seen in Figure 15, showing two vases from the valuable category.

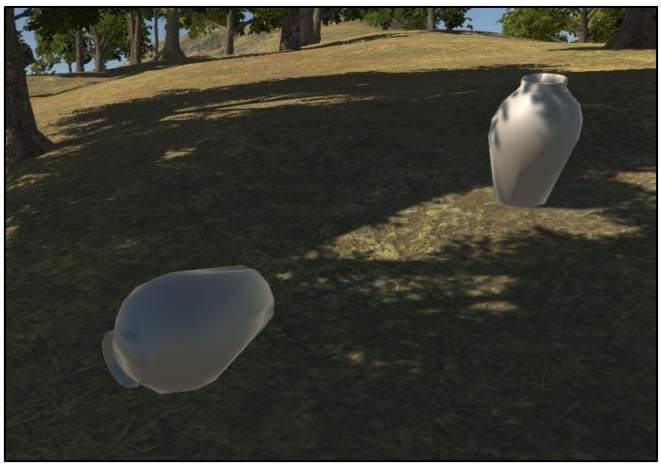


Figure 15 – Vases placed within the game world

4.1.4 Location – Finding and placing enemies and items

With both the structure and the objects and entities created, all there is left to do before creating the generator itself is finding a way to define locations wherein the generated quests should take place. It was decided that the generated quests should happen in locations where the objects or enitites where already found, seeing as this made the most logical sense as you would be more likely to encounter imps near an area where imps already were present. As such enemies and items were pre-placed in several locations around the world, surrounding the village. The enemies were places in groups creating camps of a bunch of the same type of enemy, with two camps for each type of enemy. The items were placed in a location for each type of item, such as tools would be found around the village and valuables and supplies would be found in the surrounding forest.

With the enemies preplaced the generator would be in charge of selecting a location based on a selected item or enemy used in the quest. Thereafter the generator should then be in charge of making sure there are enough of that type of enemy or item present and if not spawn the missing amount of enemies or items at that location.

4.1.5 Procedural generator – generating the quest

With all aspects needed for the generator defined, the last thing missing is the generator itself. The generator class itself is found in the QuestGenerator script.

At start-up, Start() (see Figure 16), the generator loads the all prefabs of items, objects, and enemies into memory, and creates a list of which motivations are currently in use. After this the start-up calls the fuction CheckMinimumQuests().

```
42      // Use this for initialization
43      void Start()
44      {
45          questLog = GameObject.Find("QuestLog").GetComponent<QuestLog>();
46          spawner = GameObject.Find("SpawnZones").GetComponent<EntitySpawner>();
47          environmentPrefabs = Resources.LoadAll<GameObject>("Prefab/EnvironmentObjects").ToList();
48          enemies = Resources.LoadAll<GameObject>("Prefab/Enemies").ToList();
49          boss = Resources.Load<GameObject>("Prefab/Boss/flesh golem");
50          NPCs = GameObject.FindGameObjectsWithTag("NPC").ToList();
51          motivationsInUse = new List<int>();
52          CheckMinimumQuests();
53      }
```

Figure 16 – The Start() method in the QuestGenerator script.

When a quest is created they become child objects of the game object in the scene which contains the QuestGenerator script. The CheckMinimumQuests() method (seen in Figure 17) checks if the child count of this QuestGenerator object is less or equal to the minimum of amount of quests which should be available at all times; if the child count is smaller or equal a new quest is generated and it rechecks the child count requirement until it returns fall, where after the generator stops generating quests.

```
55      public void CheckMinimumQuests()
56      {
57          if (transform.childCount <= minimumQuests)
58          {
59              CreateQuest();
60              CheckMinimumQuests();
61          }
62      }
```

Figure 17 – Code snippet of CheckMinimumQuests() script checking the child count to trigger quest generation.

When a quest is called to be generated through the CreateQuest() method (see Figure 18), this method checks for motivations currently in use to find one which a quest currently does not use. This was done to ensure there would not be duplicate motivations leading to duplicate strategies, which was reasoned to ensure variety in the quests generated. The generator gets a list of available motivations through the GetAvailableMotivations() method, then chose a random

motivation from this list through the GetMotivation() method. After the motivation is setup is assigned a id through the SetUp() method from motivations (see Figure 11).

```
64      public void CreateQuest()
65      {
66          // Get list of available motivations.
67          List<int> availableMotivations = GetAvailableMotivations();
68          if (availableMotivations.Count == 0)
69          {
70              return;
71          }
72          // Get the Motivation object from the list of available motivations.
73          QuestMotivation motivation = GetMotivation(availableMotivations);
74          motivation.SetUp();
75          string strategyName = motivation.GetRandomStrategy();
76          Debug.Log("Starting (" + motivation.motivation + ") " + strategyName + " quest.");
77          string strategyDescription = motivation.GetDescriptionForStrategy();
78
79          // Generate the objectives for the Strategy.
80          QuestStrategy strategy = GenerateStrategy(strategyName);
81          strategy.questTitle = strategyDescription;
82          motivationsInUse.Add(motivation.motivation);
83          strategy.SetUp();
84      }
```

Figure 18 – Code snippet for CreateQuest() method in QuestGenerator.

When the motivation is setup the a random strategy from this motivation is chosen and its description is set to the string available for the selected strategy. With the description assigned to the strategy the generator starts generating the quest fragments from this strategy by calling the method GenerateStrategy() method. The motivation is marked as being in use, so no further quests will use this motivation until this motivation is marked as not in use. Then the strategy is setup using the strategy's SetUp() method.

The GenerateStrategy() method makes a call to the selected strategy's AssembleStrategy() method. An example of a AssembleStrategy() for the 'Hunt' strategy which method is named AssembleHuntStrategy() can be seen in Figure 19. In the AssembleStrategy() method randomized numbers for the entities needed in the quest fragments of the strategy is found, such as amount of enemies and an enemy type for the 'Hunt' strategy. These parameters are found using a random number generator for each parameter.

When the parameters are found these are assigned the Strategy object pertaining to the particular strategy which can then be utilised by its associated quest fragments.

```
319
320     private HuntStrategy AssembleHuntStrategy()
321     {
322         HuntStrategy huntStrategy = new HuntStrategy();
323
324         GameObject selectedEnemy;
325         int spawnAmount;
326
327         //Find amount of enemies to spawn
328         System.Random randomKill = new System.Random();
329         int randomKillCount = randomKill.Next(1, 10);
330
331         if (randomKillCount == 1)
332         {
333             selectedEnemy = boss;
334             spawnAmount = randomKillCount;
335         }
336         else
337         {
338             // Select a random enemy
339             System.Random random = new System.Random();
340
341             int enemyRandom = random.Next(enemies.Count);
342             selectedEnemy = enemies[enemyRandom];
343             spawnAmount = randomKillCount + 2;
344         }
345
346         List<GameObject> spawnedEntities = spawner.GenerateEnemyEntities(selectedEnemy, spawnAmount);
347         huntStrategy.zone = spawnedEntities[0].transform.parent.transform;
348
349         // Pick a random NPC
350         int randomNPC = UnityEngine.Random.Range(0, NPCs.Count);
351         GameObject NPCobj = NPCs[randomNPC];
352
353         // Spawn the randomly selected item as quest item.
354         huntStrategy.questTitle = "HuntStrategy";
355         huntStrategy.killAmount = randomKillCount;
356         huntStrategy.killTarget = selectedEnemy.transform.name;
357
358         return huntStrategy;
359     }
360
```

Figure 19 – Code snippet of the AssembleHuntStrategy() method.

4.1.6 Testing and comment on first iteration prototype

After the conclusion of the implementation of the initial prototype, the first test was conducted. The test was done on two groups of respectively five and six people, all males. These two tests were done physically in the home of one of the testers, one from each group.

For the part of the test where the participants were playing through the game, one by one, two key observations were made by all the testers:

- The users had problems remembering how far they were in completing the quests
    - Several users noted an easy way of tracking of the quest progress was missing
- Several noted finding bugs connected to progress not counting correctly.
    - Some noted they were unsure if it was a bug, or just them not knowing how far they were.

After the playthrough had concluded the test proceeded to the post-it-note part of the test. During this the following notes were the general consensus from both groups:

- Game was nice, but simple. It was good.
    - *"It was nice seeing a game that did not overshadow the mechanics"*
- With no quest tracking it was hard to remember more than one quest at a time, and this gave very little feeling of choice.
    - *"Last quest you are given, you just complete. You still remember it"*
    - *"Were there more than one quest available at a time, I was not aware"*
- Felt hard to comment on difference in quests when they did not feel they had a choice in which ones to do.

4.1.7 First iteration – new design requirement

From the findings of the first test, it was obvious there was a crucial element which had been forgotten in the creating of the prototype; a quest log or quest tracker. Being initially given around four quests, it was hard for the testers to remember the objectives of all quests, and they generally just completed the last one they were given. As such they felt they had no choice in what quests to complete and it was very random whether or not the quests felt different, it was all chance. Therefore, to ensure the players actually realise they have a choice of several quests at a time, give the players a way to track what quests are available and how far they are is clearly needed. Therefore the following requirement was added to the overall list of design requirements for the prototype (see full list in appendix section 9.1):

11. Game should have a quest tracker or quest log, which show all available quests and their progress.

4.2 Second iteration

From the new design requirement found from the first iteration (see section 4.1.7) it decided that a sort of quest tracker or quest log for all available quests should be present in the game. Hence, the following section describes the design and implementation of this part of the system, and the accompanied test of the prototype afterwards leading to revised requirements.

4.2.1 The quest log – design and implementation

The primary addition needed for the prototype after the first iteration test was the need for a quest log. For this quest log there were a couple of consideration, these were as follow:

- A quest description – explaining the objective
- Quest progress – showing how far the player is with an objective e.g. 0/8 enemies killed

With these considerations in mind the QuestLog class was created. This class should be in charge of adding quests to a visual element in the game canvas (the log itself), and making sure the log was updated when progress is made, and removing quests when a quest is marked as completed. This structure can be seen in Figure 20.

```
6   public class QuestLog : MonoBehaviour {
7
8       public GameObject questLog;
9       public GameObject questContainer;
10      public List<GameObject> entries;
11
12      void Start()
13      {
14          entries = new List<GameObject>();
15          questLog = gameObject;
16          foreach (Transform child in questLog.transform)
17          {
18              if (child.name == "QuestContainer")
19              {
20                  questContainer = child.gameObject;
21              }
22          }
23      }
24
25      /// <summary> SetUpQuestLog for a QuestStrategy object.
29      public void SetUpQuest(QuestStrategy strategy)...
43
44      /// <summary> SetUpQuestLog for a Quest object.
48      public void SetUpQuest(Quest quest)
49      {
50          Object questEntry = Resources.Load("Prefab/UI/Quest");
51          int posY = entries.Count * -45;
52          GameObject entry = (GameObject)Instantiate(questEntry, new Vector3(0, 0, 0), Quaternion.identity);
53          entry.name = quest.id;
54          entry.transform.SetParent(questContainer.transform, false);
55          Vector3 pos = new Vector3(0, posY, 0);
56          entry.transform.localPosition = pos;
57          entries.Add(entry);
58
59          Text text = entry.GetComponentInChildren<Text>();
60          text.text = quest.definition.title;
61      }
62
63      public void UpdateQuest(string id, string newText)...
72
73      public void RemoveQuest(string id)...
90
91      public void Rearrange()...
99
100 }
```

Figure 20 – The QuestLog() class script.

4.2.2 Second iteration testing

With the quest log implemented the second round of testing on the prototype commenced. This round of testing was conducted on two groups of five, with two females in one of the groups; all participants being members of the same guild in World of Warcraft. This test was done online using a voice communication and chat software called Discord, provided by the testers who use it for communication with their guild mates during in-game play times and out-of-game sessions.

For the playthrough of the game, the testers were given a link to an online version of the game, and they played it simultaneously while talking and asking questions using the voice chat in the software. The general comments were as follow:

- Easy to play. Easy to understand.
- Very simple. Can be a good thing.
    - *"Simple is good if that is what I was going for"*

To ensure anonymity and privacy in terms of the post-it notes, this part of the process were converted into private messages in place of post-it notes; the participants would send me a private message with each of their three comments, where after the rest of the session proceeded in the same manner as described in the methods chapter section 3.1.1. They had nothing new to note on the general game feeling. However for the quests their comments were:

- Quests feel very alike.
    - *"Even with four quests up, you can already see the overlap of similar quests"*
    - *"I'm sure more types of quests could be made, I got the same ones all the time"*
- Easy to predict what quest will happen after just a few minutes.
    - *"Seems the pool of quests is very small, make it bigger, because it got very boring after a couple of minutes. Repeating the same again and again"*
    - *"I wanted to stop playing after three minutes. It was very repetitive"*

4.2.3 Second iteration - Revised requirements

With the conclusion of the second tests, the overall thing to take form the test was the amount of different quests which could be generated. Going of three different motivations and their underlying structure, did not give the testers a sufficient pool of possible quests, it became

repetitive and boring very fast. Furthermore, the testers felt the quest felt overly similar, like there was not enough difference in the underlying structure. As such adding more motivations and their underlying sequences of actions the player needs to take could help solve this problem. Using six different motivations instead of three will increase the pool of possible quests and strategies and thereby overall quest types the generator can create. The previous requirement was updated to reflect the change in design (see full list in appendix section 9.1):

3. Use a minimum of six different motivations with underlying strategies and sequences of actions as quest structure, as theorised by Doran and Parberry (2011).

4.3 Third iteration

With the revised requirement from the second test on the prototype, more motivations and underlying quests and strategies needed to be added to the generator. The following section explains the design choices behind each of the quest actions and strategies for each of the three new motivations. Thereafter explains the third test conducted on the prototype and the new design requirements found from it.

4.3.1 Three new motivations – design and implementation

Requiring adding another three of the remaining six motivations proceeded with the reasoning behind the first set of motivations; these three new motivations were chosen based on the requirement of the least amount of changes needed in order to add the motivations, strategies and quest fragments, while gaining a big section of new types of quests to add variety to the quest generation pool as commented was needed to keep the quests from seeming repetitive. As such the following three motivations were chose:

- Knowledge
- Serenity
- Protection

The motivations had a bunch of new actions which could be turned into quest fragments. However, several of the actions required mechanics which the game did not allow as such ‘Spy’, ‘Escort’, ‘Listen’, ‘Defend’ and ‘Repair’ were decided to be excluded leaving the following two actions to be made into quest fragments:

- Use – Use an item on an object
- Take (mechanically similar to 'Collect' and therefore taken as such)

Through the addition of these new quest fragments the following new strategies could be created through all the pieces of quest fragments, these being named as follow:

- Use – ' Use'
- Obtain – 'Get' and 'Use'
- UseItem – 'Get', 'Use' and 'Give'

With the definition of these new quest fragments, the definition of the strategies derived from the motivations using these respective quest fragments the implementation could be done similarly to the implementation previously described when implementing the initial three motivations, quest fragments and derived strategies (see earlier in this chapter section 4.1.2).

4.3.2 Third iteration testing

With the new motivations added into the generator the third test of the prototype could be conducted. This test was done on two groups, one group of five males and one group with four males and three females. This round of testing was done in-person. The testers played the game and had no initial comments or outburst during their play session. For the post-it note session, their comments were scarce, mostly liking the game and calling it good. However, for the quests they all agreed on one thing:

- Placement of quests were too predictable
    - *"Quests was always placed where you would have seen the creature or object before"*
- Unpredictable placement helps with variety
    - *"What about having quest spawn with objectives far away from other of that creature type, and then you know it's important because the creature have moved there suddenly. I wouldn't expect that"*

4.3.3 Third iteration - New requirements

The things to take away from this round of testing were the predictability of the quest spawn areas. The quests always spawned in areas where the object or enemies could be found, and not

away from these areas. They meant the spawn areas lacked variety and made the quest seem to alike due to predictability. As such the generator should ensure that there are always more than one possible position, or have a more loose restriction on how far from the other instances of this creature it can place the quest. These suggested changes were coined into the following new requirements for the design requirements and added to the list (full list can be seen in appendix section 9.1):

12. Ensure minimum of two position for each quest
13. No restriction of placing quests in areas where the object is already found.

4.4 Fourth iteration

With the addition of two new requirements for the prototype in relation to finding locations for the quests to take place in, the following section describe the design and implementation of this new location finder added for the generator. The remaking of the location finder also means changing the way enemies and objects are spawned into the scenes, the implementation of this new system After implementation of this new location finder the results from the fourth test on the prototype is described and the revision of a design requirement explained.

4.4.1 New quest spawn location finder parameters

With the feedback and new requirements gotten from the third iteration test it was important to completely overhaul the way locations were found and creatures spawned. The following designs were made on the basics of creating unpredictability through randomised spawn locations. As such the following specifications were decided in terms of creating a randomised spawn location finder. These specifications were as following:

- Find a random location based on pre-created points
- Minimum number of pre-set locations should be number of motivations + 2
- Locations should be marked as occupied of a quest is already happening within
- Location will not require enemies to be present beforehand

Through these specifications the methods FindAvailableZones() and PickRandomZone() within the EntitySpawner script were created (see Figure 21). These methods have the purpose of going through all available zones and creating a list of zones which are not currently occupied by any

enemies or items. After finding a list of available zones, a random one is picked. All zones are pre-created and pre-places game objects in the scene hierarchy in unity, all of these zone locations are put into a list which the FindAvailableZones() method iterate through and look at whether or not any of these zones have any child objects; a zone will have child objects if a quest is currently taking place, as quest object, items and/or enemies are spawned as children of a zone at the creating of the quest (see updated script for spawning items, objects and enemies in next section, 4.4.2). The available zones are put into a list of available zones which can be used by other methods. The method PickRandomZone() use this list of available zones to pick a random unoccupied location to be used by the generator and EntitySpawner.

```
25      /// <summary> Get all available zones (zones with no entities in them).
28      private void FindAvailableZones()
29      {
30          // Check each zone.
31          foreach (Transform zone in zones) {
32              // If the zone has children, remove it from the available list.
33              if (zone.transform.childCount > 0) {
34                  availableZones.Remove(zone);
35              }
36              else {
37                  // Otherwise add the zone as available, if it isn't already.
38                  if (!availableZones.Contains(zone)) {
39                      availableZones.Add(zone);
40                  }
41              }
42          }
43      }
44
45      /// <summary> Returns a random Transform from the Zones list.
49      private Transform PickRandomZone()
50      {
51          // Find available zones.
52          FindAvailableZones();
53          // Returns a random zone from the zones Transform list.
54          var random = UnityEngine.Random.Range(0, availableZones.Count);
55          return availableZones[random];
56      }
```

Figure 21 – Methods for finding available zones and picking a random zones to use as quest location

4.4.2 Enemy and object spawn

With the change of the location finder, the way enemies are spawned and handled had to be updated. Since the location of quests are now more unpredictable with randomized locations, enemies and objects can no longer be pre-placed and have to be spawned when the quest is generated at a certain location. As such the only specification for the EntitySpawner is as follow:

- Objects and enemies will be spawned in the location to match the amount needed for the quest at the location.

As such when spawning enemies the generator will tell the EntitySpawner which GameObject (prefab of an item, enemy or object) it needs to spawn and the amount of said GameObject. These parameters are parsed into each methods for each type of object. An example of this can be seen in Figure 22 which shows the method of spawning enemy entities.

```
122     public List<GameObject> GenerateEnemyEntities(GameObject prefabTemplate, int amount)
123     {
124         Transform zone = PickRandomZone();
125         List<GameObject> enemies = new List<GameObject>();
126 
127         for (int i = 0; i <= amount; i++)
128         {
129             float directionFacing = UnityEngine.Random.Range(0f, 360f);
130             Vector3 point = (UnityEngine.Random.insideUnitSphere * spawnRadius) + zone.position;
131             GameObject entity = Instantiate(prefabTemplate, point, Quaternion.Euler(new Vector3(0f, directionFacing, 0f)));
132             entity.transform.parent = zone.transform;
133             enemies.Add(entity);
134         }
135 
136         return enemies;
137     }
138 }
```

Figure 22 – Method for spawning enemy entities

When spawning the enemies first a zones is picked, thereafter the enemies of the selected type is spawned randomly around the given zones area and added to a list for later use.

4.4.3 Fourth iteration testing

With the implementation of the improved location finder and entity spawner the fourth iteration test could be commenced. For the fourth test the participants were two groups of six, all males. This round of testing was done online using their guild's server on the voice communication and chat software Discord, where the testers were linked to an online version of the game.

These groups of testers were very talkative and very similar opinions in terms of the quests, both during play and after during the post-it note session (again done through private messages). These opinions were as follow:

- Quests can easily be the same model, but be different due to themes or just tasks. Themes being involved creatures or objects in correlation with tasks.
- Different enemies/object helps with variety. Same enemy each time gets boring with the same task.
- Same objective should not always concern same enemy, game might need more enemy types to widen variety.

The following quote sums up all of these comments very well, which were uttered by one of the participants where the rest of the group quickly agreed:

"I can kill enemies all day, the task doesn't matter. I don't mind grinds. I just don't want to grind skeletons all day every day, and in every task"

4.4.4 Fourth iteration – revised requirement

Seeing as the general consensus from the testers was that the amount of enemies in this iteration could benefit from more variety as number of different types did not quite fit did not fit with the amount of different tasks all concerning these two types of creatures, lead to the obvious solution of adding more enemies to the pool.

14. Use all six enemy types as part of the objectives.

4.5 Fifth iteration

With the revised requirement from the fourth iteration test, the simple task of adding more enemies to the pool of possibilities were at hand, as such the following section describes the design process for deciding these enemies and the fifth iteration test leading to the revised requirements needed to create the final revision.

4.5.1 Addition of enemies

For the simple task of adding more enemies, the game already came up with six enemies so all of the remaining enemies were decided to be added into the game these were as following:

- A flesh golem
- A grey goblin ranger
- An armoured monster
- A red goblin scout

All but the flesh golem was simply put into the previous pool of enemies which the generator could take from. However, to gain more variety in the type of enemy spawned a special requirement was added for the flesh golem. As many games have special boss creatures the player can fight during quests it was reasoned that the flesh golem could fit this role; as such the

following specification for what type of enemy would be chosen by the generator was described as follow:

- If the amount of enemies to kill is 1, a boss monster will spawn (here a flesh golem)
- On any other amount of enemies to kill spawn a random of any of the five remaining enemies.

These specifications were added into any quest which requires enemies being killed such as KillQuests.

4.5.2 Fifth iteration testing

For the fifth and final iteration, the testing was done online of two groups of respectively six and seven, one group having one female all others being male. This round of test was done using the voice communication software and chat called Skype.

During the testers playing the game, several noted that they would not want to play the game for extended amounts of times. It is fun and simple for a short while, but not extensive play; generally contributing it to the quests. During the post-it note session (done by them sending me a private chat message) the commented the following:

- Good variety for a short game. Enemies and objectives are fine.
- Should be more quests than the average player can complete daily.
- Content should be exponential to the average use, for extended use.

4.5.3 Fifth iteration – Revised requirements for Final revision

The testers of the fifth iteration seemed pleased with the simple game, during a playtime of five to ten minutes. However, they noted that for extended play there should be more to do, more quests and more variety. As such for the final revision, adding the last three motivations could be a vital addition before the final test. As such the following requirements have been revised:

3. Use all the different motivations with underlying strategies and sequences of actions as quest structure, as theorised by Doran and Parberry (2011).

The complete list of design requirements used for the final prototype can be seen in the appendix section 9.1.

4.6 Final revision

From the fifth iteration test the comments were generally long time use of the generator and extend play sessions; to combat this the requirement were revised to encompass all of the different strategies, the following section describes the definition of these added motivations.

4.6.1 The final three motivations added to generator

Reaching the end of the allocated time for the project, and basing on the feedback gotten from the fifth iteration test the addition of more motivations is not to gain additional types of quest fragments but rather up the flavour variety gained from motivations descriptive reasoning. As such only new motivations were added to the generator using only the strategies which had already been created, meaning no new quest fragments or strategy types based on these new quest fragments. As such the new motivations added were simply:

- Wealth
- Ability
- Equipment

Yet again these motivations were implemented similarly to the way the initial motivations were implemented (see section 4.1.2 earlier in the chapter).

5 Evaluation

Following chapter describes the evaluation of the final test conducted on the procedural quest generator prototype, using the testing method described in the method chapter 3.3. The chapter goes through where the test and survey were distributed, the responses received, and the demographics of the participants and if and how the quests generated felt different.

5.1 Survey distribution and responses received

The game prototype and survey were distributed to four World of Warcraft guild, one Final Fantasy XIV Free Company (their guild equivalent) and several Discord servers concerned with World of Warcraft; in addition to this the game prototype and survey were also publicised on several social media sites such as Facebook and Twitter with the intention of reaching out to my network of game developers and gamers.

From this distribution of the test 67 responses were received over the course of the test being available. The full responses can be seen in the appendix section 9.3.

The following sections go through the demographic distribution of the sampled players and try to place them within the target group. After either confirming or denying the sample as part of the target group, evaluation of the responses about the quests are evaluated. Lastly the overall feel of the quests from the semantic deferential scale at the end of the survey is evaluated.

5.2 Demographics

Analysing the results from the survey proved that the participants of the final test were similar in demographic distribution as were expected of the target group (explained in section 3.2), and as such the sample could be seen as representative of the target group. However, the given the small sample over generalisations cannot be made.

The genders representation were as expected favouring males being split 79% to males, and 21% females, as seen from Figure 23.

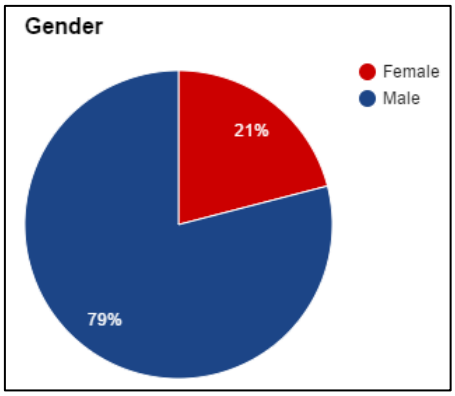


Figure 23 – Piechart of the distribution of gender in the sample. N=67

Furthermore, the ages of the participants ranged from 16 to 52 years old, with a median of 24 years old similarly to the findings of Nick Yee (2006). The age distribution can be seen in Figure 24.

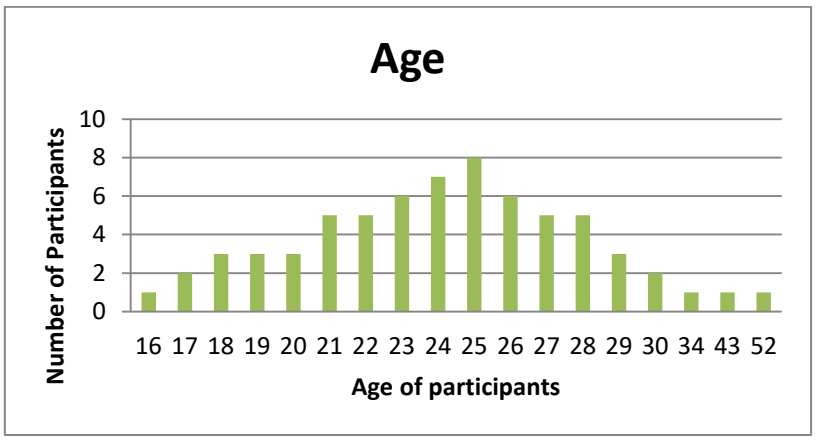


Figure 24 – Chart showing the distribution of ages in the sample. N=67

All participants in the sample answered that they play video or computer games, which was an expected outcome due to the distribution places.

5.3 The quests – feel different or alike?

Concerning the quests there where a long range of questions related to what during the iterative steps were described as having an impact on quest feeling different.

First of it was ascertained whether or not the participants could actually complete all the quests the generator put their way. 5 (7.5%) out of the participants were unable to complete every quest generated and as such had an impact on their overall choice and experience of the quests.

Afterwards all questions were statements with accompanying scales from 1 to 5 for disagree and agree respectively.

The first statement concerned whether or not the quests felt too similar (Figure 25), showed that 52.2% of the participants disagreed with the statements and 29.9% of the participant were somewhat disagreeing with the statement. As such it can be said that the quests were varied, meaning they had a wide selection of possible quests at its disposal.

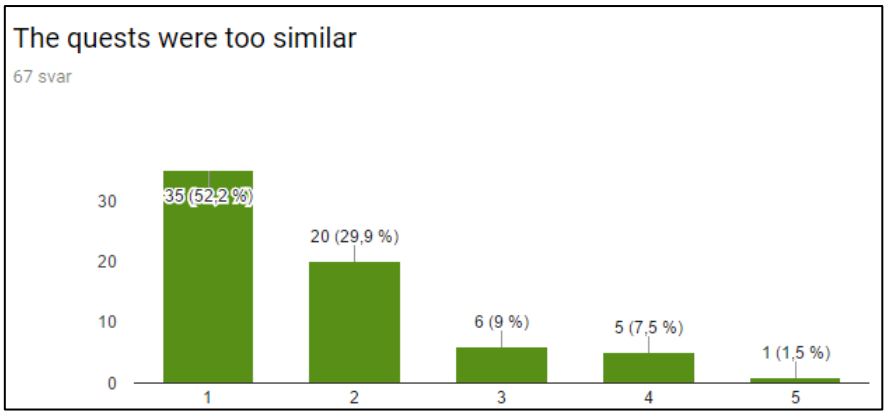


Figure 25 – Distribution of answers for 'The quests were too similar'.

In relation to whether there were many different tasks, different enemies and different items/object all three statements showed a similar distribution with agreement with each statement (see appendix 9.7.1 for these figures). Meaning the varied selection of quests could be contributed to these aspects of randomising within the generator. However, in relation to the NPCs involved in the 'Give' and 'Report' quest fragments, there were no agreement whether or not there were different NPCs involved in the quests, tilting slightly towards agreement but generally being evenly spread (seen in Figure 26)

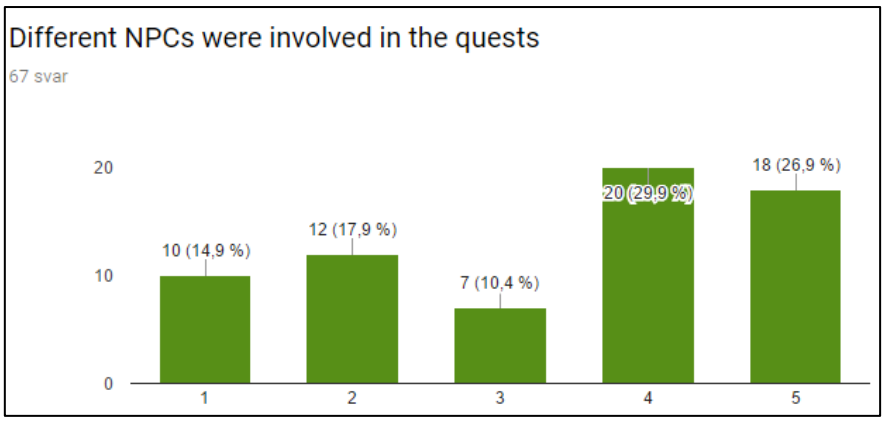


Figure 26 – Distribution of the answers to the statement "Different NPCs were involved in the quests".

Another important part of the survey is the fitness of the quests within the game environment, meaning whether or not the objectives felt out of place or the NPCs, enemies, item/objects did

not fit with the environment in which they were placed by the generator. As NPCs, enemies, objects and items came with the game wherein the procedural quests generator system was placed, it was expected that the quests would fit within any place within the game world. As such the results from the statement "The quests fit in their environment" (Figure 27) does not come as any surprise seeing as 61.2% of the participants have some degree of agreement with the statement, with 11.9% being neutral on the matter and 26.9% disagreeing.

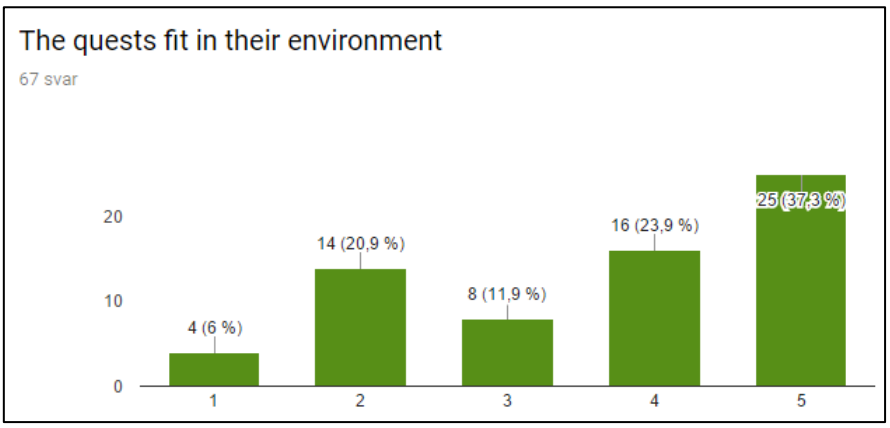


Figure 27 – Distribution of answers to the statement "The quests fit in their environment".

The two following statements concerned the locations of the quests which were generated and whether they were spread out and if the distance where too big. The intention of these questions were to ascertain whether or not the quests were spawned in clumped up locations and whether or not the distance between quest locations where too big. From Figure 28 it can be seen that the quests were spread out and not clumped up in one location, meaning this part of the location finder succeeded in finding suitable locations.

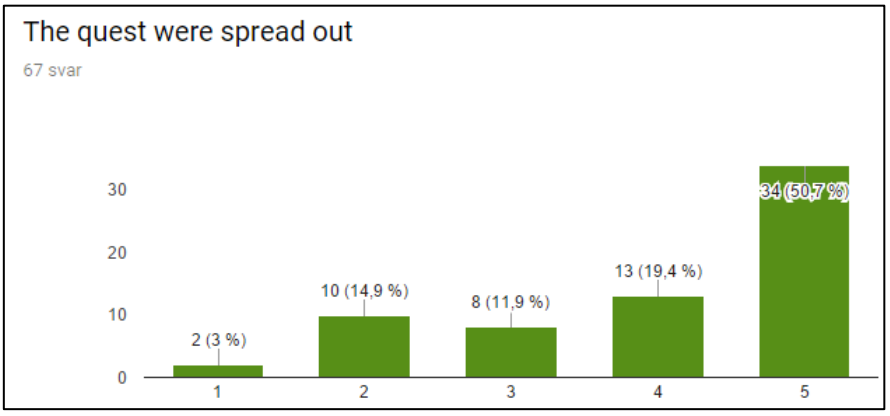


Figure 28 Distribution of answers to the statement "Quest were spread out".

However, in relation to whether or not the distance between quests locations were too big, the answers had no common group and were evenly distributed between agreement and disagreement (Figure 29).

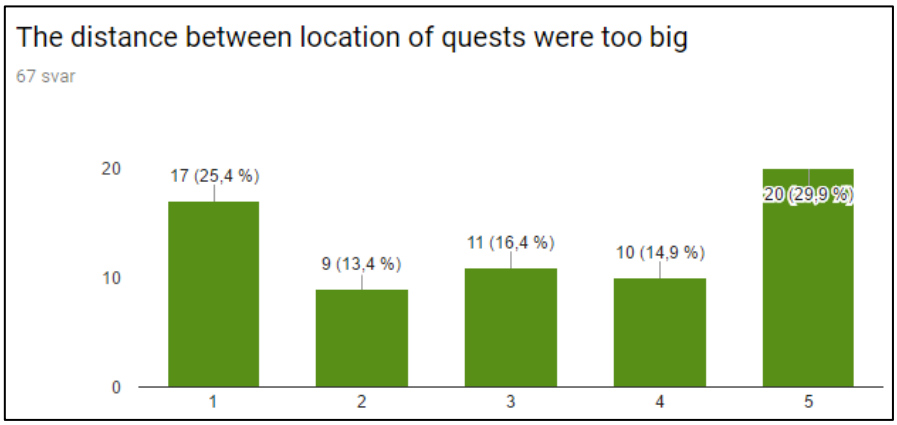


Figure 29 – Distribution of answers to the statement "The distance between locations of quests were too big"

This could be attributed with the fact that all players have a personal preference in relation to travel time between quests depending on the actions required when reaching the quests; meaning the quests could take more time getting between than the time it require the players to complete the quests. However, it could also be attributed to the fact that players prefer shorter travel times when "grinding"[3] quests, which could be associated with the session the participants were required to complete during the test.

The last two statements where intended to concern the variety the generator gave the player. This variety were exemplified through the amount of quests the players would have available to them at one time and how quickly new quests were made available; these factors would mean the player had a range of quests available to them and thereby a choice in what quest they wanted to complete when. From Figure 30 and Figure 31 can be seen the responses to the two statements: "New quests became available too slowly" and "There were not enough quests available at the same time".

For the statement about new quests being made available, there were 85.1% of the participants disagreeing with the statement, 10.4% somewhat disagreeing and 4.5% having no preference

[3] Grinding: a term used by gamers to signify completing repetitive actions or content such as completing a bunch of repeatable quests or killing a pack of enemies and waiting for them to respawn in order to kill them again and thereby repeating the action.

(Figure 30). From this it was clear that there were a good flow to the arrival of new quests being made available to the player.

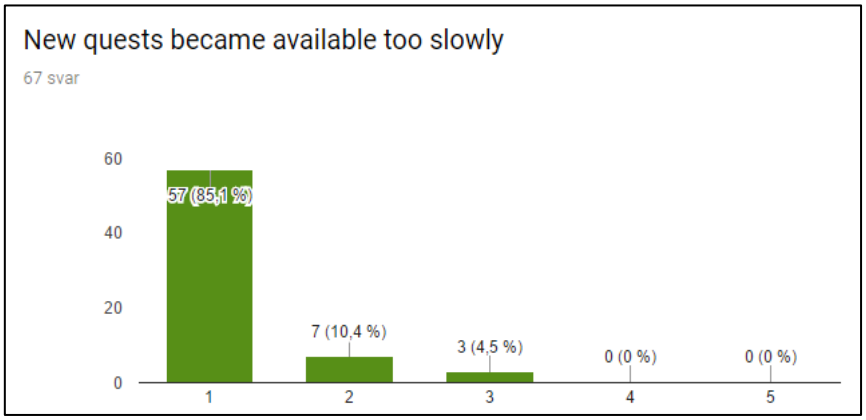


Figure 30 – Distribution of responses to the statement "New quest became available too slowly".

For the statement concerning the amount of quests being available at the same time, it was clear that the majority of the participants disagreed to some degree that there were not enough (44.8% disagreeing and 31.3% somewhat disagreeing), with a clear minority 14.9% agreeing with the statement to some degree (Figure 31).

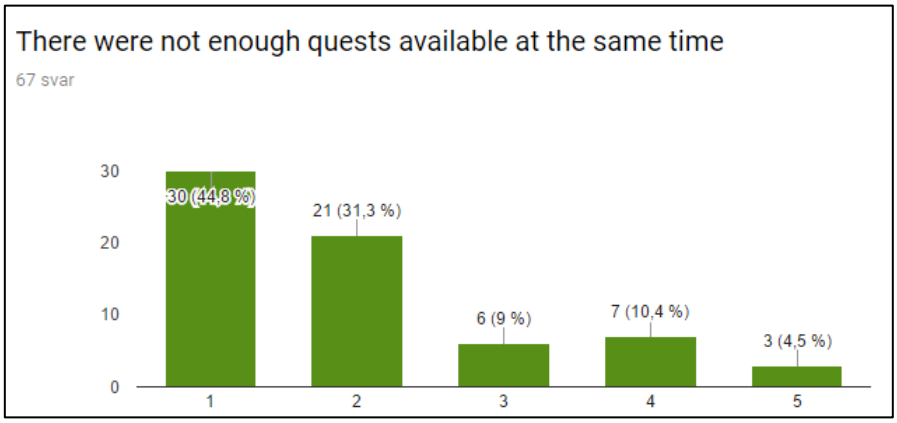


Figure 31 – Distribution of the responses to the statement "There were not enough quests available at the same time".

The last question in the survey was the semantic differential scale for the following terms: Alike vs Different, Confusing vs Understandable, and Repetitive vs Varied. For each set of these terms the testers could assign much of a respective term they agreed with. The Figure 32 shows the distribution of answers in terms of the left half of the semantic differential scale (Alike, Confusing, and Repetitive). The responses show that the general perception of the quests were that they felt different and varied, with most answers for those set of terms negated the left-hand column.

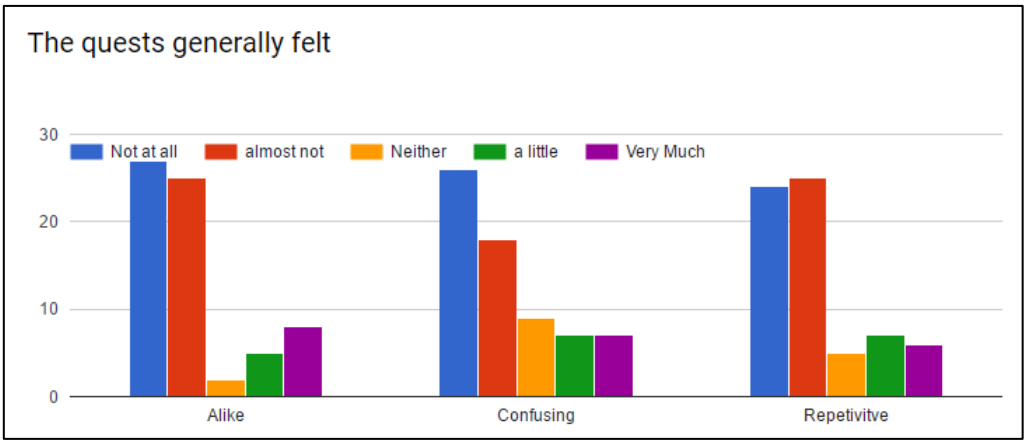


Figure 32 – Distribution of responses to the left-hand side of the semantic differential scale.

However, for the terms Confusing vs Understandable, the majority of the responses did find the quests understandable. However, 20% of the participants felt the quests were confusing. This confusion could be what made the participants unable to complete all the quests.

6 Discussion

The following chapter discuss the findings of the final test in relation to the research and analysis of previous work done within the field of procedurally generated content and procedurally generated quests (from chapter 2).

The overall intention of the project was to prove or deny if it was indeed possible to create a quest generator which upon repeated encounters of the same type of quests would feel different to the players; basing this upon the theory of motivations, strategies and actions as constructed by Jonathon Doran and Ian Parberry (Doran & Parberry, 2011).

The motivations and strategies gave the system a variety of different types of quests utilising different quest fragments, coined as actions by Doran and Parberry. This theory proved useful in the sense that the system was already quiet an abstract quest structure which would be further abstracted or made more specific using sub-quest types.

Furthermore the schemes described by Doran and Parberry also proved to be easily amendable in terms of adding more variation into procedurally chosen enemies and item or other entities each quest fragment needed. With a structure with proved both versatile and amendable, it can be construed that the decision to base the procedural quest generation system upon this theorised quest scheme was a success.

7 Conclusion

With the final problem statement coined as following:

"Can a procedural quest generator be constructed to create quests based on the same quest structure which feel different each time a player encounter them"

In terms of realising the problem set forth by the problem statement of creating quests using the same basic quest structure, and having repeated encounters of the same quest type seem different proved a general success. With only six different strategies based on nine motivations the chances of players encountering a quest using the same basic structure - the strategy setup using quest fragments derived from abstract player actions – is very high; as such seeing as the majority of the participants found the quests to be not similar, but on the other hand varied in terms of items, objects, and enemies. This variation in items, objects and enemies along with a select choice of NPCs and a randomly generated amount of times the actions part of the quest had to be executed meant two quests using the same structure could look widely different.

Even with the project being a very small scale procedural quest generator, the quests did not feel repetitive or confusing either, as attested by to the majority of the participants, who on the other hand found the quests different and varied and with understandable objectives to some degree.

8 References

Ashmore, C., & Nitsche, M. (2007). The Quest in a Generated World. *Situated Play, Proceedings of DiGRA 2007 Conference*, (pp. 503-509).

Bjørner, T. (Ed.). (2015). *Qualitative methods for Consumer Research.* Hans Reitzels.

Blizzard Entertainment Inc. (2017). *Diablo 3*. Retrieved from Diablo 3: https://eu.battle.net/d3/en/

Blizzard Entertainment Inc. (2017). *World of Warcraft*. Retrieved from World of Warcraft: https://worldofwarcraft.com/en-us/

CD Projekt. (2017). *GOG.com*. Retrieved from GOG.com: https://www.gog.com/

Chandler, H. M. (2014). *The Game Production Handbook* (3rd ed.). Jones & Bartlett Learning.

Doran, J. (2014). *Procedural Generation of Content for Online Role Playing Games.* University of North Texas.

Doran, J., & Parberry, I. (2010, May 12). Towards Procedural Quest Generation: A Structural Analysis of RPG Quests. *Dept. Comput. Sci. Eng., Univ. North Texas, Tech. Rep. LARC-2010*, p. 42.

Doran, J., & Parberry, I. (2011). A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs. *PCGames '11 Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (pp. 1-8). Bordeaux, France: ACM.

Doran, J., & Parberry, I. (2015). A Server-Side Framework for the Execution of Procedurally Generated Quests in an MMORPG. *GAMEON'15 - Proceedings of the 16th Annual European Conference on Simulation and AI in Computer Games*, (pp. 103-110). Amsterdam, The Netherlands.

Griffiths, M. D., & Davies, M. (2003, February). Breaking the Stereotype: The Case of Online Gaming. *Cyberpsychology & Behaviour, 6*(1), pp. 81-91.

Lazar, J., Feng, J. H., & Hochheiser, H. (2009). *Research Methods In Human-Computer Interaction.* Wiley.

Lee, Y.-S., & Cho, S.-B. (2012). Dynamic Quest Plot Generation using Petri Net Planning. *WASA '12 Proceedings of the Workshop at SIGGRAPH* (pp. 47-52). Singapore: ACM.

Nielsen, J. (1993, 11). Iterative user-interface design. *IEEE Computer, 26*(11), pp. 32 - 41.

Reddit inc. (2017). *Reddit - Procedural Generation*. Retrieved from Reddit - Procedural Generation: https://www.reddit.com/r/proceduralgeneration/

Rogers, S. (2010). *Level Up! The Guide to Great Video Game Design.* Wiley.

Shaker, N., Togelius, J., & Nelson, M. J. (2016). *Procedural Content Generation in Games: A Textbook and an Overview of Current Research .* Springer.

SQUARE ENIX CO., LTD. (2017). *Final Fantasy XIV*. Retrieved from Final Fantasy XIV: http://www.finalfantasyxiv.com/

Trion Worlds, Inc. (2017). *RIFT*. Retrieved from RIFT: http://www.trionworlds.com/rift/en/

Valve. (2017). *Steam*. Retrieved from Steam: http://store.steampowered.com/

Yee, N. (2006, June). The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments. *Presence: Teleoperators and Virtual Environments, 15*(3), pp. 309-329.

9 Appendix

9.1 Final Design requirements

1. Generated quest will be functional game elements.
2. Focus on content, not narrative.
3. Use all the different motivations with underlying strategies and sequences of actions as quest structure, as theorised by Doran and Parberry (2011).
4. Generated quest contain different elements from game world.
5. Generated quests should be located in a wide array of locations.
6. Generator should use a search-based algorithm approach.
7. Same quest can repeat if the conditions repeat.
8. Generator will not adapt quest to player input.
9. Generator work online (during play) to find new quest once one have been completed.
10. Replaced quests cannot be a repetition of the one it replaced.
11. Game should have a quest tracker or quest log, which show all available quests and their progress.
12. Ensure minimum of two position for each quest
13. No restriction of placing quests in areas where the object is already found.
14. Use all six enemy types as part of the objectives.

9.2 Survey Questions and Scales

- Gender:
    - Male/female
- Age:
- Do you play video or computer games?
    - Yes/no
- Could all the quests be completed?
- The quests were too similar.
- The quests concerned different tasks.
- Different NPCs were involved in the quests.
- Different creatures/monsters were involved in the quests.

- Different items/objects were involved in the quests.
- The quest fit in their environment.
- The quests were spread out.
- The distance between quests was too big.
- New quests became available too slowly.
- There were not enough quests available at the same time.
- Generally the quests felt:

	Very Much	Somewhat	Neither	Somewhat	Very Much	
Different		X				Alike
Understandable					X	Confusing
Varied				X		Repetitive

9.3 Survey responses

The survey responses can be seen in the excel sheet in the Appendix ZIP-fil names 'Survey Responses'.

9.4 List of derived abstract strategies

- Deliver – 'Collect' and 'Give'
- Hunt – 'Kill'
- Bounty – 'Kill' and 'Report'
- Use – ' Use'
- Obtain – 'Get' and 'Use'
- UseItem – 'Get', 'Use' and 'Give'

9.5 Directions to scripts in ZIP

Each type of script in the 'Scripts' folder found in the appendix ZIP-fil.

Under the QuestSystem sub folder the following is true:

- Events contain: Item, Target, Enemy
- Motivations contain: Each motivation class
- Quests contain: each quest class fragment such as 'KillQuest' for 'Kill'

- Subfolder Definitions contain: definitions for each quest fragment

- Strategies contain: strategy classes for each strategy (see 9.4)
- Tracker contains: quest log class

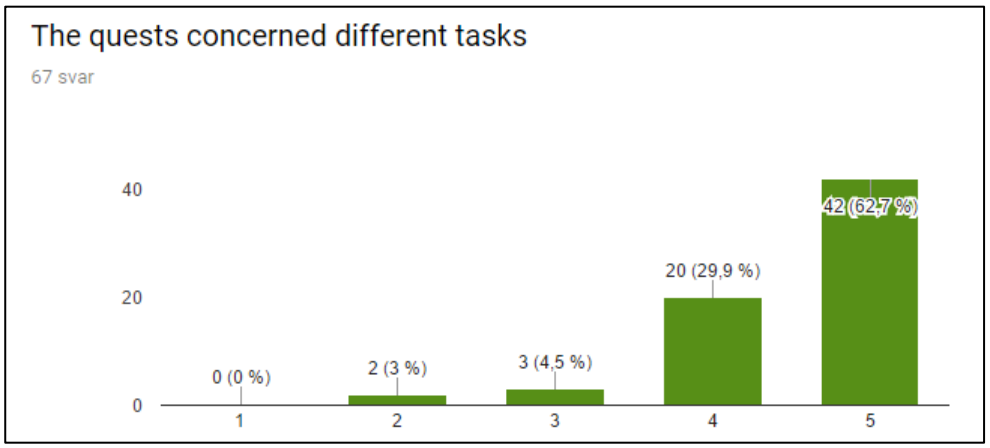
9.6 Builds of prototype game

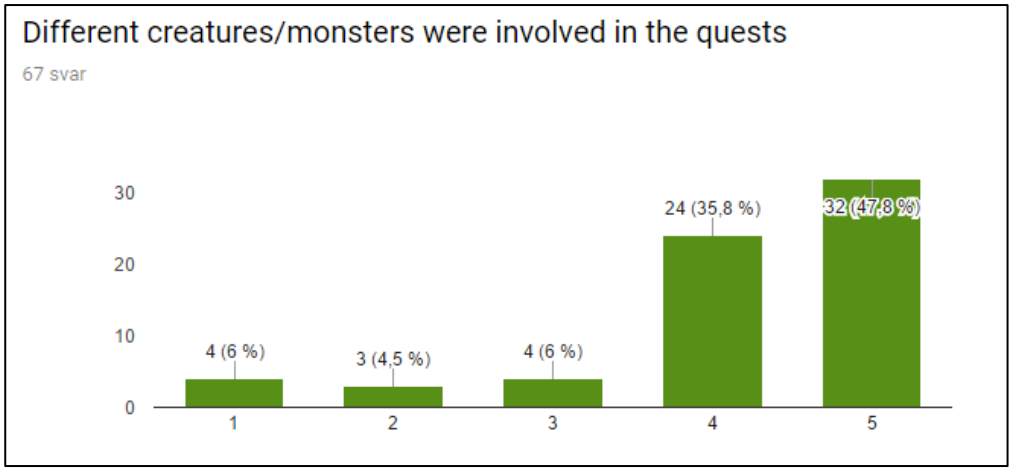
In the appendix ZIP-file is a folder called 'Builds' with a sub-folder for each 'Windows' and 'Mac' containing working builds of the prototype.

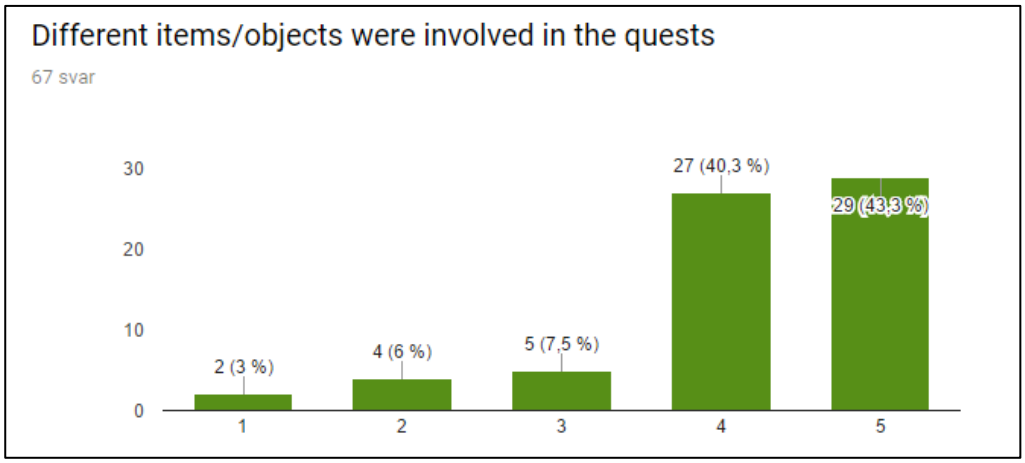
9.7 Figures of survey evaluation

Following appendix section contain additional figures done in relation to the survey.

9.7.1 Tasks, Enemies (creatures/monsters) and items/object






Different items/objects were involved in the quests
67 svar
30
20
10
0
2 (3 %)
4 (6 %)
5 (7,5 %)
27 (40,3 %)
29 (43,3 %)
1
2
3
4
5