Sandbox Warrior

Procedurally generated Roguelike RPG

4/19/2015

Emmet Boylan
20044540
Final Year Project

Table of Contents

Introduction 5

Glossary 6

1 Game Overview 9

1.1 Game Concept 9

1.2 Feature Set 9

1.3 Genre 9

1.3.1 Principles of a Roguelike game 9

1.3.2 Roguelikes in the Greater Gaming Context 14

1.3.3 Roguelike Summary 15

1.4 Target Audience 16

1.4.1 Player Motivations 16

1.4.2 Engaging Player Types 19

1.4.3 Overall Assessment Of Target Audience 20

1.5 Game Flow 21

1.6 Look and Feel 21

1.7 Project Scope 21

2 Game play and Mechanics 22

2.1 Game play 22

2.2 Mechanics 23

2.2.1 Movement 24

2.2.2 Objects 26

2.2.3 Actions 27

2.2.4 Combat 32

2.2.5 Economy 34

2.3 Screen Flow 35

2.3.1 Starting Screen 35

2.3.2 Play Screen 36

3 Story, Setting and Character 37

3.1 Story and Narrative 37

3.2 Game World 37

3.3 Characters 37

4 Levels 39

4.1 Cavern Generation using Cellular Automata Theory 39

4.2 Cavern Generation Implementation 40
4.3 Cellular Automata Weaknesses 41
5 Interface 42
5.1 GUI Layout 42
5.2 Keyboard Input 44
6 Artificial Intelligence 47
6.1 AI 47
6.1.1 WanderMonsterAI 47
6.1.2 HuntingMonsterAI 47
6.2 Support AI 47
6.3 Path finding 49
6.3.1 A * Summary 49
6.3.2 A * Pseudocode 49
7 Technical 51
7.1 Target Hardware 51
7.2 Development hardware and software 51
7.3 Slick 2D 51
8 Game Art 52
8.1 Screen Art 52
8.2 Terrain, Item and GUI Graphics 52
9 Secondary Software 53
9.1 FLICster 53
9.2 Photoshop CS2 53
10 Management 54
10.1 Time Management 54
10.2 Version Control 54
11 Implementation Iterations 55
11.1 Determine Objectives 55
11.2 Prepare Environment 56
11.3 Input/Output 56
11.4 Map 57
11.5 Data Format and Persistence 57
11.6 Simple Game Structure 57
11.7 Entity Interaction 58

11.8 Game Data 58
11.9 Items and Inventory 58
11.10 Special Effects 59
11.11 Simple Game Implementation 59
11.12 Game Levels 60
11.13 Experience and Player Levels 60
11.14 Non Player Characters 60
11.15 Unique Features 61
12 Conclusions 62
12.1 Fundamental Design Seems Sound 62
12.2 Slick2D Animation Issues 62
12.3 Binary Space Partitioning Results Unsatisfactory 63
12.4 Effect System Has Great Potential 63
12.5 Tools For Player Modification Will Be Needed 63
12.6 Future Plans 64
12.7 Final Thoughts 64
13 REFERENCES 65
14 Additional resources 66
14.1 Online tutorials 66
14.2 Blog 66

Introduction

This document describes the design and implementation of the "Sandbox Warrior" video game. It is an example of a sub-genre of computer role-playing games called Roguelikes, named for the original game "Rogue". This sub-genre features simple clear objectives and challenging game play with excellent replay value due to the use of randomised, procedurally generated content rather than manually designed elements. Although developed in this way due to a lack of computer memory and processing power that no longer exists, procedural generation is a powerful gaming tool used in projects such as Minecraft, Elite: Dangerous (current version of the original Elite an early implemented of procedural generation) and the still in development space exploration simulation No Mans Sky.

This document references a model of Player Motivations and compares them to the features of a Roguelike game. This will allow us to identify a target audience. This model acts as a guide in the decision making and design process. Aspects and design choices that complement the target audience are favoured over those that do not.

This document takes the format of a Game Design Document, a document used in the video game industry to provide clear guidelines for the development of a project. Advantages of having a Game Design Document include: (Gamasutra.com, 2015)

Elimination of hype

By defining substantial elements of the project specifically, scales back unrealistic aims and expectations.

Clarity and certainty

They create uniformity and a common frame of reference for the development team. Design has clear objectives to follow.

Ease of drafting schedules and test plans

By itemising the needed components and assets clearly it becomes easier to accurately form schedules and plans.

It was agreed that this form of document is best suited to describe the design and implementation of the project.

Glossary

***Actor - (*subtype of *Player Motivation*)**

A player whose actions are predominantly motivated by immersion in the role. A player who makes choices based on "what my character would do" rather than tactical considerations.

***Casual - (*subtype of *Player Motivation*)**

A player looking for a simple and enjoyable gaming experience that can be picked up or left off with a minimum of complication.

Chimeric

A creature made up of the components of other creatures. From the ancient Greek Chimera a fire breathing combination of a lion, goat and serpent. This games *Manticore* is *Chimeric.*

***Explorer - (*subtype of *Player Motivation*)**

A player motivated by the desire to find out more about the game world, discovering new areas, new opponents, NPC's or levels.

Game Design Document

A game design document is a highly descriptive living design document of the design for a video game. Used in the Video Game Industry to organise the implementation of a game. Although widely used there is no set standard form of the document.

The game design document used in the report follows the template designed by Mark Baldwin of Baldwin Consulting and available from :

https://www.kth.se/social/files/545bbfadf276545bbefa9cf6/BaldwinGameDesignDocumentTemplate.doc

Goblin

A goblin is a legendary evil or mischievous grotesque dwarf-like daemon or monster that appeared in European stories and accounts during the Middle Ages. They are common staple of fantasy RPGs where they offer a relatively low level threat. In this game they act as minions to the *Manticore*.

***Instigator - (*subtype of *Player Motivation*)**

A player who takes actions based on curiosity about the repercussions. A player who wants to "see what happens next"

Manticore

A monster from ancient Persian myth. A *chimeric* combination of lion and human with bat like wings. Similar to the Egyptian Sphinx. An intelligent, evil and man-eating creature. It is the players mission to find and slay this monster.

Non Player Character (NPC)

A fictional Game Entity controlled by the computer. Enemies, allies and neutrals that share the world with the PC.

Player Character (PC)

The fictional Game Entity controlled by the player.

***Power Gamer - (*subtype of *Player Motivation*)**

A player interested in maximising their advantages and power level.

Roguelike

A subgenre of RPGs featuring common aspects such as simple graphics, random procedural generation of game content and no restart upon player character death.

Role Playing Game (RPG)

A game in which players assume the roles of characters in a fictional setting. Come in Tabletop, Computer Game and Live Action formats. In this game the player assumes the role of a brave inhabitant of a fantasy medieval world, of indeterminate gender and background, who must enter a system of caverns to slay an evil monster.

***Slayer- (*subtype of *Player Motivation*)**

A player who enjoys combat above other elements.

***Storyteller - (*subtype of *Player Motivation*)**

A player interested in the plot, story and their role within in.

***Thinker - (*subtype of *Player Motivation*)**

A player interested in mental stimulation. Interested in puzzles and tactical challenges.

1 Game Overview

1.1 Game Concept

A procedurally generated medieval fantasy RPG. Challenge based on exploring unknown world, managing limited resources and becoming powerful enough to defeat a powerful enemy.

A Roguelike game with animated graphics.

1.2 Feature Set

- Random generation of game world for challenge and replay potential.
- NPC ecosystem with variety of AI types.
- Crafting system.
- Simple and high risk combat system.

1.3 Genre

Sandbox Warrior is a Roguelike game. Roguelikes constitute a distinct subgenre of role playing games (RPGs) that has been popular since the 1980s.

1.3.1 Principles of a Roguelike game

In 2008 the International Roguelike Development Conference agreed on a definition of a Roguelike game. This "Berlin Interpretation" will be used as a reference point to discuss my projects. (Roguebasin.com, 2015)

1.3.1.1 High Value Factors

These are the factors considered core to a Roguelike game.

Random environment generation

Roguelikes produce random dungeon layouts and populate with randomly placed monsters and items.

This best suits players with an explorer motivation.

Permadeath

Traditional roguelikes feature permanent death, without resurrection. This is important to give the risk of exploration and combat meaning. Save files are deleted upon load to prevent the player avoiding this restriction.

This provides combat challenge for the Slayer and Power Gamer motivations. The fact actions cannot be repeated or retried adds risk, consequence and realism to choices, this can appeal to Actors. The challenge of determining the best course of action with limited data appeals to Thinkers.

Turn Based

The game will feature the traditional turn based feature of Roguelikes.

This allows for time to think tactically, rather than reaction based combat. This appeals to Thinkers, and Power Gamers. Slayers might enjoy the tactical side, or may be put off. Casuals may be put off by the slower action.

Grid Based

The game will feature the traditional grid based game play of Roguelikes.

Actors and Storyteller may be put off by the unrealistic grid constraint on movement.

Non Modal

The game will feature the traditional non modal game play of Roguelikes. Combat, exploration and other actions all take place in same mode. With upgraded graphics it is possible to provide more information and feedback to the player than in a more traditional Roguelike. Without a need for separate 'look' or 'examine' commands and screens this project meets this factor more than the original.

This should have no real influence of Player Motivation.

Complexity

Roguelikes allow enough complexity to allow several solutions to common goals. This is obtained by providing enough item/monster and item/item interactions and is strongly connected to having just one mode. There are no plotted scenarios, balanced encounters or expected outcomes. The player must engineer their own solution.

Thinkers and Power Gamers appreciate game complexity. Instigators can enjoy finding game play combinations. Casuals will be put off by too much complexity.

Resource Management

Resources are limited and have purpose. Without enough food the player will starve. Without equipment upgrades the player is unlikely to defeat the Manticore. Without healing potions the player is unlikely to survive to reach the final confrontation. However to obtain these resources the player must risk combat against other dungeon denizens. Balancing risk versus reward with limited knowledge of the game world is a cornerstone of a Roguelikes game play.

Thinkers and Explorers enjoy resource management components.

Hack'n'slash

Roguelikes feature a simple player-versus-world, monsters are fought or avoided rather than bargained with.

Slayers, Casuals and Power Gamers appreciate this clear game play style.

Exploration and Discovery

The random and changing environment requires careful rediscovery each time it is played. This adds drama, tension and re-playability.

Explorers and Thinkers are the main target audience for Roguelikes. The focus on exploration, challenge and discovery is the major factor why.

1.3.1.2 Low Value Factors

These factors are common in Roguelikes, but are not considered as definitive of the style.

Single Player Character

Traditionally the player controls a single character whose death ends the game. While challenging game play wise, can be bad for a story narrative as the main character may die arbitrarily and without seeming purpose.

Bad for Storytellers. Good for Thinkers and Explorers.

Monsters Are Similar To Players

Traditionally monsters have the same rules as players, having inventories of usable equipment, spells etc.

Slayers, Casuals and Power Gamers appreciate this clear game play style.

Tactical Challenge

Roguelikes focus on tactical challenge due to changing environment rather than strategic challenge or puzzle solving.

Slayers, Casuals and Power Gamers appreciate this clear game play style

ASCII Display

Traditional Roguelikes used ASCII characters for graphics. e.g. "@" as the player, "#" as a wall and so one.

This game does not follow this factor. While seemingly a large change from the classic rogue archetype, graphical enhancements are common among roguelikes. As with procedural generation this aspect of early roguelikes was imposed by computer limitations. Unlike procedural generation there are no real advantages for continuing the practice. Graphical enhancement provide a great deal of information to the player (removing requirements for 'look' and 'help' screens).

Graphical enhancements will appeal to a wider audience, however this may well come at the cost of alienating 'purists' who prefer the original approach.

A side effect of this factor is that the scale of Game Entities is not proportional. Since the graphics for each entity consisted of a single ASCII character and the games grid based a hobbit or rat occupied the same area as a dragon or Balrog. This project follows this aspect of the genre to maintain the same game play feel, despite relatively updated graphics. Hence, large creatures like the end boss Manticore are scaled down to fit into the same tile as other Game Entities.

Dungeons

Levels consist of dungeons; a collection of rooms, connected by passages and filled with monsters and loot. Dungeons can be large natural cave structures, artificial subterranean warrens of mythic underworlds.

Good for Thinkers, Explorers, Slayers and Power Gamers.

Numbers

Player stats are deliberately shown.

1.3.2 Roguelikes in the Greater Gaming Context

There are 2 main development paradigms in video gaming. Large AAA type games developed by large companies involving 100 of programmers and many subcontractor companies, and Indie type games developed by individuals and small teams.

1.3.2.1 AAA Game Approach

A modern triple-A video game has a budget measured in millions. In 2008 Ubisoft spent between 12 and 18 Million dollars on its games for PC/XBOX/PS3 and expected the budget for next generation games to rise to about 60 Million. (Boyer, 2015) 6 video games have passed the 100 million mark (Star Wars: The Old Republic cost 200 Million) and Call of Duty: Modern Warfare 2 spent 200 Million on Marketing(compared to "only" 50 Million on actual development). (Wikipedia, 2015) Triple-A games are a massive endeavour with large staffs and huge stakes.

Crytek CEO Cevat Yerli has claimed that graphics are 60% of a game, but that statement needs to be qualified. (GameSpot, 2015) Graphics might be 60% of what it takes to be a commercial success. With massive budgets and careers on the line, sales are vital. Advertising hype and good reviews are worth far more in economic terms than good game play. Metacritic scores are used to determine if bonuses are paid or not, and these can make or break companies. (Schreier, 2015) For all that in 2011 Ubisoft made a 3% profit margin, and EA 7%. (Staff, 2012)

While Triple-A games are the glamorous end of the industry they are impossible to break into as an individual or small team and feature such a massive cost base and high risk, low yield dynamic that may not be worth pursuing.

1.3.2.2 Indie Game Approach

Freed from graphics engine, art, voice acting and other requirements drastically reduces cost and allows development for focus on game play elements instead. Such games can still be massively profitable even without impressive graphics. Minecraft is mainly the work of one programmer yet has sold nearly 19 million copies and the development company Mojang was bought by Microsoft for over 2.5 Billion dollars. Game of War: fire edition, a browser game has sales of over 1 million dollars, per day, it's developed Machine Zone are valued at 3 Billion dollars after an investment of 13 Million dollars.

The Video Game industry is worth about 24 Billion dollars a year (compared to the movie industries 10 Billion) and while casual games make up only 10% the significantly lower costs and greater growth rate make it a very attractive option.

1.3.3 Roguelike Summary

Roguelike games were initially developed in an era of restricted processing power and memory. Procedural generation allowed games like Elite to feature huge explorable game universes with minimal resource cost. The same constraints forced a focusing of player aim on clear targets. While the processing and memory constraints of the past no longer apply, the same processes can be used to meet the challenges of a limited budget, both financially and temporally.

This makes them ideal models for development by individuals and small teams.

1.4 Target Audience

To determine the target audience for a game, the author will begin by defining a typology of gaming audience groups. This will give us a definitive metric we can use to quantify and better define what might otherwise be a subjective and vague area.

The essence of player engagement is giving the player what they want. While individual motivations can vary, even for the same player at different times, there are recurring patterns that can be examined. The following list of player types is taken from the Tabletop RPG Dungeons and Dragons. While the medium is different, motivations for playing RPGs are the same for computer and tabletop RPGs. Game design and theory is better documented for Tabletop RPGs as essentially in each group one person must act as a game designer.

1.4.1 Player Motivations

Initial advice for designing and running games of Dungeons and Dragons consisted of examples of good and bad players and how to use in game events to correct the behaviour of bad players. This type of authoritarian and passive-aggressive approach is no longer favoured. The modern approach is to examine what each type of player looks for in a game, and how to meet these desires. (Laws, Player Motivations, 2009) (Laws, The Players, 2008)

This gives us a list of elements to include that will be of interest to the group or groups in mind. It may not be possible or even desirable to meet all player types expectations in one game. A clear understanding of which elements map to each motivation will allow the design to focus on creating a good game for one target audience rather than a poor or mediocre game for all.

Player Motivation Types:

1. Actor
2. Explorer
3. Instigator
4. Power Gamer
5. Slayer
6. Storyteller
7. Thinker
8. Casual

1.4.1.1 Actor

A player motivated by immersion and playing a role. Game choices are motivated by what is appropriate for the role chosen rather than simply choosing the most powerful option. For example a player might forgo wearing more powerful armour because it "doesn't look Viking enough". Actors will take actions that support the illusion that the character is a real person, for example eating at mealtimes rather than just for healing, or sleeping at night-time. Values narrative game elements over mechanical ones.

An actor is likely to play according to the motivations they have set for themselves rather than following plotline/quest line in strict order. Actor appreciate opportunities to portray their character traits, interests or backgrounds.

1.4.1.2 Explorer

The explorer is motivated by discovery. They will explore the world, search out hidden zones, new creature types, terrains, cultures and levels. They respond well to atmosphere, a clear sense of style and details that vary across the world. They are curious and happy to immersive themselves in the game-world, so long as the game-world rewards their curiosity.

1.4.1.3 Instigator

Instigators are impulsive players. They will take actions, often detrimental to their character, just to see what happens next, triggering traps, insulting high level NPCs. They enjoy risky behaviour and are less likely to plan things carefully or do things slowly. Rather than engaging with the plot on the games terms they are likely to act randomly or even subvert the plot. However, as long as the game responses are enjoyable they will happily continue to play, taking responsibility for their own enjoyment and actively engage, even if it is not in the manner the designer intended.

1.4.1.4 Power Gamer

Power Gamers are motivated by the desire to make their character as powerful as possible. They will carefully optimize their characters stats, equipment and powers. Their choices are determined more by mechanical advantage than by narrative theme. They will seek to take advantage of any bugs in the game system or unexpected combinations. They might create or consult spreadsheets of variables or take boring, repetitive actions to eke out any advantages.

1.4.1.5 Slayer

The Slayer is motivated by the enjoyment of combat encounters.

1.4.1.6 Storyteller

The Storyteller is more interested in the overall narrative and plot of a game, rather than individual motivations and personality. They are interested in uncovering the plot and seeing where it leads. They will take options that they think will advance the story, rather than those that might better fit the characters personality or current context. They want to see where the tale goes, rather than force their own.

1.4.1.7 Thinker

A Thinker likes to make careful choices about their actions. Forward planning, careful analysis and strategy are used to minimize risks and resource use and maximize advantages. Puzzles and challenges are the major sources of enjoyment.

1.4.1.8 Casual

Dungeons and Dragons refers to this player type as a "Watcher"; a casual player more interested in spending time with friends who are playing, rather than the game itself. The computer gaming equivalent is the casual gamer. One who is not so invested in the game, or not willing to learn complex back stories or controls. A casual gamer plays for short periods of time so looks for a game that is simple to get into and does not require a great deal of effort.

1.4.2 Engaging Player Types

The different strategies that are required to engage the different player types are briefly outlined.

1.4.2.1 Actor

- Provide role playing encounters
- Focus on character development and give personality traits an effect
- Interesting/realistic NPCs

1.4.2.2 Explorer

- Reward curiosity and exploration
- Include interesting locations/environments
- Include "side-quests", unexplored areas and hidden treasures

1.4.2.3 Instigator

- Include game elements (items, traps, spells, factions...) that the player can experiment and interact with; factions that can be played against each other, items and spells with powerful combinations etc.

1.4.2.4 Power Gamer

- Adding rare items or spells that increase character power to be sought out by the player
- Allowing player to optimize choices as they level and become more powerful
- Giving monsters strength that can be overcome, or weaknesses that can be exploited by the selection of the correct powers, equipment or options
- Adding titles or status to the game that the player can aim to achieve

1.4.2.5 Slayer

- Provide an interesting combat system
- Avoid long sections without combat encounters

1.4.2.6 Storyteller

- Include an engaging plot and definite story arc
- Integrate PC background into the story

1.4.2.7 Thinker

- Include puzzles and resource management
- Allow tactics and strategy to benefit player in combat
- Include turn based combat based on tactics rather than "twitch" based combat relying on reactions

1.4.2.8 Casual

- Simple instructions and combat
- Clear UI and controls
- Game play that can be picked up or dropped without complication

1.4.3 Overall Assessment Of Target Audience

Roguelike games focus on Thinker, Slayer, Power Gamer, Explorer and to a lesser extent Instigators. They are generally a poor match for Storytellers and Actors. Casual gamers can be welcomed buy the clear and simple game play style, but can be put off by the difficulty and presentation. This will define our target audience.

1.5 Game Flow

The game features a relatively simple flow. The player begins with inadequate resources to challenge the main threat. Additional resources must be gathered by exploring the dungeon. Rewards and challenges both increase as the player descends levels in the dungeon. However, exploration exposes the player to greater danger and combat with more monsters. The game play element is based on the player attempting to juggle these resources and determining what risks to take.

The game only reveals a limited area of the world to the player at any time. The player will have to take risks based on limited information. Furthermore, as the player does not have multiple lives any knowledge gained on one play through is useless in the next. This prevents the player exploring all options and determining a course of action based on knowledge gained over multiple attempts.

1.6 Look and Feel

The game is a 2 dimensional, grid based RPG. It uses open source 32 x 32 pixel tiles intended for the development of Roguelike games such as Crawl and Nethack. These graphics are supplemented by sprite based unit animations. The sprite sheets are converted by the Author from open source fan made unit animations for the Civ3 strategy game.

A bar on the right hand of the screen details the players stats, equipped items, inventory and text based feedback on game events. Textual heading can be turned on to clarify symbols used. The main section of the screen displays the portion of the game world the player can see. The size of this area is affected by the characters light source.

1.7 Project Scope

The game takes place in a randomly generated 3 dimensional maze of caverns. A new world is generated for each play through in keeping with the games genre conventions and in the interest of replay value. Navigating a layer of the world may require travelling via other layers of the dungeon.

The entities that inhabit the world, as well as items and equipment are also generated randomly for each instance of the game.

The scope is therefore fluid. It is therefore impossible to scope out precisely. Game challenge and balance is dependent on the algorithms used to generate the world.

2 Game play and Mechanics

2.1 Game play

Game play consists of moving around the game world searching for resources needed to confront the games Boss Monster. A limited area of the world is visible, the radius of this visible area dependent on if the player carries a torch. If a shield is found visibility may be traded for increased defence.

It would take exceptional luck for a beginning character to defeat the Manticore, realistically the player will need upgraded equipment and potions. These items can be found around the game world, but searching for them is likely to lead to confrontation with Goblins. As each attack causes a minimum of 1 damage, any conflict costs the player a valuable resource, i.e. hit points. Running from combat is possible, but may cause the player to be confronted by monsters in front and behind.

Hunger acts as a pacing mechanism. If the player is too cautious or slow they will lose hit points to starvation. Food can be gained by attacking Lizards for meat, but these creature will defend themselves, again costing the player resources.

Limited information about the location of monsters and the layout of the caverns mean the player will have to make choices without all the facts. They will have to trust their instinct and experiment to develop the best strategy.

If the Player Character is killed, all information and resources gained so far are lost. The is no player re-spawn. This generates a sense of real risk and loss often absent from games where it is trivial to re-spawn and make a decision based on information you have about the alternative. Roguelikes are generally hard, frustrating even. The Player Character is not central to a plot or narrative and so has no "plot armour". Because of this difficulty beating one has an added satisfaction of finally overcoming a challenge. Plus the added bonus of a replay being unique and challenging all over again.

2.2 Mechanics

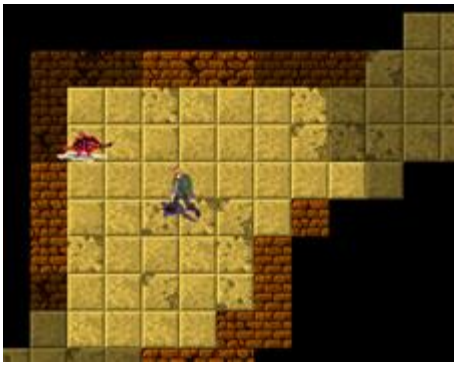
The game world is made up of a 3 dimensional grid of tiles. This grid is used to display graphics in the correct location, determine Game Entity position and movement. It is used to simple and quickly adjudicate collisions, line of sight, path finding and visible radius.

Each Game Entity consists of the data need to display it correctly on screen, AI algorithms to determine options and actions as well as combat variables such as Attack, Defence and Hit Points

2.2.1 Movement

Movement is in 8 directions and controller via the keyboard number pad, with 8 representing up, 6 representing right, 9 representing up and right and so on.

Each game entity has movement animations for each direction. Game Entity movement is grid based rather than pixel based. A Game Entity occupies a distinct grid cell, it is not possible to "straddle" 2 or more cells. Game Entity Animations give the illusion of movement by transforming the position of the graphic along a line joining the 2 cells while the game logic simply records the cell location of the Game Entity.



Player character moving North West, in the direction of the lizard, by pressing7 on the number pad. Although shown between tiles the system considers the PC to be in the lower right square until the end of movement.

*M*ovement is blocked by Walls and the boundary of the game world.

Movement to other levels of the Game World is possible via stairs up or down. This done by pressing 5 on the number pad while on a stair tile.

Player Character approaching stairs down to the next cavern level on the left, and on the right shown at the top of the stairs on the next level.

Movement is conducted on a turn basis. System waits for player input. Once given a move or attack command the system animated the players move. Once this animation is over the system determines move for the computer controlled game entities in one batch. Once this determination is made, all visible computer controlled Game Entities moves are animated at the same time, rather than making the player wait as they are animated sequentially.

Movement into an unfriendly Entity causes a combat attack to occur rather than movement.

2.2.2 Objects

The game contains a selection of items useable by the Player Character and Non Player Characters. Items can modify Entity statistics or impose status effects. Items are stored in an inventory and the players inventory shown on screen. A selection box show information about the item shown. This box is also used to select items to be equipped, dropped or used as well as items to be used in crafting/cooking. The Items the player has equipped are shown in the top right corner beside icons for each hand and the body (the locations where items are equipped to).

The GUI shows that the player has a dagger equipped in their right hand, a shield in the left and is wearing leather armour on the body. These icons are displayed in the top right corner. The large central area with the chest icon is the players inventory. Here items can be selected. A dagger is the current selected item. The lower box is an information panel. It shows the name of the item selected, the attributes of the item (+1 Attack) and the commands available for the item.

Game object include;

Weapons, that increase the Attack score when equipped

Armour, that increases defence when equipped

Food, that can be picked up from creatures and eaten raw or cooked

Potions that have a variety of effect and whose effects differ from game to game

Items used in the left hand such as a torch that increases visible radius, or shields that increase defence

Crafting/Cooking can create new objects out of 2 ingredient objects

2.2.3 Actions

Apart from movement and combat there are a variety of other actions the player can take on their turn. These actions are discussed below

Wait

By pressing 5 on the umber pad the player simply stays in place. The NPC Game Entities can then take their next turn. This action can be useful if you want to wait for a wandering monster to move out of the way, while hiding on a goblin, or simply to wait until you can regenerate some hit points

Pick up Item

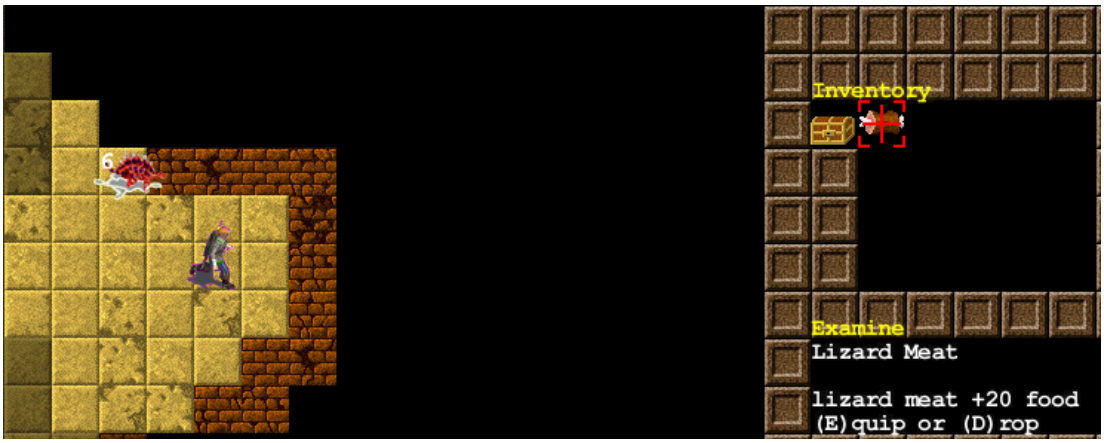
By pressing the letter 'p' the Player Character picks up the first item in the same tile. This item is then added to the players inventory, if there is a free slot. Items must be picked up to be used or equipped.

Player Character moves towards tile containing Items. When in the same Square pressing 'p' removes the Items from the game world and places it in the players inventory.

Player Inventory showing Items picked up and message box informing player of what these Items were.

Drop an Item

Pressing 'd' will remove the selected item from the players inventory and place it in the game world on the same tile.



Player Character with some Lizard meat in inventory.

Player Character has dropped Lizard Meat and moved away.

Use an Item

This action is fired by pressing 'u'. If the Item has an effect, pressing 'u' triggers it. Food items are destroyed and an amount added to the players food score to stave off starvation. Potions are destroyed and a permanent or temporary effect is added to the Player Character.

Player Character before using Blue Potion

Player Character aftre taking potion. In this game the blue potion heals us by 2 points.

Craft/Cook

The 'c' key will bring up a second (yellow) select box. This box is controlled by the arrow keys just like the normal item select. By selecting a second Item and pressing 'c' again the system will attempt to combine both Items into something useful. For example combining a torch with raw meat results in cooked meat, that increases food by more than raw food. Incorrect combinations will destroy both ingredients and create a ruined Item that merely takes up inventory space.

Selecting an empty inventory slot cancels the craft selection box without consuming Items.

A failed attempt at crafting

A successful attempt at cooking

2.2.4 Combat

Moving into a tile already occupied will attack the Entity already in that tile. Combat is simple, the attacker deals an amount of damage to the defender on each attack. It is not possible to dodge an attack and while armour and physical toughness reduces damage it does not eliminate it. This means it is impossible to be invulnerable, combat always costs, and will not go on indefinitely. Even a parried attack or one that hits armour would still deal blunt trauma to the enemy and tire them out. Even the greatest fighter can be brought down by numbers and exhaustion.

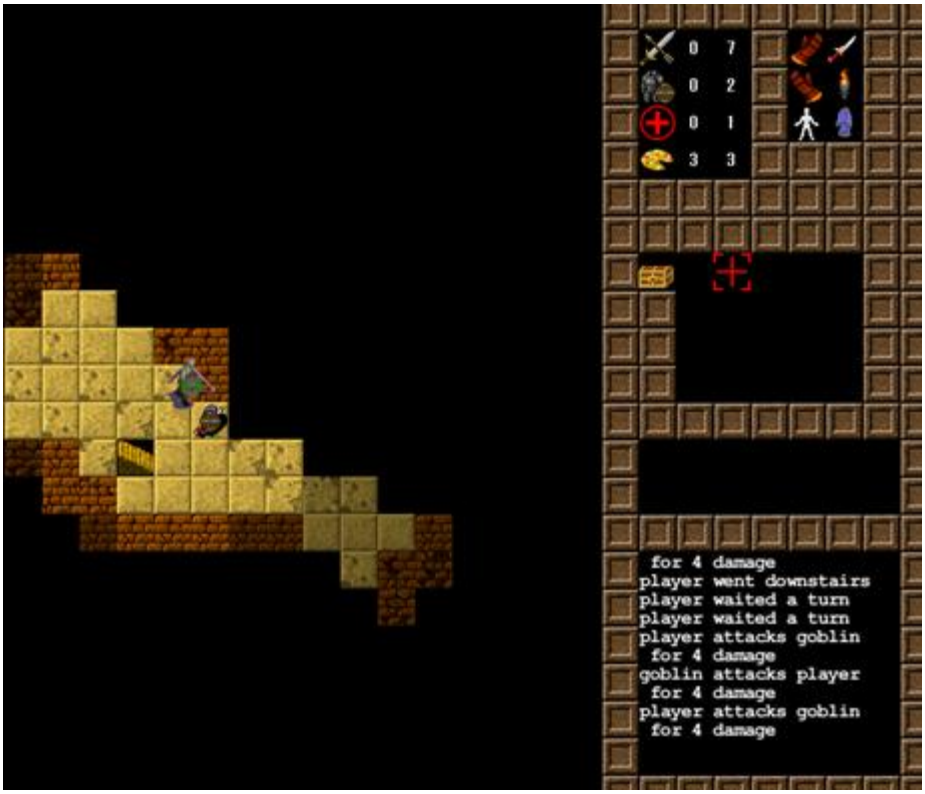
That combat is risky and costly is vital to the tone of a Roguelike. Picking your fights carefully, knowing when to cut and run and learning how to use the terrain to your advantage give a different feel to combat than typical RPGs.

The combat mechanic is simply a random number between ;

(0 and (attacker Attack Value - defenders Defence Value)) + 1

Armour will reduce the highest, average and mean values of an attack, but cannot eliminate it entirely.

A fight versus a vicious Goblin. As the minimum damage 1 and the Player Character only has 1 hit point left, the next attack is vital.



Victory! The Goblin has dropped the equipment it carried upon death and they are now the Player Characters to loot.

2.2.5 Economy

Although the game currently lacks NPC shops and monetary loot as mentioned earlier the in game economy is very important. The Player Character has limited resources and must risk the resources they have in the hopes of finding and winning more. These resources include

Hit Points

The players most vital resource. If they run out, all is lost. They will regenerate slowly but too much exploring or waiting increases the chance of being found by a wandering Goblin. There is a limit to how long the character can wait due to hunger. Lizards can be hunted for food, but doing so risks more hit points in combat. Potions can replenish hit points, but as their effects are randomised a given potion may harm instead of healing. It is best to "test" potions early in the game when the player has hit points to spare and then search for more potions of the same colour.

Food

Food is another important resource. Once the PC runs out of food they will begin loosing hit points. This dwindling resource acts as a pacing mechanism encouraging the PC not to spend too long searching for resources.

Armour and Weapons

These Items directly and permanently increase the players attack and defence values closing the gap between the PC and the Manticore and increasing odds of victory. Increasing these values protects the PCs most vital resource, hit points; Armour by reducing the damage of each attack and Weapons by reducing the amount of attacks the PC is likely to receive (by killing the enemy in fewer turns).

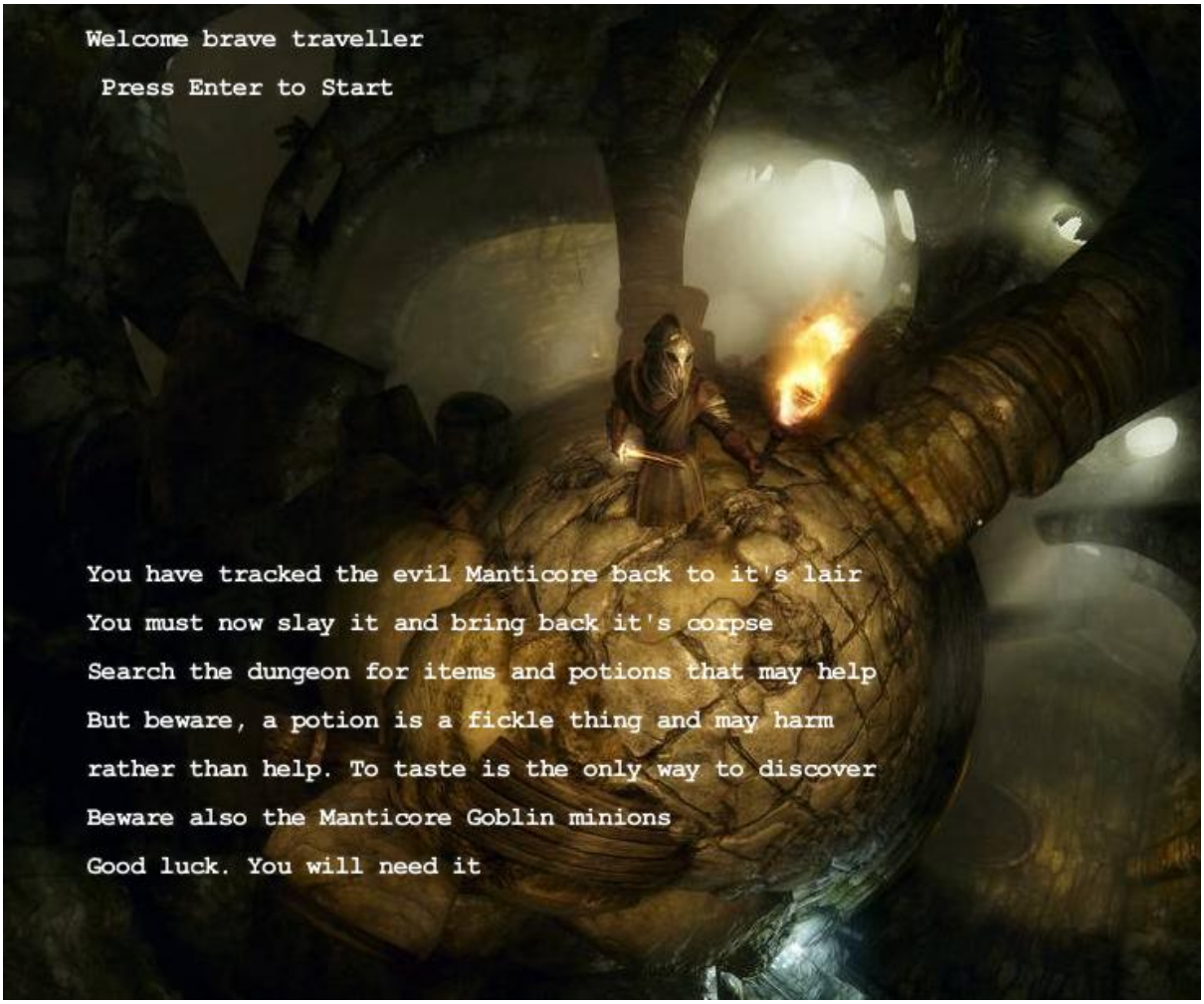
Potions

These one use Items can temporarily increase or decrease players Attack Value, Defence Value and Hit Points. The boost provided can help defeat the Manticore or save a desperate player earlier in the game. They are limited in number and scattered around the game world.

2.3 Screen Flow

2.3.1 Starting Screen

This screen acts as an introduction to the game, giving some background context and tips.



From here play proceeds top the Play Screen

2.3.2 Play Screen

The Main Game screen, with the world displayed on the left and centre and the players User Interface on the right.

From here the player has 3 potential exits.

1. They can leave the caverns via the stair up from the highest level. This will result in the player losing the game.
2. The PC might be slain in combat or die of starvation. This will result in the player losing the game.
3. The PC can win the game by slaying the Manticore and returning to the surface with its remains.

3 Story, Setting and Character

3.1 Story and Narrative

The story and narrative of Sandbox Warrior are rather simple. There is little more than a Protagonist with a clear goal, an Antagonist and some minion to provide a threat, a non defined home to defend and a dangerous setting unique to each instance of play. Ultimately video game stories and narrative exist to enable and inform the game play, to provide context and motivation to what are fundamentally a limited set of arbitrary decision points. A simple and generic story serves this purpose as well as a more complex one and is far easier to come up with. However, efficiency is not the only argument for such an approach.

Minimalist, archetypal characterisations aid immersion by allowing the audience to project themselves onto the role of the protagonist. This is the reason so many first person protagonists in games do not speak, why Keanu Reeves makes millions portraying fairly wooden and unemotional heroes and Bella Swan receives almost no character or personality traits in Twilight. It is far easier to see a character as your own and to make choices without the baggage of character details that would suggest another course of action. Encouragingly this approach is also significantly cheaper in terms of game assets.

3.2 Game World

The perceivable portion of the game world consists of a 3 dimension network of caverns inhabited by dangerous creatures. The world persists as long as the *player character* does. Entities on all levels exist and can act. Levels will remain constant if the character leaves a level and returns. If the player character is killed or quits his mission a new world will be generated upon return to the game.

3.3 Characters

The Antagonist is a *Manticore*. An evil mythical *Chimeric* monster. This monster spreads death and destruction through the surrounding lands, both personally and via it's goblin minions. Slaying this monster will restore peace to the land and protect the Protagonists home.

The Protagonist is an individual of indeterminate gender and background. Their motivation might be to protect loved ones, rewards of fame and glory, an enforced obligation or something else as determined by the player.

The Goblins are malevolent servants of the *Manticore*, who will search the caverns looking for trouble and will attack and chase the player character if they spot him.

The caverns are also home to a species of large lizard. They are mainly neutral and uninterested in the player character but have the potential to turn hostile. They breath fire.

4 Levels

Game levels are procedurally generated upon entry to the Play State. They will therefore be different for each play through. Rather than being split into distinct levels or states that the game switches between the entire game world is generated as a 3 dimensional array of grid cells. Entities on all levels are active and can interact, move, collide with walls etc. These processes are all resource light. Only the game tiles and entities within the player characters range and line of sight are drawn to the screen.

Treating the game as a 3 dimensional array has the added advantage of ensuring connection of isolated regions generated by the Cellular Automata world generation algorithm in a more organic way, with more natural results than joining these isolated areas on a 2 dimensional level. To put this in context let us examine the cellular automata algorithmic approach.

4.1 Cavern Generation using Cellular Automata Theory

A cellular automaton consists of a regular grid of cells, each in one of a finite number of states, such as on and off. Each cell has a relationship with the 8 cells surrounding it. An initial generation is set by giving each cell a state. Each generation after this changes the state based on a fixed rule, mathematical expression or computer algorithm. (Wikipedia, 2015)

To generate a cave level each cell is given a percentage chance of being a wall, otherwise it is a floor tile. Each iterative generation compares a centre cell with its 8 neighbours, if the majority are floor tiles it becomes a floor tile, otherwise it becomes a wall. Repetition of this process has a "smoothing" effect. Floor tiles group into rooms and passages and walls to make up the solid rock. This process is detailed below.

4.2 Cavern Generation Implementation

Each cell is given a 50:50 chance of being a wall tile or a floor tile. A number of generations are iterated through 2 similar but subtly different rules, changing state as required. the reason there are 2 rules is that following the basic rule often results is large open cavern areas, a little too large for the vision radius determined. An initial smoothing method is added which reintroduces wall tiles into open spaces. This results in a more cavern-like series of looping and twisting corridors. The number of iterations of each is randomised through a small number set (2-5 iterations of the initial smooth and 1-4 of the basic) this results in a variety of differing cavern structures with wider or narrower caverns within a desired range of results. These numbers were arrived at through trial and error and may change in future.

Cellular automata typical smoothing rules (Roguebasin.com, 2015).

Initial Smoothing Rule

For each cell:

```
    P = number of neighbouring tiles which are walls
    if P >= 5 or P <= 1 :
        cell becomes a wall tile
```

Basic Smoothing Rule

For each cell:

```
    P = number of neighbouring tiles which are walls
    if P >= 5
        cell becomes a wall tile
```

4.3 Cellular Automata Weaknesses

Generation of caverns via Cellular Automata has the disadvantage of often generating isolated cave sections. Generally this is corrected by filling in isolated cave sections, connecting these sections after generation, or adding an initial path across or down the screen. This works because the path is wide enough to survive the smoothing process so there will always be a route across or down the level.

Filling isolated sections can greatly reduce the playable area of a level.

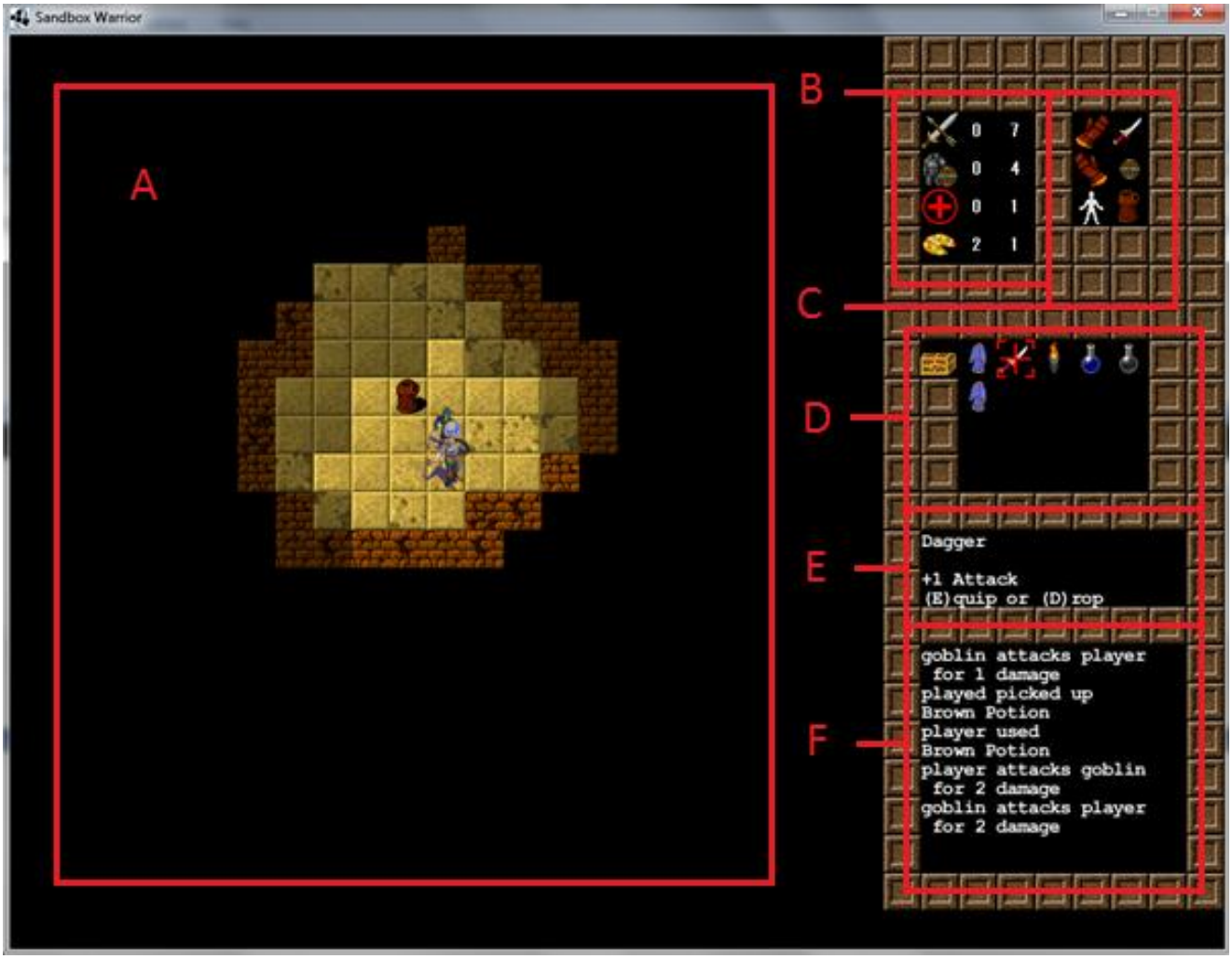
Joining areas after generation or using a path before smoothing can both result in an artificial look that goes against the fundamental aim.

This project follows an online example that instead uses stair up and down where clear regions overlap. This allow cut off sections of a level to be reached by going over or under obstacles' in the way. (Trystans.blogspot.ie, 2011)

5 Interface

5.1 GUI Layout

The Graphical User Interface of the Game Screen is laid out below. There is a large area to the left and centre displaying the portion of the game world around the character. The left hand panel provides information about the player and textual feedback about what is going on.



GUI as seen during play

The GUI is split into a number of sections:

A. Play Area displays the game world as perceived by the PC including player and NPC animations.

B. Shows the current values for the players Attack, Defence, Hit Points and Food.

C. Equipped items box. Shows an icon for the Items equipped in the players left hand, right hand and body.

D. Shows the players inventory, the collection of items carried, using icons. Moving the select box allows interaction with inventory objects.

E. Displays information on the object currently selected in the inventory along with possible commands for that Items.

F. Provides textual feedback and details on recent player actions and actions that effect the character.

5.2 Keyboard Input

A list of the keyboard inputs used:

Game State Controls

'ENTER'	Used to switch between game screens. For example from the introduction screen to the game screen and from the victory or defeat screens back to the introduction screen.
'ESCAPE'	Used to exit the game.

Inventory Controls

'Up Arrow' 'Down Arrow' 'Left Arrow' 'Right Arrow'	Used to move the inventory selection box and the craft selection box.
'p'	Used to pick up Items from the game world and place into the inventory.
'd'	Used to remove Items from the inventory and place into game world at players location.
'u'	Used to activate the effects of an Item, eating food, drinking potions.
'e'	Used to equip an Item to one of the hands or the body. Used with Weapons, Armour, Torches and Shields.
'c'	Used to activate/deactivate the craft box in the inventory. When 2 Items are selected (using the select and craft boxes) and 'c' is pressed the game will attempt to craft a new Item out of both. Also used to cancel the craft selection by selecting an empty inventory slot and pressing 'c'.

PC controls

'NUMPAD 1' If down and left/south-west tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 2' If down /south tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 3' If down and right/south-east tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 4' If left/west tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 5' If occupying stairs up on highest level and player has Manticore remains
player wins, switch to victory screen
If occupying stairs up on highest level and player does not have Manticore remains
player fails mission, switch to LoseQuit screen
If occupying stairs up on another level
player moves up one level
If occupying stairs down
switch to lower level
If not occupying stair tile
player pauses this turn, NPCs take turn as normal

'NUMPAD 6' If right/east tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 7' If up and right/north-west tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 8' If up/north tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

'NUMPAD 9' If up and left/north-east tile relative to the PC is occupied
attack NPC occupying tile
Otherwise
move PC to that tile

6 Artificial Intelligence

6.1 AI

A general AI class that other AI classes extend from. Consists of helper methods to help determine course of action, such as checking line of sight and checking for collisions. Also contains a generic wandering method.

6.1.1 WanderMonsterAI

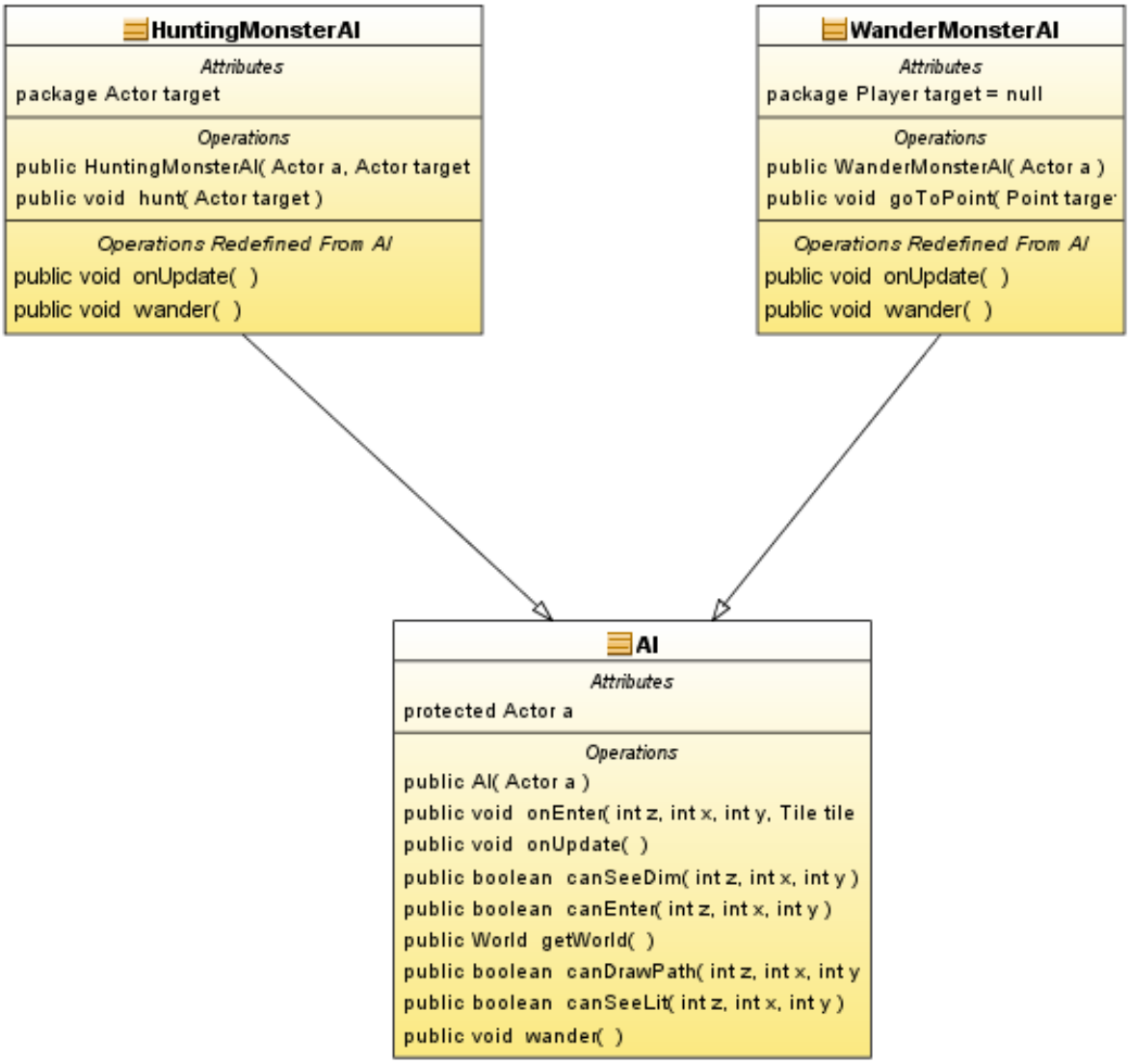
AI class for Wandering monsters, such as Lizards that are generally non hostile, but may attack if player gets too close. If attacked will switch to a HuntingMonsterAI hunting the attacker. Will return to WanderMonster AI if target is killed.

6.1.2 HuntingMonsterAI

AI class for Goblins and the Manticore. Include a reference to the player, who they will hunt when in sight. Will wander if they lose track of target.

6.2 Support AI

The game contains a selection of support AI methods used by the AI files to gather the data needed by the AI to determine actions.



AI class diagram showing inheritance of AI classes

6.3 Path finding

Path finding is via an A* implementation in the "util" folder.

6.3.1 A * Summary

A* uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way.

It uses a knowledge-plus-heuristic cost function of node x (usually denoted f(x)) to determine the order in which the search visits nodes in the tree. The cost function is a sum of two functions:

the past path-cost function, which is the known distance from the starting node to the current node x (usually denoted g(x))

a future path-cost function, which is an admissible "heuristic estimate" of the distance from x to the goal (usually denoted h(x)).

The h(x) part of the f(x) function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, h(x) might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.

If the heuristic h satisfies the additional condition h(x) \le d(x,y) + h(y) for every edge (x, y) of the graph (where d denotes the length of that edge), then h is called monotone, or consistent. In such a case, A* can be implemented more efficiently—roughly speaking, no node needs to be processed more than once (see closed set below)—and A* is equivalent to running Dijkstra's algorithm with the reduced cost d'(x, y) := d(x, y) + h(y) - h(x).

6.3.2 A * Pseudocode

```
function A*(start,goal)
    closedset := the empty set    // The set of nodes already evaluated.
    openset := {start}    // The set of tentative nodes to be evaluated, initially containing the start node
    came_from := the empty map    // The map of navigated nodes.

    g_score[start] := 0    // Cost from start along best known path.
    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)
```

```
    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            if neighbor in closedset
                continue
            tentative_g_score := g_score[current] + dist_between(current,neighbor)

            if neighbor not in openset or tentative_g_score < g_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := g_score[neighbor] + heuristic_cost_estimate(neighbor, goal)
                if neighbor not in openset
                    add neighbor to openset

    return failure

function reconstruct_path(came_from,current)
    total_path := [current]
    while current in came_from:
        current := came_from[current]
        total_path.append(current)
    return total_path
```

7 Technical

7.1 Target Hardware

The game is developed in Java meaning it can be used on platforms with the correct Java Runtime Environment. This also opens up the possibility of porting to Android. Slick2D has an Android version that would have to be used, but the majority of the code is in plain Java and could be reused.

7.2 Development hardware and software

The game is written in Java 8 using the Netbeans IDE. It uses the Slick2d Java Game Library for display windows, game loop, sounds and animation code. This allows game development to focus on actual game logic rather than time consuming "boiler plate" code.

7.3 Slick 2D

Slick2D is an easy to use set of tools and utilities wrapped around LWJGL OpenGL bindings to make 2D Java game development easier. While it is no longer supported there already exists a user forum and multiple resources such as online guides, documentation and tutorials.

It is used in this project to display the game window, handle game states, loading and drawing graphics and sound and to handle input/output.

8 Game Art

8.1 Screen Art

The introduction screen background image is called "Dungeon Crawl" by an Artist called "midhras" and available for download on the Deviantart website. It is an image created from 50 screenshots of game play footage of the Bethesda Softworks game "The Elder Scrolls V - Skyrim". It is available via the following link.

http://midhras.deviantart.com/art/Dungeon-Crawl-289744305

8.2 Terrain, Item and GUI Graphics

The sprites for all terrain tiles, items and GUI elements and icons all come from the Dungeon Crawl project Utumno Tileset. This is a set of public domain 32 x 32 pixel tiles in 3/4 orthogonal view. They have been used in a number of Roguelike games.
From the README.txt:

> *Initially there was rltiles (http://rltiles.sourceforge.net/). This was/is a set of public domain 32x32 tiles for Crawl and Nethack.*
>
> *Over time these were extended, modified, and new tiles were added. Much of this work was done by the Crawl, and, later Crawl Stone Soup team (http://crawl.develz.org/wordpress/). For a long time, these changes/additions were under the "Crawl" license, unable to be used by most other projects.*
>
> *The wonderful developers/artists who work on Crawl Stone Soup have signed off their copyrights on the tiles enclosed in here, returning them back to a state similar to public domain (CC Zero, see LICENSE.TXT). They are free to use for any purpose. However it would be nice for those using them to credit the developers/artists, and state where you received them so others may find and use them as well.*

They are available at the following link:

http://opengameart.org/content/dungeon-crawl-32x32-tiles

9 Secondary Software

9.1 FLICster

FLICster is a FLC file utility for working with unit '.FLC's for Civilization III from Firaxis/Infogrames. FLICster allows you to view Civ3UnitFlcs, export them to several formats for modification, and create new Civ3UnitFlcs from the edited exports.

The FLC format is a file format for animation - a series of still frames, displayed in sequence. CIV III uses FLC files for many purposes, most complex among them for unit animations. Each unit in CIV III has several different animations - run (movement), attack, death, fidget, etc. Each animation is stored in a FLC file. Unfortunately for potential unit creators, CIV III uses a non standard FLC format. For this reason, existing FLC editing programs, such as Jasc's Animation Shop, cannot edit these files. FLICster was written to meet this need.

FLICster is free for non-commercial use.

FLICster is used to generate storyboard images from the unit animation '.FLC' files.

9.2 Photoshop CS2

Storyboard images generated by FLICster are imported to Photoshop. Here they are resized, the backgrounds converted to a transparent layer, and shadows are colour corrected.

10 Management

10.1 Time Management

A supervisor was agreed on the 2nd February 2015, when the proposal was discussed via email.

Initial meeting took place on 6th February.

The submission date for code and report was midnight 19thApril 2015.

This resulted in approximately 11 weeks to plan and implement the project.

A poster was submitted on the 23rd February based on an initial high level plan discussed.

Meetings were arranged for 12.15 every Tuesday and Friday to track progress.

Code implementation began on the 28th February based on the implementation plan discussed in section 11. The week before implementation began was spent on Slick2d and Roguelike tutorials. Time was also spent playing a number of simple roguelikes to develop a better feel for the genre.

10.2 Version Control

The implementation code was hosted in a GitHub repository, allowing for easy version control, backup and code sharing and review.

11 Implementation Iterations

A 15 point plan was developed to better define the project and to set targets. As projects of this size, and the gaming genre and framework chosen were new to the author it was decided that a strictly structured plan with definitive timeframes was unrealistic. A manageable, flexible, iterative approach was chosen instead.

11.1 Determine Objectives

Determine the following

- Target Audience
- Genre and Tone
- Desired Game Play Elements

The target audience was determined by examining RPG player motivations against the features of common Roguelike games. The choice of target audience was used to guide design and implementation choices throughout the project.

The author chose to focus on elements of randomised procedural generation, limited resources and challenging game play partially as a result of fatigue with traditional RPG structure and an attempt to try something new.

11.2 Prepare Environment

Determine following elements

- Language
- Platform
- Editor
- Version Control
- Automated Build Tools
- Libraries

Java was chosen as the development language for a number of reasons. The author feels that Java is unfairly maligned and seen as a bad choice for game development. This means it is rarely chosen to implement Roguelikes so the author could cover new ground and go some way towards confirming or challenging the authors opinions on Java's suitability as a game development language. Familiarity with Java also counters the difficulty of dealing with an unfamiliar game genre somewhat.

Platform was chosen somewhat arbitrarily. The possibility of converting to Android will be examined in future. Theoretically the idea is feasible. Slick2D has an Android version and the code is light on dependancies.

NetBeans was chosen over Eclipse solely on personal preference. GitHub was likewise chosen due to familiarity and Automated Build Tools such as Ant or Maven consider unneccisary.

Slick2D was chosen over LWJGL as it is higher level, wrapped around LWJGL and hardware accelerated, making Slick2D easier to use. It was chosen over JMonkey as JMonkey has a steeper learning curve and is specialised for 3D. LibGDX was a viable choice however the learning curve was found to be steeper/more involved. For the minimal work I required a Framework/library for Slick2D proved to be simplest to set up and learn, with ample tutorials online despite the fact it is no longer supported.

Create a basic application to test environment. A simple full screen game window that printed a message to the screen using Slick2d was the example chosen.

11.3 Input/Output

- Test input and output methods
- Begin Screen layout design
- Make a placeholder character image
- Move character image based on input
- Handle status and other messages to screen

Implementation of this step was routine. The screen layout chosen at this stage was kept, as was the input method. Other components were replaced.

11.4 Map

- Design Map structure
- Begin with displaying map background
- Add ability to scroll
- Add a variety of other map elements, walls for example
- Add character to map, and move around map
- Convert character image to a game entity
- Block movement at map border and at walls

Implementation of this stage was again routine. The foundation of the system began to develop at this stage. The Tiles, blocked variables and basic entities began here but underwent significant changes.

11.5 Data Format and Persistence

- Generate Map based on data read in from file rather than hard coded
- Plain Text/ human readable and editable data to allow simple changes
- Read in Entity data
- Save data to files

Another routine implementation. This allowed experimentation with look and feel of the game and familiarity with the framework. It showed the tileset in use and icons and GUI were developed at this stage. Much of the data was discarded in the planned move to procedural generation.

11.6 Simple Game Structure

- Add simple turn based structure and timing
- Add additional Game Entities
- Implement simple AI

The game began to acquire more of its final structure. The final AI was built on the structure developed here, although the timing and turn based structures had to be redesigned after animations proved buggy.

11.7 Entity Interaction

- Add Creature stats as needed
- Add additional Game Entities
- Implement AI decision making, allow entities to react to the world

Entities acquired their combat statistics and combat in its initial form (without Item or Potion effects) and tested. A simple game was playable at this level.

11.8 Game Data

- Move Player and Entity data to files
- Add Data files for terrain
- Group logically in editable format (UTF-8/CSV files)

Implemented early, but abandoned when information needed to display animations became too large a body. The intention had been to allow a user to enter a small number of entries to a text file to enable changes in graphics without programming knowledge. Once this became untenable it was decided to replace this aspect with a modification screen as part of the program to aid changes in graphics. This tool is beyond the scope of the current project however.

11.9 Items and Inventory

- Add functionless items without properties
- Enable Entity to pick up
- Add properties and stats to item
- Add Inventory and required helper methods
- Add items to world for discovery

The Item and Inventory developed at this stage remain intact in the final game. The ItemFactory class means that creation of new Items is quick and easy. There is scope for a massive growth in variety of objects and the crafting system moving forward.

11.10 Special Effects

- Add special effects such as hunger and health regeneration
- Add effects to player and entities
- Add items that cause effects, potions for example, that can be consumed to produce an effect

Status effects exist as a way of modelling a wide variety of effects. Included in the game are Hunger and Health Regeneration as well as the magical effects of Potions. Anything that effects an attribute of a game entity can be a sub class of the Effects class. Each entity contains a collection of status effects, each updated when the host entity is updated and firing effects as needed.

This structure allows status effects to be added and removed easily and new ones created. Setting something on fire, for example could be modelled by an effect that causes damage every update and can be ended by using a bucket of water. Effects can also serve as the basis for a Spell system and granting monster or player entities special attacks or effects.

11.11 Simple Game Implementation

- Create a simple game environment to test elements
- Generate creatures and items
- Collect items, test combat
- Test collisions and path finding
- Create Win and Lose conditions

A simple game was implemented and demonstrated. Implementation found to be working well at this stage.

11.12 Game Levels

- Add procedural level generation, rather than existing data
- Add functionality to move between levels
- Persist level data upon exit, to allow player to return to an area
- Deploy Entities and Items to the game randomly
- Increase challenge as level gets deeper (stronger, or more enemies)

Levels generated using Cellular Automata method described in document. Binary Space Partitioning generation of Dungeon Generation found to be unsatisfactory and dropped. Slick2D states maintain data persistence on exit. Random deployment of Entities works well.

11.13 Experience and Player Levels

- Add mechanics for Player Character improvement ("Levelling").
- Add special skills as desired
- Determine experience point values for defeating monsters and set point levels for Player improvement

This option was not implemented.

11.14 Non Player Characters

- Add non-hostile or allied Entities such as followers, shopkeepers or quest givers
- Create home base/village if desired
- Add events that happen at home base/between adventures

These options were not implemented.

11.15 Unique Features

Examples

- Plot/Mission generator
- Faction system
- Infinite wilderness generator
- Personality modelling
- Unit Animations

The significant unique feature of this Roguelike is the animated graphics of units instead of ASCII code characters or static Sprites. Approximately 3 weeks of the 7 weeks of development dealt with implementing animations in a satisfactory manner.

12 Conclusions

The project now stands as a completed example of a Roguelike game, with animated graphics. It meets the objective as originally agreed upon at the start of the project. Furthermore it represents a solid basis for moving forward in the development of a game far larger than the scope of the project. The design allows for significant expansion of game elements with little need for new code. As the design and implementation progressed many avenues for improvement and expansion have opened up and given the knowledge and expertise gained in this project appear viable and exciting.

There follows some reflection on specific aspects of the project.

12.1 Fundamental Design Seems Sound

The cavern generation, AI algorithms, combat, item and effects systems all work well. They can handle a variety of configurations and large numbers of instances. Before the frame rate was capped at 60 frames a second the system ran happily at approximately 160-200 frames a second even when updating hundreds of entities in large game worlds.

12.2 Slick2D Animation Issues

Although Slick2D was easy to set up and use and performed very well with unanimated sprites, it was very buggy with animated sprites. The game loop is optimised for constant rather than turn based gaming and the animations would fire while the entity was in the wrong states. Slowing the frame rate improved performance and I suspect the animations code does not do well with many changes before it has finished playing. Attempting to patch a fix took time, made the code messier and ultimately failed to fix the issue. After about 2 weeks of trying to work with the Slick2D animation class the decision was made to refractor the turn loop and handle the animation with state based frame counts and draw commands.

This system resulted in the desired behaviour but approximately 2 weeks of development time out of 7 was lost.

12.3 Binary Space Partitioning Results Unsatisfactory

The original design envisaged BSP generated dungeon structures to contrast and compliment the Cellular Automata cavern generation. However this element was dropped. Although rooms were spaced appropriately the passage generation failed to produce convincing layouts. Passages would extend in random directions without apparent purpose, begging the question why anyone would have dug it out in the first place. Finding a solution would have taken too much work for too little gain.

Alternatives to BSP have been designed by the author but lay outside the scope of this project.

12.4 Effect System Has Great Potential

The system of storing effects on an entity and adjudicating on update proved very simple and quick to implement. The potential for magic spells, blessings, illnesses and special abilities is huge, all using the same structure and mechanics. Since effects have a reference to the Entity storing them any of the Entities variables can easily be effected.

12.5 Tools For Player Modification Will Be Needed

The initial plan to store animation data in a pure text file to allow "modding" without coding was abandoned when the text files because too complex. Tweaking animation offsets would be a boring and frustrating process in any event.

An alternative approach where a tool displayed where, relative to a tile a unit animation would show up, could be adjusted to taste and the system calculates the needed offsets would be better. Since such a tool will be developed the same can be extended into use with tiles also.

The complication was in the lack of a standard size for unit animations in Civ3. Had they been standard like initially hoped, the calculation of offsets would have been routine and the text files simple to use.

12.6 Future Plans

By saving the random seed used to generate a cavern system, the same cavern can be regenerated for a tiny memory cost. This suggests the possibility of expansion into a huge game world, with multiple caverns and monsters all defined by a single *long*. Creating Dungeon and town generators, AI for the rulers of castles, towns and entire states the potential exists for a truly massive game world implementable by an individual.

The use of Java also suggests the possibility of converting to android to open up another platform with a minimum of fuss. The idea seems sound but will need to be examined properly at a later date.

12.7 Final Thoughts

Even a simple Roguelike proved to be a lot more work than I initially expected. However once the initial set up is done the potential for future development seems massive. Having personally tired of many plot driven modern RPGs the idea of a game that was unique on each play through seemed intriguing. The tone is very similar to early Dungeons and Dragons which made me nostalgic and I found the game play genuinely challenging and enjoyable. Although tired coming up to the deadline, I also feel energised and am eager to continue expanding the project in my own time.

I feel the project has been a large benefit to my development as a programmer.

13 REFERENCES

Roguebasin.com, (2015). *Cellular Automata Method for Generating Random Cave-Like Levels - RogueBasin*. [online] Available at: http://www.roguebasin.com/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels [Accessed 18 Apr. 2015].

Roguebasin.com, (2015). *Berlin Interpretation - RogueBasin*. [online] Available at: http://www.roguebasin.com/index.php?title=Berlin_Interpretation [Accessed 19 Apr. 2015].

Wikipedia, (2015). *Cellular automaton*. [online] Available at: http://en.wikipedia.org/wiki/Cellular_automaton [Accessed 18 Apr. 2015].

Trystans.blogspot.ie, (2011). *Trystan's blog: roguelike tutorial 07: z levels and deeper caves*. [online] Available at: http://trystans.blogspot.ie/2011/09/roguelike-tutorial-07-z-levels-and.html [Accessed 19 Apr. 2015].

Gamasutra.com, (2015). *The Anatomy of a Design Document, Part 1: Documentation Guidelines for the Game Concept and Proposal*. [online] Available at: http://www.gamasutra.com/view/feature/131791/the_anatomy_of_a_design_document_.php?page=1 [Accessed 19 Apr. 2015].

GameSpot, (2015). *Crytek: Graphics are 60 percent of a game*. [online] Available at: http://www.gamespot.com/articles/crytek-graphics-are-60-percent-of-a-game/1100-6406893/ [Accessed 19 Apr. 2015].

Schreier, J. (2015). *Metacritic Matters: How Review Scores Hurt Video Games*. [online] Kotaku. Available at: http://kotaku.com/metacritic-matters-how-review-scores-hurt-video-games-472462218 [Accessed 19 Apr. 2015].

Wikipedia, (2015). *List of most expensive video games to develop*. [online] Available at: http://en.wikipedia.org/wiki/List_of_most_expensive_video_games_to_develop [Accessed 19 Apr. 2015].

Staff, Polygon. (2012). *The state of games: State of AAA*. [online] Polygon. Available at: http://www.polygon.com/2012/10/1/3439738/the-state-of-games-state-of-aaa [Accessed 19 Apr. 2015].

Boyer, B. (2015). *Ubisoft: Wii To Rule Them All, Microsoft/Sony Battle Split In U.S/Europe*. [online] Gamasutra.com. Available at: http://www.gamasutra.com/php-bin/news_index.php?story=18389 [Accessed 19 Apr. 2015].

Laws, R. (2009). Player Motivations. In B. S. James wyatt, *Dungeons and Dragons Dungeon Master Guide 2* (pp. 42-50). Renton: Wizards of the Coast.

Laws, R. (2008). The Players. In J. Wyatt, *Dungeons and Dragons Dungeon Masters Guide* (pp. 8-12). Renton: Wizards of the Coast.

14 Additional resources

14.1 Online tutorials

Slick2D tutorials by YouTube user Frums

https://www.youtube.com/playlist?list=PLJOUQy9-kHFdOHHA5TlHsBSuMfPwkML1g

Java 2D game design tutorials by DesignsbyZephyr.

https://www.youtube.com/playlist?list=PL8CAB66181A502179

14.2 Blog

Trystan's blog details a Java Roguelike implementation using ASCII.

http://trystans.blogspot.ie/2011/08/nameless-roguelike-for-tutorial.html

The Cave Generation, Combat, Path finding in this project are based on and adapted from the implementation here. Line of sight calculations, entity movement and inventory were expanded and adapted from implementation detailed here. See Class headings for details.