College of Engineering and Informatics

4BCT Bachelor of Computer Science and Information

Comparing Procedural Level Generation Methods for Video Games

Final Year Report

Gbadebo Adeyela

g.adeyela1@nuigalway.ie

Academic Supervisor: Dr. Patrick Mannion

Table of Contents

Disclaimer

Assets, sprites, and fonts utilised in this project are licensed from the Unity Asset Store and Itch.io and are the sole property of their respective owners. This project is not affiliated with, endorsed by, or sponsored by Unity Technologies, Itch.io or any of their affiliates.

Acknowledgements

I would like to express my gratitude to all those who provided assistance during the course of this project. A special thanks to my academic supervisor, Patrick Mannion, whose encouragement allowed for me to develop my skills and knowledge in the field of procedural content generation.

Furthermore, I would also like to acknowledge with much gratitude, the University, who provided me with the guidelines and deadlines necessary to complete this project.

Glossary

`AdaptDifficulty` – Function that implements adaptive difficulty events.

`addToBoard` – Function in `BoardManager` script that determines whether tiles are generated based off player's line of sight.

`AttemptMove` – Method that checks whether enemy is blocked from moving.

`BoardManager` – Manages the generation of tiles that are laid out on Game Boards.

`BuildEssential` – Function in `DungeonManger` that builds the essential path that connects entrance of dungeon to exit.

`BuildRandom` – Function in `DungeonManager` that generates the random path branches and chambers from the essential path.

`Debug.Log()` – Logs messages in Unity Console.

`DungeonManager` – Script which creates data for dungeons and passes it on to the `BoardManager` script.

`Enemy` – Script that handles enemy's interactions within the game.

`FindGameObjectWithTag()` – Returns `GameObject` that is to be searched.

`ForEach()` – Loop that calls a function for each element in an array/list.

Game Board – Refers to the grid of tiles that the player explores. Game Board is used for the Endless World and dungeons in this project.

`GameManager` – Singleton that handles the different game states in project.

`GameObjects` – Objects in Unity that represent characters, items, and scenery. These objects act as containers for components in a Unity project.

GANs – Generative adversarial network

`MoveEnemies` – Function that controls the enemy movement logic.

`MovingObject` – Script which controls movement logic for game. Scripts such as `Player` and `Enemy` inherit their movement logic from `MovingObject`.

`OnCantMove` – Function that handles the logic for when player or enemy has been blocked from moving.

`PathTile` – Nested class that calculates and keeps tracks of tiles that are adjacent to the current tile in dungeons.

PCG – Procedural Content Generation

`Player` – Script that handles the player's interactions within the game.

PRNs – Pseudo Random Numbers

`RandomRange()` – Returns random number between specified range.

RPG – Role Playing Games

`SoundManager` – Script that manages the sounds which are composed to form the music for this project.

`SpriteRenderer` – Renders a sprite for Unity Game using 2D graphics.

`updateBoard` – Calls `addToBoard` from the `GameManager` script.

VANs – Variational autoencoder

`Weapon` – Script that creates connection between player and weapon. This script also generates the animation used for when the player is attacking an enemy.

`WeaponComponents` – Script that controls the rendering of weapon components.

List of Figures

List of Tables

Chapter 1. Introduction and Project Overview

1.1 Introduction

Procedural content generation (PCG) in video games is the algorithmic creation of data by combining developer-generated assets, algorithms, and pseudo-randomness, this allows video games to generate its own content by following a set of rules. Procedural content generation increases gameplay variety, saves on memory usage, relieves the pressure on developers and improves replay ability for players. This project focuses on how various PCG methods can be implemented to create a roguelike 2D game. This project will be carried out by setting up relevant technologies, taking inspiration from previous roguelike games and following the proposed game design with the intent of achieving all predefined overall objectives.

1.2 Project Brief

This project is designed in Unity game engine and its intended platform is Windows. A roguelike game is chosen for this project as PCG is a fundamental feature in roguelike games. Given that 2D graphics are the most commonly used graphics in roguelike games, this project will be designed utilizing 2D graphics. PCG algorithms in this project will be used for enemy generation, item generation, weapon generation, dungeon generation, music generation, and adaptive difficulty.

The main objective of this project is to construct level components in a modular fashion that allows for easy combinations and for multiple variations of these combinations using various PCG methods.

1.3 Overall Objectives

1.3.1. Setup Technologies

The set up and installation of the technologies Unity, Trello, One Drive, Git for Desktop and GitHub for this project will allow for an easier development process, a structured approach to project management, access to cloud storage, uncomplicated version control and simple repository management.

1.3.2. Develop Application

This project should implement PCG and be comprised of features such as permadeath, adaptive difficulty and grid-based movement to ensure the project incorporates the core mechanics that define a roguelike game.

A wide variety of PCG methods should be used to algorithmically generate the project’s dungeons and enemies. A variety of items and weapons should also be generated using PCG for the purpose of replenishing the players health and adding strength to the player.

Music from existing audio sources should also be added as music helps in creating a sense of atmosphere for the game.

The project should establish and follow a concrete set of rules to guarantee playability for users.

As a result, the project should be a 2D roguelike game which runs on a Windows PC.

Chapter 2. Literature and Technology Review

2.1 Review of Literature

2.1.1. Literature Introduction

PCG is an area of game development that has seen an exponential growth in recent years since its inception in the early 1980s. Coinciding with this growth of PCG has been various definitions and conflicting arguments for its scope of use as a result of large interest from academia. PCG has been viewed as an AI problem (Yannakakis et al., 2018) where levels are viewed as solutions that fulfil the constraint of being playable and maximise some metric (e.g., length) [1]. While viewing PCG as an AI problem is a suitable method of implementing PCG, it does not always reduce the workload of developers. Developers who have often sought out a reduced workload when implementing PCG have suggested (Watkins, R., 2016) that PCG should utilize randomness and random events to produce content [2]. In recent years, PCG has also been seen as a PCG system (Togelius et al., 2016) given that the system incorporates a PCG method as one of its parts [3]. Analysing these studies can assist in having better understanding of PCG and thus helped in implementing PCG for this project.

2.1.2. PCG as an AI problem

Yannakakis et al. have written extensively about how PCG can be viewed as an AI problem. Yannakakis et al. tackle the PCG as an AI problem by suggesting various AI and machine learning methods [1]:

- **Search based methods** search for content with desired qualities using stochastic search or optimization algorithm.
- **Solver-based methods** for PCG uses logic programming, to search for content artifacts that meets a series of constraints.
- **Generative adversarial network (GANs)** where two neural networks battle with each other in a zero-sum game.[4]
- **A variational autoencoder (VANs)** is an autoencoder which has regularised training, deters overfitting and guarantees a latent space that enables content generation. [5]

The advancement of PCG in recent years and the foreseeable future seems to be focused on training generators on existing content to be able to produce more content of the same design and is driven by the results obtained from networks architectures such as GANs and VANs. Though this advancement comes with a downside as these architectures are not well equipped to deal with games with strict structural constraints. This downside is highlighted further in the fact that Yannakakis et al. provide little detail on how each of the four key PCG roles (experience-driven, experience-agnostic, autonomous and mixed initiative) apply to either search based methods or solver-based methods [1].

2.1.3. PCG utilizing randomness

Watkins specifies how randomness, specifically pseudo-random numbers (PRNs), can be used to generate inventory, health items, levels and seeding these levels for future use for the player. This randomness helps in easing the developer's workload while not making the role of developer completely redundant. Watkins also specifies that the quality of run should not feel random but fails to elaborate on how to best decide the bounds for a PRNs (e.g., should the odds of obtaining a health item be 1/10 or 1/100) [2].

2.1.4. PCG as a system

Togelius et al. propose four metaphors for how a PCG system relates to the game design process [3]:

- **PCG system as a tool** which help in elevating the designer's capabilities when working on source code.
- **PCG system that creates a new medium** by defining new kind of materials such as grass, stone, and sand.
- **PCG systems as designers** which undertakes fully autonomous design of parts or possibly even an entire game.
- **PCG systems as domain experts** that hold comprehensive knowledge of game design that can be used to critique or improve designs.

Despite the fact that there are many diverse uses for PCG as a system which provide numerous benefits, it is often too difficult to achieve the goals of being fast, reliable, controllable, expressive, and creative for a PCG system while minimizing the trade-offs that comes with achieving these goals.

2.1.5. Literature Conclusion

Given the fact that PCG is a relatively new area of interest in game development, it is reasonable for there to exist differing opinions on the applications of PCG amongst researchers and developers. It is also reasonable that there would exists some gaps on the topic of PCG such as GANs and VANs not being well equipped to deal with games with strict structural constraints. In accounting for these gaps, this project incorporates methods from all aforementioned literature but will particularly focus on the work by Watkins (PCG utilizing randomness) as variety of level/content generation is a fundamental feature of roguelikes and this can best be achieved using the methods of randomness outlined by Watkins [2].

2.2 Technical Review

2.2.1. Technical Introduction

In seeking to obtain the overall objectives of developing an application that results in a roguelike game after setting up the relevant technologies, it was important to understand the key features of a roguelike game and how these features relate to

PCG. This information can then be used to determine what technologies are best suited to developing an application that results in a roguelike game.

Roguelike games are a subgenre of role-playing games (RPG). The name roguelike is derived from the game *Rogue* which was released by Glen Wichman and Michael Toy in 1980. *Rogue* uses ASCII characters and a tile-based level generation system to generate its levels. *Rogue* is a dungeon crawler which incorporates features such as procedural content generation (PCG) and permadeath. All Roguelike games are games that emulate the fundamental features implemented in *Rogue* which are defined as PCG, permadeath and turn-based gameplay. These features are key to roguelike games and give insight on what is needed in the development of a roguelike game.

2.2.2. Features of Roguelike Game

2.2.2.1. Procedural Content Generation

PCG is the generation of game content algorithmically either autonomously or with limited human input. The game content that can be generated with PCG ranges from weapons, animations, enemies to levels, maps, AI behaviour, textures, items, stories, music, models, and storylines with levels, with AI behaviour currently being the most popular form of uses. An increasing number of studios are currently using PCG as it is quicker and cheaper than human designers. PCG also permits for developers to implement AI that assists in the creation of player-adaptive games that maximize players enjoyment. Players have also been given the opportunity to react to real-time events in a way that would otherwise be unfeasible using a variety of PCG methods:

- **Grammars** is a set of production rules for rewriting strings. Grammars create content without any evaluation functions or re-generation when used as a constructive method.
- **Cellular automata** are used to model environmental system in video games.
- **Noise and fractal** algorithms can be used to generate textures and terrains.
- **Search based methods** search for content with desired qualities using stochastic search or optimization algorithm.
- **Solver-based methods** for PCG uses logic programming, to search for content artifacts that meets a series of constraints.
- **Pseudo Random Numbers** with PCG can be used to determine the outcomes of certain events and can assist in diversifying the content of the game.

The aforementioned methods are not just from AI but also from biology, graphics, and theoretical science. While there is a diverse array of methods that can be used for PCG, in the scope of this project, search based methods and pseudo random numbers were primarily focused on as their ability to search for content with desired qualities, determine the outcomes of certain events and diversify game content are best suited to generating the dungeons/levels that are required for this project [1].

2.2.2.2. Permadeath

Permadeath is a game mechanic in which a game character/player loses all their health and dies permanently. It is a core game mechanic in any roguelike game and works by having the player restart game from the beginning once they have died as there are no checkpoints in the game. Permadeath is implemented in this project as it adds the thrill, urgency, and risks which single player games often lack. Permadeath is usually implemented in roguelike games by wiping save files on reloading of the game.

2.2.2.3. Grid based movement

A grid-based movement system implements the world of the game as a grid and defines a player's position as a (x, y) coordinate which gives the player the ability to move in up to 8 different directions. The player can move orthogonally and/or diagonally. The main difference between grid-based movement and free movement is that the player moves multiple pixels in one movement. The movement in this project is restricted to 4 different directions (up, down, left, right) as to avoid enabling the player to escape from enemies easily. Animation will also be used in the project to disguise the movement of multiple pixels from one location to the other.[6]

2.2.3. Technologies

2.2.3.1. Unity

Unity was first introduced in June 2005 by Unity Technologies as a cross-platform game engine (27 platforms). Unity permits for the development of games with both 2D and 3D graphics along with drag and drop functionality and scripting through C#. Unity is most popular for mobile games and indie game development as most of their focus is on mobile/PC platforms. Unity also supports the development of games on both consoles and Virtual Reality headsets.

Unity utilizes the following graphics APIs: OpenGL ES on IOS and Android; Direct3D on Windows and Xbox One; Vulkan on Android, Linux, and Windows; Metal on iOS and macOS; OpenGL on Windows, macOS, and Linux; WebGL on the internet; and established APIs on video game consoles.

In relation to graphics, Unity provides access to sprites and an advanced 2D world renderer for developers working with 2D games. Unity also provides access to texture compression, resolution settings, dynamic shadows, screen space ambient occlusion, reflection mapping, parallax mapping, bump mapping and a 3D renderer for 3D games.

Unity's interface is comprised of a Scene View, Game View, Hierarchy, Project, and Inspector. [7]

Unity is one of many game engines available for game development currently. Unity is chosen over other game engines, such as Unreal Engine and Cry Engine, for this project as it provides the following advantages when developing a game:

- **Engine is suited for beginners:** Unity being free of charge and offering a user-friendly GUI makes it perfect for students and limited experience developers.
- **Uses C#:** Unity's use of C# removes the frustration commonly experienced when switching from Java to C++.
- **Cross platform integration:** Unity cross platform integration far exceeds their competitors as Unity allows for developers to switch between 27 different platforms. Unreal Engine and Cry Engine pale in comparison both offering 10 and 5, respectively.
- **Asset Store:** Unity has an immense and diverse asset store that offers up to 56,000 packages.
- **Community:** Unity community has many members who can provide help and resources that aid in game development.
- **File Formats:** Unity offers support for many file formats such as Maya, Blender etc.

These benefits make Unity ideal for the development of a Roguelike game which uses PCG.[8][9]

2.2.3.2. C sharp

C# was first developed by Microsoft in 2000 for its .NET initiative and is a general purpose, object-oriented program. C# is a multi-paradigm language which is comprised of imperative, declarative, lexically scoped, component-oriented, generic, strong typing, static typing, and functional programming practices. C# is most often used in the development of web applications and game development.[10]

C# is enabled in Unity by scripting. Scripting functions in Unity by dictating the behaviour of `GameObjects`. Scripts and components attach to `GameObjects` and these `GameObjects` interact with each other to determine gameplay. C# is thus utilized to determine the gameplay of this project.[11]

2.2.3.3. Trello

Trello is a work-based management program founded in 2011. Trello's features include creating a Kanban board, creating a development card, creating/managing tasks and managing checklists. Trello enabled better management of this project given that it permits users to manage workload using a Kanban Board.

Figure 2-1 illustrates …

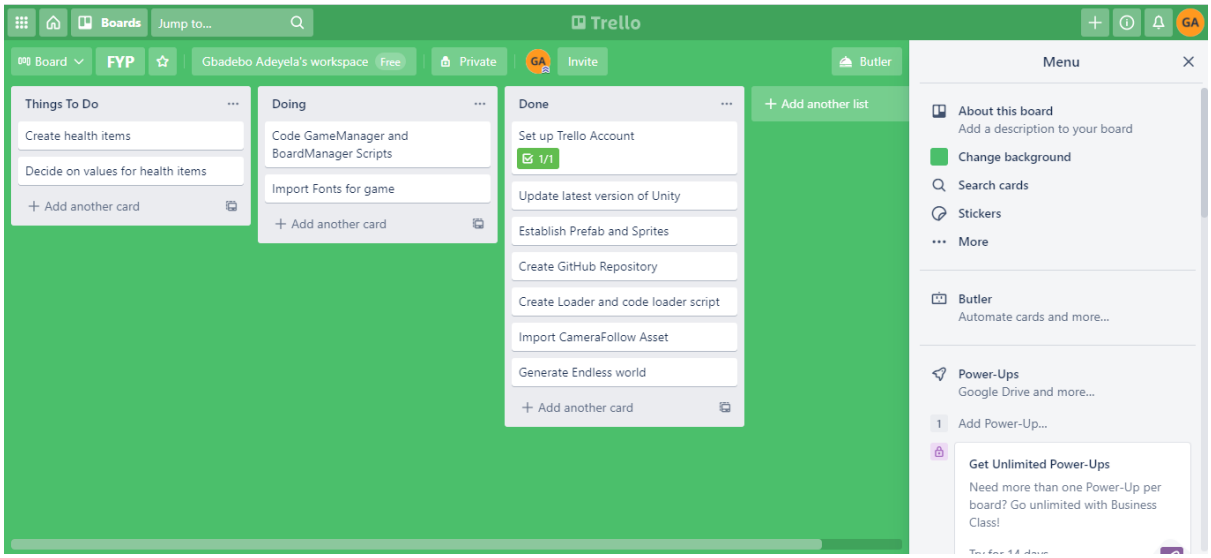


Figure 2-1 Trello Board for project

2.2.3.4. Git/GitHub for Desktop

Git is an open-source program which implements version control that can be used to track, reverse, and merge changes for the project.

Figure 2-2 illustrates …

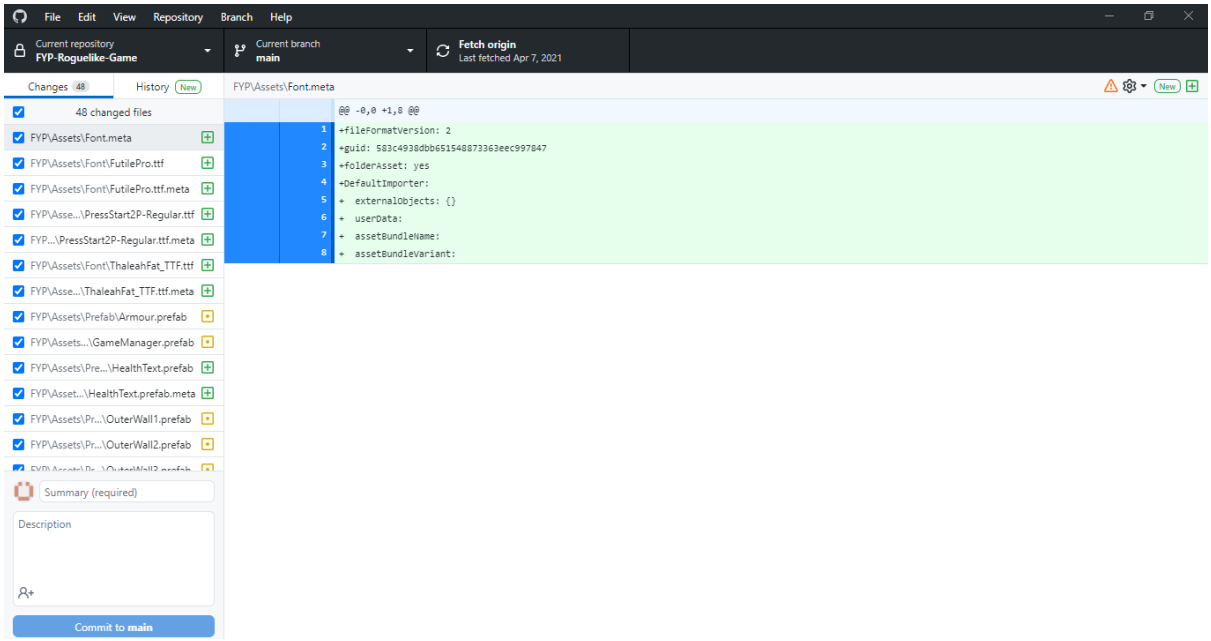


Figure 2-2 Git/GitHub for Desktop commiting changes to GitHub for project.

2.2.3.5. GitHub

GitHub is a cloud-based hosting service for managing Git repositories. GitHub provides access to the project, feature request, bug tracking and continuous integration for the project.

2.2.3.6. One Drive

One Drive is used for cloud storage and is where the Project Definition and Final Year Report will be uploaded and stored.

2.2.4. Technical Conclusion

The implementation of fundamental features of a roguelike game and the equipping of the technologies that are best suited for the project has permitted for the overall objectives of developing an application that results in roguelike game after setting up the relevant technologies to be attained.

Chapter 3. Proposed Game Design

3.1 Sources of Inspiration

3.1.1. Rogue

Figure 3-1 illustrates …

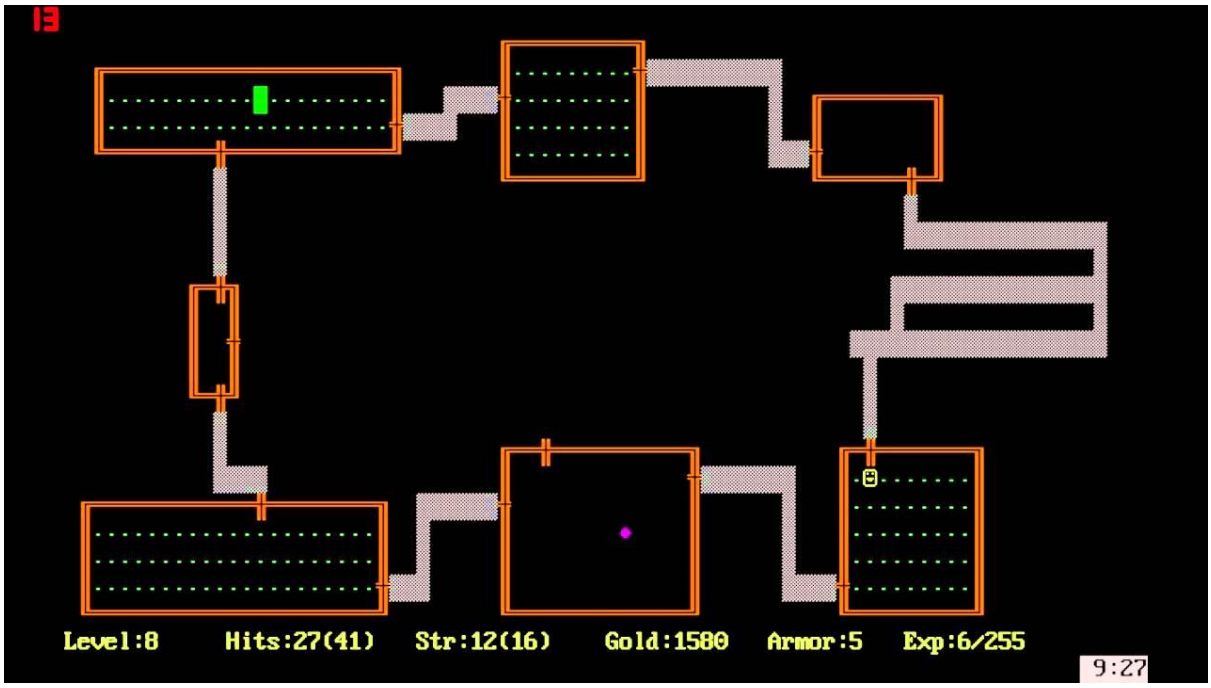


Figure 3-1 The game *Rogue* which is generated using ASCII characters , a tile-based level generation system and PCG.

Rogue put players in the position of an adventurer whose main objective is to obtain the *Amulet of Yendor* by reaching the dungeon's lowest level. Along the way the player will face hordes of monsters who get stronger as the player progresses. The player will also obtain loot and improve their character as they descend further into the dungeons. *Rogue* implements permadeath so that each time a character dies they lose all progress and must start game from beginning. In *Rogue* its layout, loot and monsters are procedurally generated meaning the player faces a new challenge whenever they start a new game.[12]

Rogue serves as the main inspiration for all roguelike games including this project. Its fundamental features of permadeath and PCG can be seen in all roguelike games and are implemented in this project.

3.1.2. Heroes of Hammerwatch

Figure 3-2 illustrates …

Figure 3-2 The game *Heroes of Hammerwatch* is a roguelike action-adventure RPG game developed and published by Crackshell and Sunfire Games.

Heroes of Hammerwatch was first released on March 1st, 2018 and is a roguelike action-adventure game. In *Heroes of Hammerwatch*, the player traverses through procedurally generated levels while encountering traps, puzzles, hordes of enemies, and loot in order to reach the top of the *Forsaken Spire*. Players can upgrade their hero through multiple enhancements as they progress further into their quest. The game is available on Windows and Linux.[13]

Heroes of Hammerwatch use of 2D sprites, modular weapon system, and adaptive enemy AI serve as inspiration for the aesthetics and gameplay mechanics for this project.

3.2 Rules

Rules are a set of statements which must be followed to make games playable for players. The following set of rules are implemented in the project to ensure playability for users:

1. **Quality of run should not feel random:** Even though randomness is used to generate content, this should not come at the expense of the player's skill.
2. **Mistakes should have more impact than luck:** This means if a player fails at the game, it is more than likely their own fault rather than the game's. This provides the player more incentive to improve as the most likely source of the player's failure is their own lack of skill.
3. **Restarts of games should be near-instantaneous:** Allowing the player to restart the game quickly after dying rather than go through multiple menus increases the addictiveness of game to players.

4. **Sessions should not feel futile:** A difficulty curve should be added to the game to give the player a sense of skill improvement as they progress further into the game.
5. **Progress cannot be wiped quickly:** There should be no enemy or any other element in the game that can cause death almost instantly. This will only serve to increase the player's frustration with the game.
6. **Situational Item advantage:** Items should be good in some situations and bad in others. This increases the difficulty of the game and gives the player the ability to determine which items work best depending on the situation.
7. **Rate of difficulty:** The difficulty of game should only increase slightly faster than the progress of player. This forces the play to rely on skill and tactics but does not make the game feel unwinnable.[14][15]

The above set of rules ensure quality of run is based off player's skill, levels are beatable, player has a sense of progression and enables the project to be restarted quickly to make the project more appealing. This all done to increase the playability of the project.

3.3 Graphics

This project implements 2D graphics, sprites and PCG with relatively few art assets to generate a large and complex game world. Assets are obtained from Unity's Assets store. The size of each sprite used is 32x32 pixels. When imported into Unity, the Pixels to Units import setting is set to 32 pixels equals 1 Unity unit of measure so the Game Board aligns with the Unity x-y plane. The assets are rendered using the `SpriteRenderer` component. The sorting layer determines which sprites are rendered first. The prefabs used in the project are as follows:

- **Player** prefab which is the character the player controls. The character has three animations: idle animation, chop animation and hit animation.
- **Floor** prefab which represents the ground in the game.
- **Wall** prefab is used for the walls of the dungeons.
- **Food** prefab holds the food items that helps regenerate the players health.
- **Soda** prefab holds the drink items that helps regenerate the players health.
- **Armour** prefabs holds the glove and boot items which can be added to the player's inventory if acquired.
- **Weapon** prefab which will implement a modular weapon system. Weapon modules will be aligned using a bounding box. There will be total of 12 modules (4 blades, 4 hilts, 4 handles).
- **Chest** prefab represents a chest which spawns items from the armour prefab and swords from the weapons prefab upon interaction with the player.
- **Exit Door** which represent the doorways where upon interaction with the player it will enter/exit the player to/from a dungeon.
- **GloveUI** prefab displays the current glove in the player's inventory.
- **BootUI** prefab displays the current boot in the player's inventory.
- **HealthText** prefab displays the player's health.

- **LevelImage** prefab operates by displaying the “Opening Title” text for the game and also displays the “Game Over” text when the player dies.
- **Canvas** prefab holds the GloveUI, BootUI and HealthText prefabs
- **Enemy** prefab which is controlled using adaptive AI. The enemies have two animations: idle animation, chop animation.

These graphics are used alongside PCG to generate the content to create a game with roguelike features for this project.

3.4 Procedural Content Generation

The following methods reference PCG methods as described by Watkins [2].

3.4.1. Endless World

The world in this project is viewed as grid that has a graph of points on the x-y plane. The world is generated as the player explores. This works by creating a tile-based Game Board that expands as the player moves. The Game Board is potentially infinite as there are no bounds. The algorithm used to create Game Board is a constructive algorithm that uses two types of PCG. Player-triggered PCG to only create tiles the player explores and PCG using random numbers to determine the look of tiles. The PCG algorithm works as follows:

- Player moves one tile in any direction.
- Obtain the direction where player moved.
- Co-ordinates of player’s position is updated.
- Use the position of the player to obtain the six tiles in the player’s line of sight.
- Tiles that have not been explored are added to Game Board and is added to the data structure Dictionary (associative array).
- After these undiscovered tiles have been added to the Game Board, a randomly chosen floor tile is selected to be placed down.
- A randomly chosen food tile is selected to be placed on top of one of the placed down floor tiles.
- Ignore previously explored tiles.

Figure 3-3 illustrates …

Figure 3-3 Endless World being generated by player.

3.4.2. Dungeon Generation

As the world expands as the player moves, dungeons will need be created for the player to explore. This will be done by creating a separate Game Board for dungeons which will be inside the initial Game Board. The initial Game Board for the world will lead the player to instances of the Game Boards created for dungeons. The Game Boards for the dungeons are viewed as grid that has a graph of points on the x-y plane similar to how the Game Board for the world is viewed. Dungeons are enclosed, bounded and finite for the purpose of containing the player within these bounds. Pathfinding will be used to construct a path through the dungeons. The pathfinding algorithm in this project is built on randomness and does not have strict constraints on the direction of generated paths as this project seeks to avoid building the shortest path as a means to increase playability for player. The dungeons will be created by a constructive algorithm that uses system driven PCG and pseudo random numbers to determine the look, size, and shape of the dungeons. The algorithm for dungeon generation is as follows:

- Set up grid (using a square grid means only a single number must be stored).
- The entrance of dungeon is defined on far left of grid by setting x co-ordinate to 0 and randomly selecting the y co-ordinate.
- The **essential path** is generated using the pathfinding algorithm. The essential path is the path which leads from left to right and connects the entrance of the dungeons to their exit. Pseudo random numbers used to choose direction of the essential path.
- Pseudo random numbers are used to specify the probability of branch creation.
- If this probability is met **random path branches** are added to essential path.
- Pseudo random numbers are then used to determine the probability that large opening known as **chamber** will be added to the end of random path. These chambers are generated procedurally to ensure they will fit in grid.

- Once the essential path has been built, the exit to dungeon is then added to the end of the essential path.
- Fill in the rest of the grid with wall tiles that cannot be passed through to encase the dungeon.

The additions of these dungeons produce levels that assist in creating a comprehensive and intriguing world for this project.

Figure 3-4 illustrates …

Figure 3-4 Player exploring dungeon.

3.4.3. Health Item Generation

As the player generates and explores the Game board for the world, pseudo random numbers will be used to determine whether a health item is placed on floor tile and is also used to determine the type of health item (food or soda) and the health it will add to player.

Figure 3-5 illustrates …

Figure 3-5 Player about to acquire health items.

3.4.4. Chest and Armour Generation

Chests in the project are spawned based on a predefined probability set by pseudo random numbers. If the player is blocked from moving by a chest, this chest will then open, and a glove or boot could possibly be spawned derived from a predefined probability utilizing pseudo random numbers. The glove or boot will then be added and displayed in the player's armour inventory. These armour items will then increase the strength of the player by adjusting the attack and defence ratings based on these item ratings:

- Blue Glove/Blue Boot gives ratings of Attack: 1-4, Defence: 1-4
- Green Glove/Green Boot gives ratings of Attack:4-8, Defence: 4-8
- Orange Glove/Orange Boot gives ratings of Attack: 8-12, Defence: 8-12
- Red Glove/Red Boot gives ratings of Attack: 12-16, Defence: 12-16

Figure 3-6 illustrates …

Figure 3-6 Player acquires armour item from chest.

3.4.5. Weapon Generation

The weapon system is achieved by incorporating modular PCG into the project. Modular PCG is beneficial to creating a roguelike game as it allows for minimal art assets to be used to generate a large number of weapons. For example, if there are 3 components (blade, hilt, and handle) and there are 5 modules per components then a total of 125 possible combinations is achieved. The `WeaponComponents` are held by the `Weapon GameObject` as children of the `Weapon GameObject`. The weapon will then be rendered by randomly selecting one module from each component and then combining these components to form a weapon while continuously polling the parent weapon object for its angle so that the components match the angle of the weapon and, subsequently, the angle of the player. Weapons can then be acquired from chest for players based on predefined probability utilizing pseudo random

numbers. These weapons can then be used to battle enemies and increase the strength of the player based off these ratings:

- Steel Sword gives ratings of Attack: 1, Defence: 1
- Gold Sword gives ratings of Attack:2, Defence: 2
- Diamond Sword gives ratings of Attack: 3, Defence: 3

Figure 3-7 illustrates …

Figure 3-7 Player attacking enemy with weapon.

3.4.6. Turn based movement

The movement cycle in this project is turn based. This movement cycle operates by needing player to play a turn by moving and only then is it the enemy's turn to move, and back to the player in a cycle. The attack functionality is incorporated into the movement cycle by allowing the player to attack the enemy any time he is blocked by the enemy and then the enemy can attack the player if he is blocked by the player and back again in a cycle.

3.4.7. Enemy Generation

As the player explores the world/dungeons they will encounter enemies to fight along the way. The enemy will be controlled by AI and will be manipulated by PCG. The enemy AI works as follows:

- Enemies will be randomly generated as the player traverses and discover new sections of the world. Enemies left on the Game Board for the world will be destroyed as player enters the dungeon.
- Any time enemy moves off Game Board/camera for the world they will be destroyed.
- Any time enemy moves off camera for the dungeon they remain still in their last on-camera position in dungeon.
- Enemies left on Game Board for dungeon will be destroyed as player exits the dungeon.

- Enemies will use a pathfinding algorithm to navigate world and reach the player.
- When enemy is blocked from moving the player, the enemy will attack.
- Enemies will be restricted to the Game Board for the world/dungeon.

3.4.8. Adaptive Difficulty

Adaptive difficulty is implemented into the project to avoid the game becoming too easy for the player and to give the player a sense of skill improvement as they progress further into the game. Adaptive difficulty will work by:

- Setting up flags that will determine how and when the difficulty advances. These flags will spawn enemies at a higher rate, increase efficiency of the pathfinding algorithm and ensure enemies will no longer skip a turn attacking enemy.
- Calling a function to check whenever the player powers up (. i.e., obtains item or weapon).
- Adapting AI to check player's position and move enemies towards player according to (x, y) coordinates.
- Adapting AI to make enemy move every turn the player moves. This will help the enemy avoid walking into walls.

3.4.9. Music Generation

The presence of music as the player explores through the world can add atmosphere and tension to the game. Music is implemented in the game through a `GameObject` known as the `SoundManager`. The `SoundManager` is comprised of two audio sources with one audio source that plays the game's sound effects and the other plays the looped background music. The game's sound effects have a low pitch range and high pitch range +5/-5 as a means to produce slight variations in pitch that are not distracting. The sound effects will be played for when the player moves, picks up health item and is attacked by enemy. Once the player dies, the looped background music will stop playing continuously.

3.5 Gameplay and Platform

The player's interaction with the project is a core component as it is a key contribution to the player's enjoyment of the game. Defining the gameplay permits for the project to be tailored to the user's enjoyment. The project will be strictly available to PC (Windows) as it is more accessible than consoles and has better controls than mobile. The player will be able to use a keyboard to play the project. The controls for keyboard will be:

- **Move/ Interact with objects in game:** Up, Down, Left, Right
- **Attack:** Up, Down, Left, Right (when player is blocked by enemy)
- **Restart:** R key
- **End Game:** P key

Chapter 4. Implementation and Testing

Figure 4-1 illustrates …

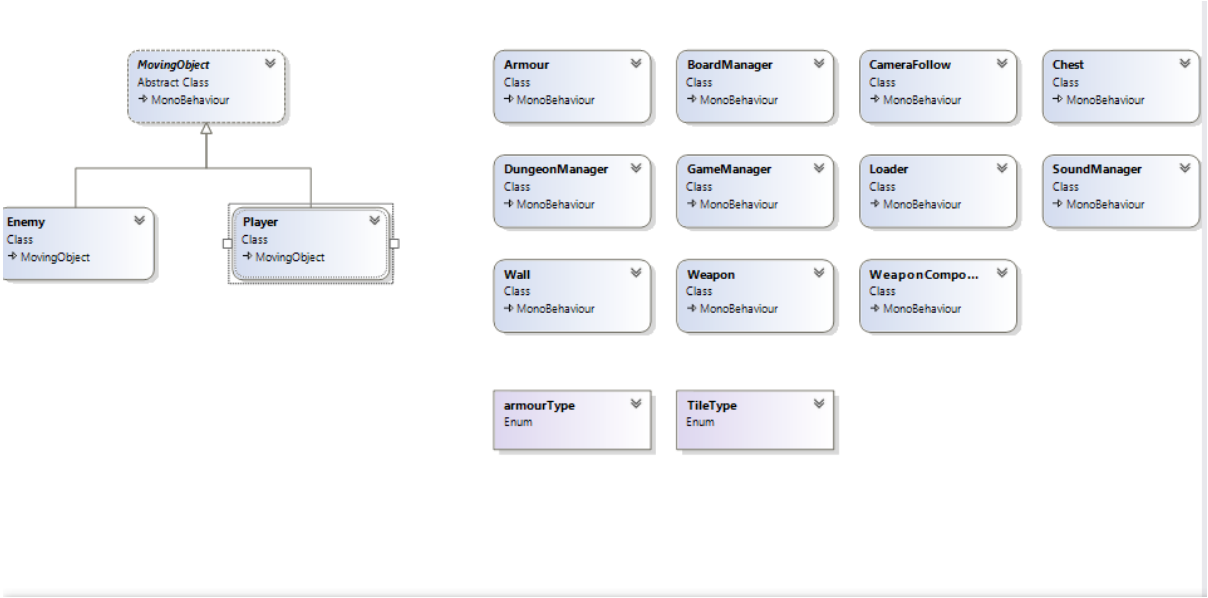


Figure 4-1 Class Diagram for project.

4.1 PCG Methods

4.1.1. Endless World PCG

The algorithm begins by placing the player in the centre of a 5x5 grid. This initial grid is built by looping through a nested for-loop used for rows within another for-loop used for columns. The `BoardManager` script in which the grid is initialised is then called by the `GameManager` script when the game is initialised. When the player moves, the player's current coordinates are held in a static Vector2 variable (Vector2 is used to represent 2D vectors and points) in the `Player` script. A Boolean value in the `Player` script that tracks whether the player is blocked from moving is then set equal to the `AttemptMove` function in the `MovingObject` script which returns false if the player is blocked and true otherwise. If the player can move and is moving, the `GameManager` script calls the `BoardManager` script which loops through a series of if-statements that tracks the player's position/line of sight and instantiates tiles in front of player based on their current position/line of sight. The tiles instantiated are randomly selected from predefined array of tile objects in the Unity Editor.

Figure 4-2 illustrates …

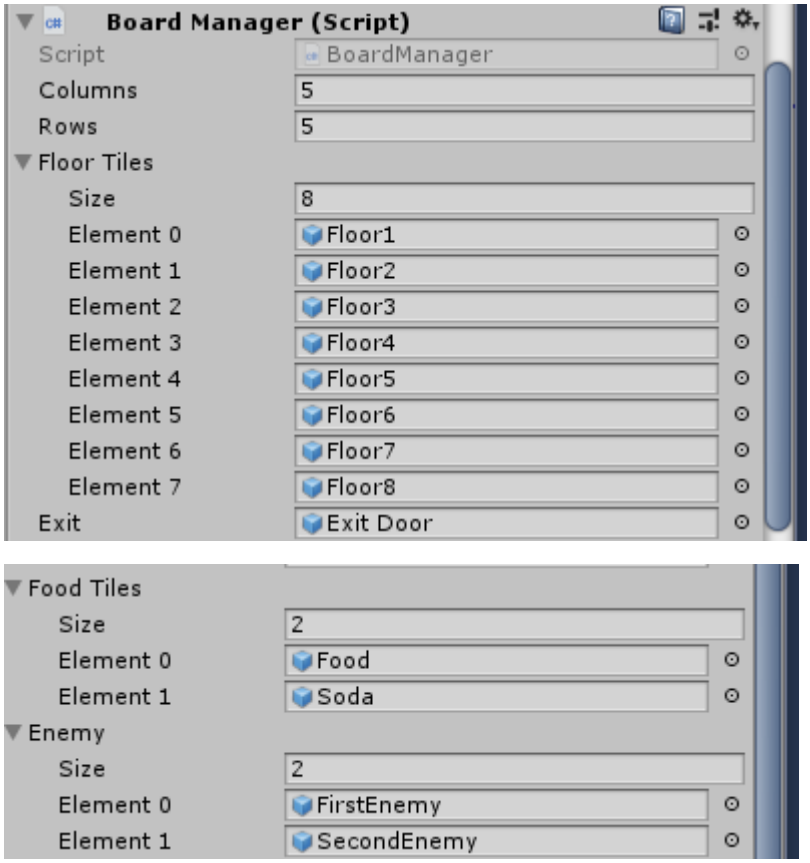


Figure 4-2 Array of floor, health items and enemy tiles `GameObjects` that are used to generate endless world in Unity Editor.

4.1.2. Dungeon Generation and Dungeon Pathfinding

PCG for dungeons starts with specifying a global enumeration in the `DungeonManager` script that keeps track of the tile types in the dungeon. These tile types are either essential, random, enemy, chests or empty. The `DungeonManager` script contains a nested classed referred to as the `PathTile` class. This `PathTile` class calculates and keeps tracks of tiles that are adjacent to the current tile based on the dimensions for the Game Board for dungeons and tiles that have been previously laid out. The `PathTile` class ensures the essential path moves forward (towards the right) by checking that the tile to the left of the current tile is essential.

The `DungeonManager` script the then initialises a Dictionary variable which stores the structure of the generated dungeon. As the game will be generating multiple dungeons, this Dictionary is cleared every time the driver function in the `DungeonManager` script is called. This driver function also randomly specifies the bounds for the dungeon, calls the function to build the essential path and calls the function to build the random path.

The `BuildEssential` function operates by selecting a random y coordinate for the entrance/start position given that the x coordinate is 0. The function also tracks how far along the grid length of the dungeon the essential path is using a local integer variable and uses a while loop to loop through tile spaces in the grid which ends when the essential path has reached the right side of the grid.

The `BuildRandom` function starts with copying the essential path to a Queue/List variable. Built on a predefined probability each essential tile has a chance of creating of random path branches. This same probability is then used to create a chance of a chamber being built at the end of the random path.

The remaining empty tile types are then filled with impassable wall tiles. The tile types are then instantiated and are randomly selected from predefined array of tile objects from the `BoardManager` script in the Unity Editor.

Figure 4-3 illustrates …

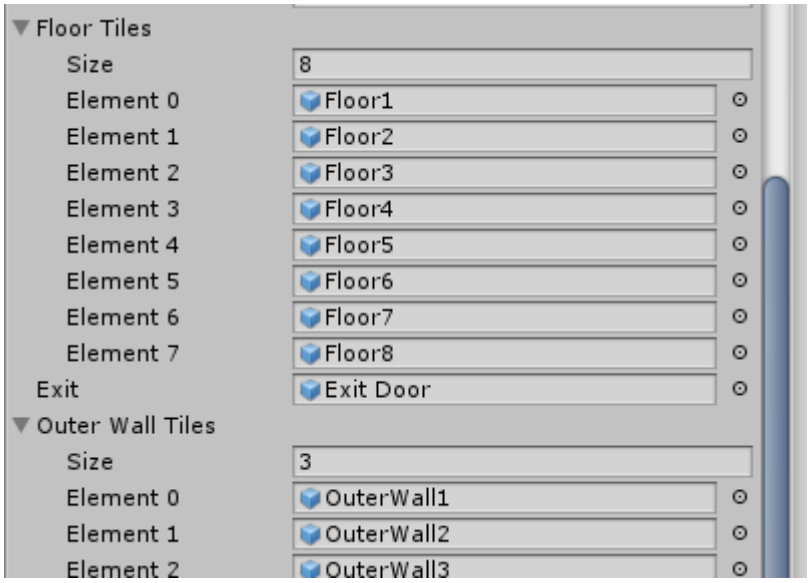
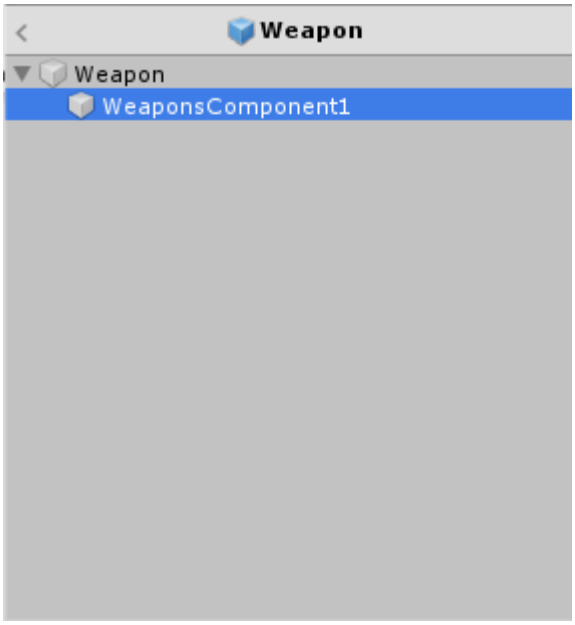


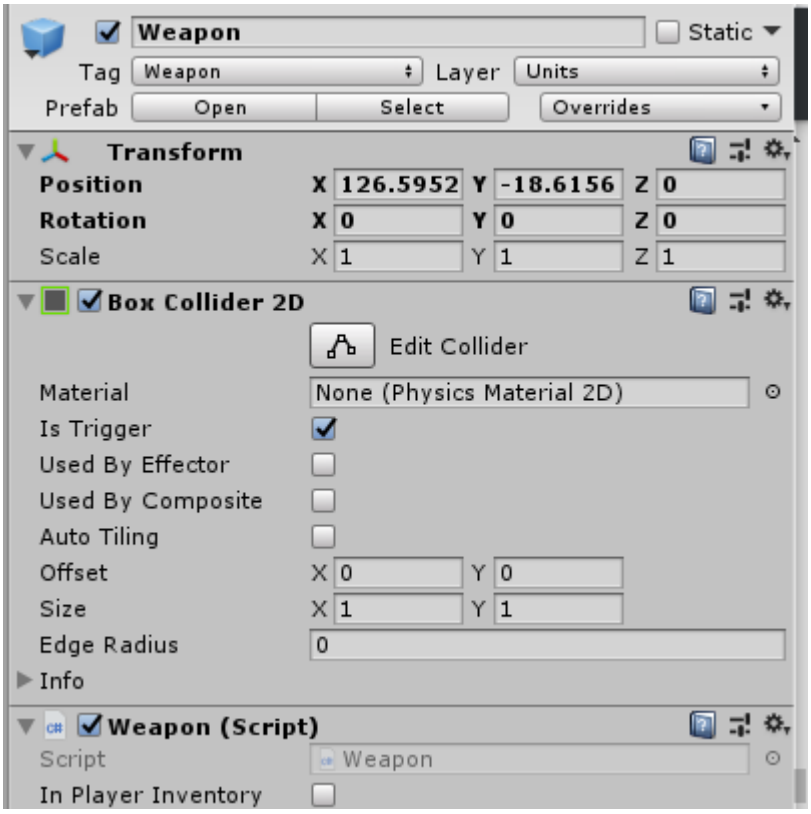
Figure 4-3 Dungeon tile and array of floor and wall tiles `GameObjects` that are used to generate dungeons in Unity Editor.

4.1.3. Modular Weapon PCG

Modular Weapon PCG is implemented with the `Weapon` script that holds a Boolean flag to identify whether or not the weapon was picked up by the player, a reference to `Player` script and an array of references to the `WeaponComponents` child. All references to the `WeaponComponents` child are operated utilizing the `WeaponComponents` scripts. The `WeaponComponents` script also stores the array that holds the weapon modules, calls the `Weapon` script to direct the `WeaponComponents` to turn off and on, stores a reference to the `SpriteRenderer` component to turn it off and on and then randomly selects the module from the array of modules to render.

Figure 4-4 illustrates …


Weapon
Weapon
WeaponsComponent1


Weapon
Static
Tag
Weapon
Layer
Units
Prefab
Open
Select
Overrides
Transform
Position
X 126.5952
Y -18.6156
Z 0
Rotation
X 0
Y 0
Z 0
Scale
X 1
Y 1
Z 1
Box Collider 2D
Edit Collider
Material
None (Physics Material 2D)
Is Trigger
Used By Effector
Used By Composite
Auto Tiling
Offset
X 0
Y 0
Size
X 1
Y 1
Edge Radius
0
Info
Weapon (Script)
Script
Weapon
In Player Inventory

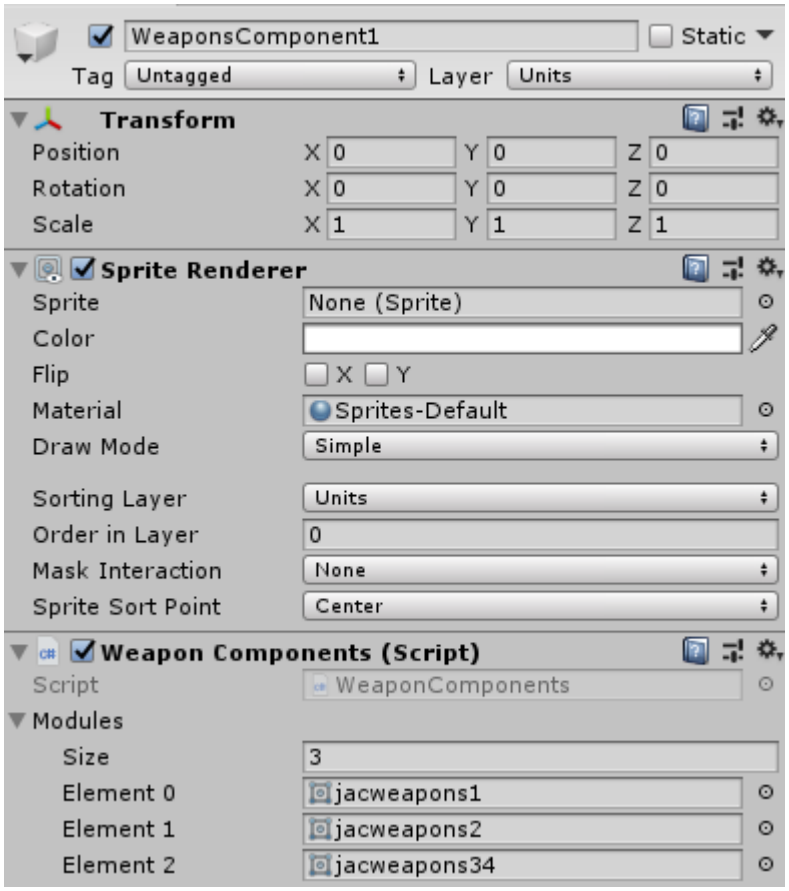


Figure 4-4 `Weapon` and `WeaponComponent` `GameObjects` being shown in Unity Editor and Hierarchy.

4.2 Pseudo Random Numbers

Pseudo Random Numbers are not truly random events but are preferred in games programming as it is easier to generate. Pseudo Random Numbers are implemented in this project using the `RandomRange()` method. The `RandomRange()` method returns a random float within its predefined range (range is inclusive). Pseudo Random Numbers are implemented in this project as follows:

- Randomly selects floor tile to be rendered from array of floor tiles.
- Specifies the probability that food tile is rendered on top of floor tile.
- Specifies the probability that exit tile is spawned.
- Determines the dimensions of the dungeons.
- Choose the direction of the essential path.
- Specifies the probability that a random path branches off from the essential path.
- Specifies the probability that chamber is added to end of random path.
- Controls the probability that a chest is spawned in dungeon.
- Randomly selects either a glove, boot, or weapon.
- Randomly select whether a glove/boot will be blue, green, yellow or red.
- Randomly selects whether weapon will be steel, gold or diamond.
- Randomly selects weapon module for each component.
- Specifies the probability that enemy will be spawned in endless world.

- Specifies the probability that enemy will be spawned in dungeon.
- Randomly selects which enemy type is selected.

Figure 4-5 illustrates …

```
163             //1 in 100 probability that an exit tile is spawned
164             if (Random.Range(0, 100) == 1)
165             {
166                 objectInstantiate = dungeonExit;
167                 instance = Instantiate(objectInstantiate, new Vector3(tileToAdd.x,
168                tileToAdd.y, 0f), Quaternion.identity) as GameObject;
169                 instance.transform.SetParent(holderBoard);
170             }
```

Figure 4-5 `RandomRange()` method being used to specify probability of exit tile spawning.

4.3 Enemy AI and Turn based movement

In a similar manner to the `Player` script, the `Enemy` script is inherited from the `MovingObject` script. The `Enemy` script keeps track of enemies by adding enemies to a List variable, sets the player as a target for the enemy to chase the player by utilizing `FindGameObjectWithTag("Player")`, initially makes the enemy not as fast as the player by ending the enemy's turn if a Boolean condition is met and attacks the player when the enemy is blocked from moving by overriding the `OnCantMove` function from the `MovingObject` class.

The initial enemy AI is also implemented in the `Enemy` script and functions as follows:

- If the enemy and player are on the same x coordinate, then enemy move towards player in the y direction.
- Otherwise, the enemy will move towards the player in the x direction.

The coordinates of the enemies and player are tracked with the **position** property in Unity which keeps track of the positon of `GameObjects`.

The `GameManager` class assists in generating the turn-based movement cycle as it sets up Boolean flag in its `MoveEnemies` function that prevents the player from moving when this flag is active and allows player to move when flag is deactivated.

4.4 Adaptive Difficulty

Adaptive difficulty adds a difficulty curve to the game and gives the player a sense of skill improvement as they progress further into the game. Adaptive difficulty is implemented in this fashion:

- Call the `AdaptDifficulty` function in `Player` script every time an item is picked up.
- In the `AdaptDifficutly` function there are predefined thresholds for the Attack rating for each Boolean flag. These Boolean flags spawn enemies at a higher rate, increase efficiency of the pathfinding algorithm and ensure enemies will no longer skip a turn attacking enemy.
- If threshold is exceeded, then the Boolean flag for that threshold will be activated.

- The enemy AI/pathfinding for enemy is upgraded by implementing a series of If-statements that calculate the shortest path to player and if there is nothing blocking the enemy (checked using Boolean variable), it will take this path. If path is blocked, then process is repeated and if blocked again the enemy will revert to using its initial AI.
- Enemy is made faster by permanently deactivating Boolean variable that cause the enemy to end their turn prematurely.
- Enemies spawn faster by increasing the probability that enemies will spawn as endless world is being generated.

4.5 Testing

4.5.1. Console Window

The console window in Unity was the primary method of testing and debugging the project. It displays errors, warnings and other messages generated by Unity. Log messages can also be displayed in Unity with `Debug.Log()`. The console toolbar and list are used to display, search and filter errors. The detail area of the console shows the full text of the errors and a full stack trace which assists in finding the source of the error. [16]

Figure 4-6 illustrates …

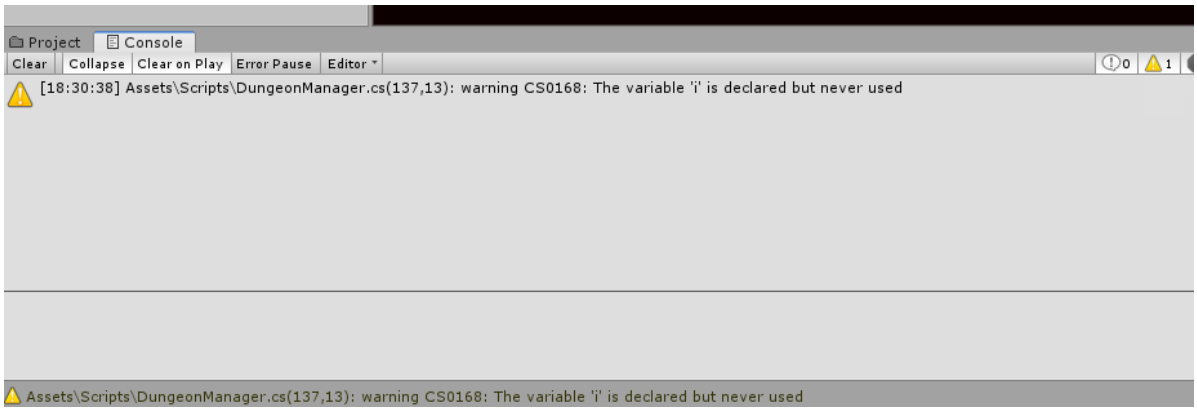


Figure 4-6 Console displays warning in Unity.

4.5.2. Technical Issues

4.5.2.1. Endless World Issue

In the early stages of the project, the endless world was not being generated as the player moved. In attempt to figure out this issue, a `Debug.Log()` message was placed in the `addToBoard` function in `BoardManager` and the game was run. When the game was run, no log message appeared in the console indicating that the `addToBoard` function was not being called by the `GameManager`. The `updateBoard` function was then added to the `GameManager` to call `addToBoard` function from the `GameManager` and thus the endless world could then be generated.

4.5.2.2. Building Random Path/Queue Issue

As the player explores, they will eventually they will encounter a dungeon door that takes them to a dungeon. Initially when the player was being transferred to the dungeon, there was no random path and thus no random chambers being generated. Alongside with the lack of random paths the error "*InvalidOperationException: Collection was modified; enumeration operation may not execute*" was being displayed in the console. Tracing the source of the error from the stack trace in the console window, the error was shown to be the `Foreach()` method that was looping through a List variable and copying the essential path to the List variable. The `Foreach()` method was failing because the List variable could not be looped through while modifying the essential path by adding random path branches. This issue was solved by creating a separate List variable to loop through/copy the essential path and keeping the original List variable that was then used to add random path branches.

4.5.2.3. Insufficient Sprites for Weapon Components

Modular Weapon PCG could not be fully realized as the sprite needed for each of the three proposed components (5 blades, 5 hilts, 5 handles) could not be resourced and creating them from scratch would be too time consuming. As an alternative, only one component was instead set up with each module containing a full sword sprite. While this alternative reduced the number of combinations possible, it greatly reduced the amount of sprites used, thus aiding in reducing the space consumed by the project.

Chapter 5. Conclusion

5.1 Deliverables and Milestones

Table 1-1 illustrates …

Project Definition Document • **Proposed date of completion: 28/11/20** • **Actual date of completion: 23/11/20**	Report detailing the project outline and proposals for tackling it
Technologies Set up • **Proposed date of completion: 21/12/20** • **Actual date of completion: 23/12/20**	Installation and setup of the technologies that will be used in project
Game Development Completed • **Proposed date of completion: 30/03/21** • **Actual date of completion: 25/04/21**	Coding and implementing the game rules, graphics and PCG methods for the project
Testing Completed • **Proposed date of completion: 12/04/21** • **Actual date of completion: 26/04/21**	Playtest to be conducted personally and by academic supervisor
Potential Added Ideas (if there is time is left over) • **The underlined text was implemented in the game with the additional time that was left over for development**	• Story/missions. • Conversing with NPC characters. • Opening menu screen and closing screen. • Sounds for specific items and player movement. • Evaluate PCG algorithms/methods using AI. • Let other users playtest the game.
Final Report • **Proposed date of completion: 3/05/21** • **Actual date of completion: 03/06/21**	Report that analyses the project and evaluates its results
Project Demonstration and Viva Voce • **Proposed date of completion: 10/05/21 – 14/05/21** • **Actual date of completion: 11/06/21**	• Demonstration of project • Interviewed on project by academic supervisor and second marker

Table 1-1 Deliverables and Milestones for project.

5.2 Evaluation

5.2.1. Limitations of Project

5.2.1.1. Modular PCG

As stated previously, the number of possible weapon combinations was drastically reduced from 125 combinations to 3 due to an insufficient number of available sprites. While this setback fails in obtaining the full potential of the code used to implement modular PCG, it was also beneficial as it allowed for a reduced number of art assets to be used and it permitted more time to be allocated to development rather than art creation.

5.2.1.2. Size of Dungeons

Massive dungeons that the player could potentially explore for hours create an immersive world and increases the difficulty curve of player. Although the dungeons in this project are still quite large, they are limited in their size and only offers the player a few minutes of play time. This limitation was discovered during the testing process which showed that any dungeon exceeding the dimensions 250 by 250 will considerably add to the load times of the project as too many wall tiles are being generated. Despite this limitation, it allows for the player to explore more dungeons in shorter space of time which increases replay ability for the player.

5.2.2. Future Work

5.2.2.1. Story

While the primary focus of this project was to compare the PCG methods, it is important that any video game has a story and is an aspect that could be incorporated in the future to further improve the project. A video game's story is important because it assists the player in feeling more included and immersed in the game. It provides meaning/weight to everything in the game and helps the player the understand what is required of them. The advantage of having a story in video games over other forms of media is that the player is a part of the story. [17]

5.2.2.2. Threats

Note: Full threat report can be found in Appendix.

The security of user's data should be a priority to any developer when building a video game. The rise of security vulnerabilities in the video game industry has led to many users feeling less secure when playing video games. Threat modelling is a method that can be used to combat security vulnerabilities in relation to video games. Threat modelling is the process of finding potential security vulnerabilities in the design of an application and eradicating those vulnerabilities prior to writing any code. OWASP Threat Dragon was used for threat modelling for this project. OWASP

Threat Dragon permitted for a DFD to be built which shows which processes and entities are vulnerable in project:

Figure 5-1 illustrates …

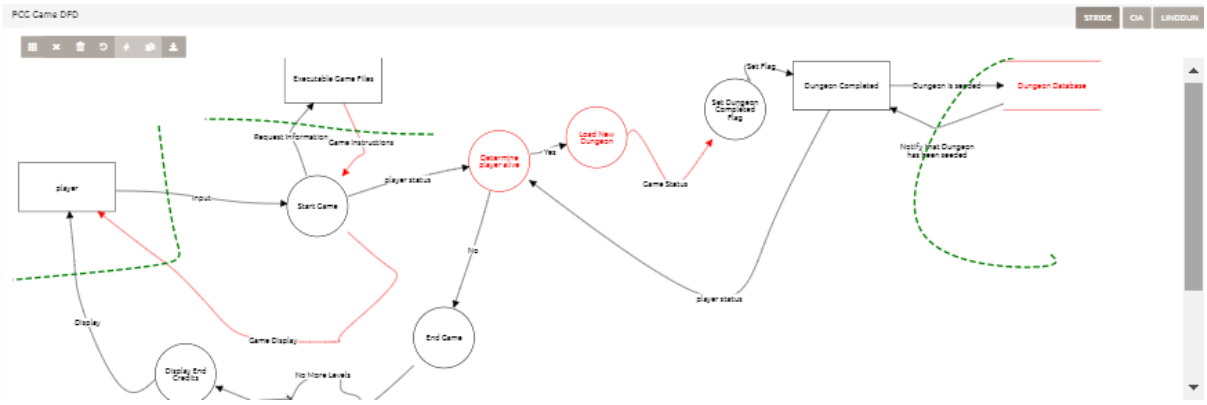


Figure 5-1 DFD Threat diagram for project.

It also allowed for a threat report be generated that displays identified threats (Note: these are just some of the threats, all the threats can be found in threat report in the Appendix):

Figure 5-2 illustrates …

player (External Actor)

Description:
User whos action determine game generation

Bot spoofing threat
Spoofing, Mitigated, Medium Severity

Description:
User could use bot in order to do well in the game.

Mitigation:
Add a Captcha test. As a Captcha test effeciently removes bots that cannot read and supply a correct answer to the test

No Login repudiation threat
Repudiation, Mitigated, Medium Severity

Description:
The lack of a login system in the game means that the game does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation

Mitigation:
Add a login system for the game to track and log users' actions in order to prevent malicious manipulation

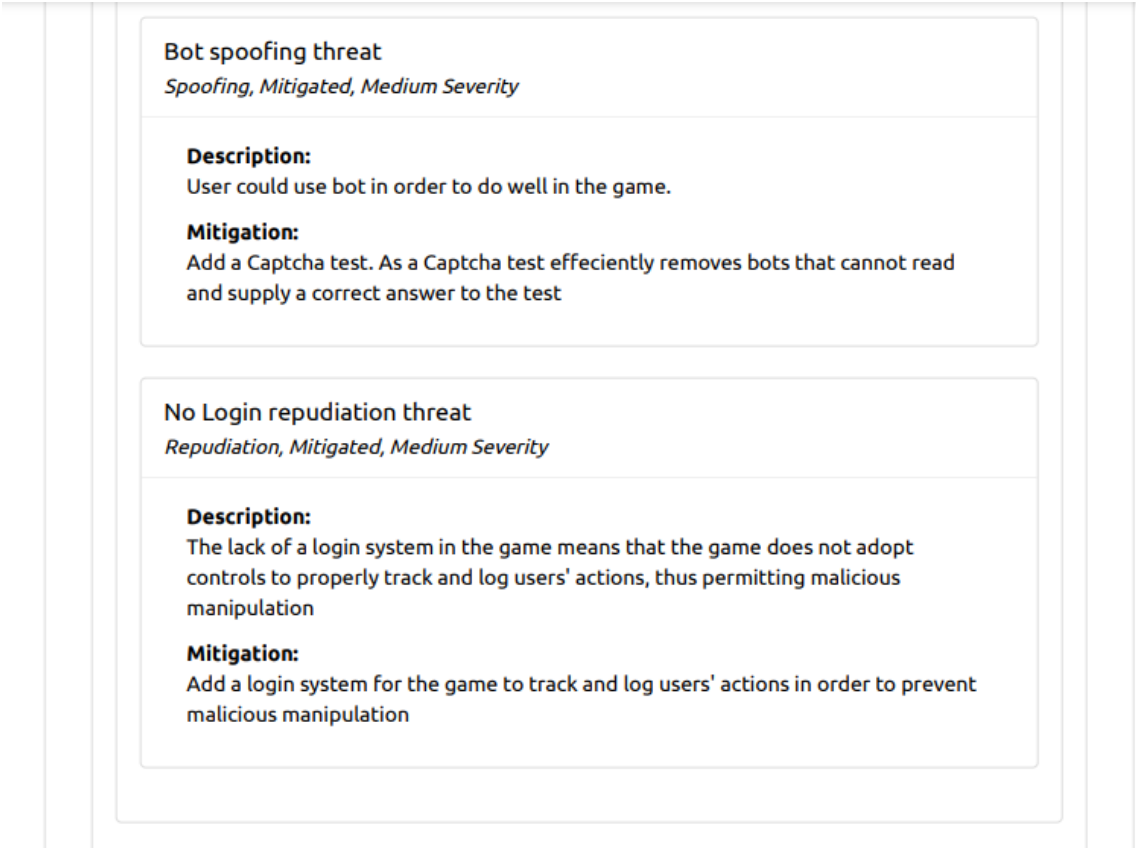


Figure 5-2 Threats from Threat Dragon Report.

The following design changes are proposed for future implementation for a roguelike dungeon crawler game that implements procedural content generation in order to mitigate threats found using OWASP Threat Dragon:

- Captcha acts as the first level of defence for the game given that it provides a Captcha test to any visitor it deems as a threat to game. A Captcha test efficiently removes bots that cannot read and supply a correct answer to the test.
- The addition of a login system reduces the likelihood of manipulation by malware as a login system allows for proper validation of users' inputs and proper encoding of outputs.
- Cryptographic Protocols such as salting are deployed to battle hackers who implements Trojan hacks. Salting prevents hackers who have gained access to one user’s hashed password from gaining access to other users with the same password thus reducing the cost of such an attack.
- Source code files are protected by the addition of a firewall which adds a level of encryption for the game and prevent hackers from performing Local File Inclusion.
- Source code can further be protected with the use of two-factor authentication which can block any suspicious sign-in attempts by requiring access to another authentication factor e.g., phone or email.

- SQL Injection attacks on the dungeon database for the game can be avoided using input validation and parametrized queries which include SQL prepared statements

5.2.3. Closing Remarks

In achieving the proposed game design, primarily focusing on PCG utilizing randomness, selecting the suitable technologies for the development process, and incorporating various PCG methods; a roguelike 2D dungeon crawler game that utilizes PCG was achieved. As a result, the overall objectives of setting up the relevant technologies and developing the application in its entirety for the project was also achieved.

Numerous hours were spent developing this project and there was often a struggle to try and understand the new concepts relating to PCG. There was also many limitations and time constraints faced in the development process of this project. Although these setbacks could have been seen as barriers that could not be overcome, they instead gave the opportunity to exceed past personal limitations and deliver a project that compares PCG methods in video games.

Bibliography

[1] Georgios N. Yannakakis and Julian Togelius (2018) *Gameaibook.org.* Available at: http://gameaibook.org/book.pdf (Accessed: 17 April 2021).

[2] Watkins, R., 2016. *Procedural Content Generation For Unity Game Development*. Birmingham: Packt Pub (Accessed: 17 April 2021).

[3] Julian Togelius, Noor Shaker and Mark J. Nelson (2016) *Procedural Content Generation in Games.* Springer (Accessed: 17 April 2021)

[4] Brownlee, J. (2019) *A Gentle Introduction to Generative Adversarial Networks (GANs)*, *Machine Learning Mastery*. Available at: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/ (Accessed: 17 April 2021).

[5] *Understanding Variational Autoencoders (VAEs)* (2021). Available at: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (Accessed: 17 April 2021).

[6] *Movement: Grid-based · Chaotik Blog* (2020). Available at: https://chaotik.co.za/mechanics/movement-grid-based/ (Accessed: 17 April 2021).

[7] *Unity Game Engine Guide: How to Get Started with the Most Popular Game Engine Out There* (2020). Available at: https://www.freecodecamp.org/news/unity-game-engine-guide-how-to-get-started-with-the-most-popular-game-engine-out-there/ (Accessed: 17 April 2021).

[8] *What is the best game engine: is Unity right for you? (*2020). Available at: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you (Accessed: 17 April 2021).

[9] *CryEngine vs Unreal vs Unity: Select the Best Game Engine* (2018). Available at: https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e#:~:text=While%20matching%20the%20Unity%20vs,difference%20between%20Unreal%20and%20Unity. (Accessed: 17 April 2021).

[10] *Why Is C# Among The Most Popular Programming Languages in The World?* (2017). Available at: https://medium.com/sololearn/why-is-c-among-the-most-popular-programming-languages-in-the-world-ccf26824ffcb (Accessed: 17 April 2021).

[11] *Learning C# and coding in Unity for beginners - Unity* (2020). Available at: https://unity3d.com/learning-c-sharp-in-unity-for-beginners (Accessed: 17 April 2021).

[12] McHugh, A. (2018) *What is a Roguelike? - Green Man Gaming Blog, Green Man Gaming Blog*. Available at: https://www.greenmangaming.com/blog/what-is-a-roguelike/ (Accessed: 18 April 2021).

[13] Hammerwatch, H. (2018) *Heroes of Hammerwatch on Steam, Store.steampowered.com*. Available at: https://store.steampowered.com/app/677120/Heroes_of_Hammerwatch/ (Accessed: 18 April 2021).

[14] *GameDev Protips: How To Design A Truly Compelling Roguelike Game* (2016). Available at: https://medium.com/@doandaniel/gamedev-protips-how-to-design-a-truly-compelling-roguelike-game-d4e7e00dee4 (Accessed: 18 April 2021).

[15] Harris, J. (2020) *Analysis: The Eight Rules Of Roguelike Design, Gamasutra.com*. Available at: https://www.gamasutra.com/view/news/123031/Analysis_The_Eight_Rules_Of_Roguelike_Design.php (Accessed: 18 April 2021).

[16] Technologies, U. (2021) Unity - Manual: Console Window, Docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/Console.html (Accessed: 20 April 2021).

[17] The Importance Of Narrative In Video Games (2017). Available at: https://wibbu.com/importance-narrative-video-games/#:~:text=A%20video%20game's%20narrative%20is,is%20part%20of%20the%20story. (Accessed: 21 April 2021).

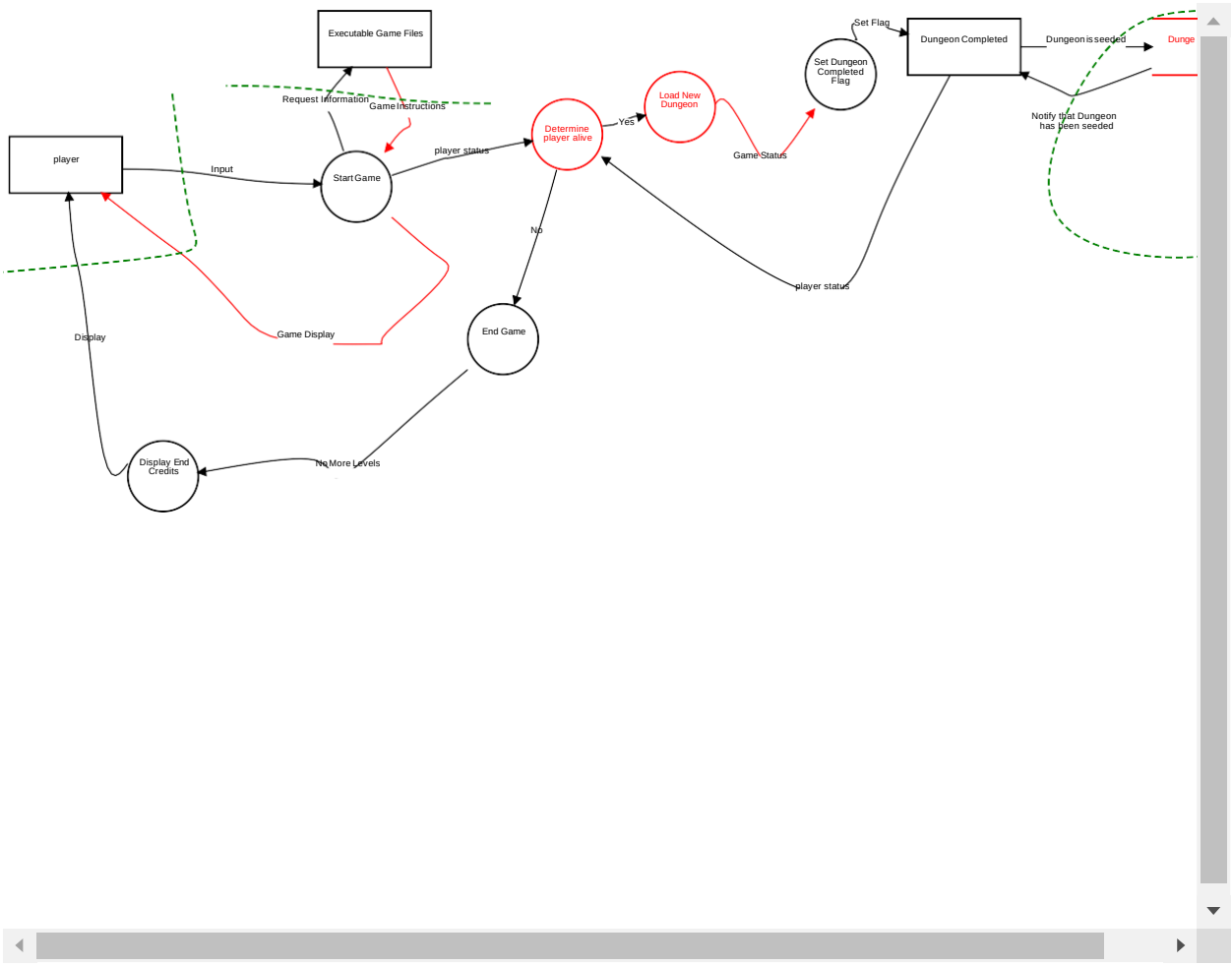
Appendix: Threat Report

Owner:
Gbadebo Adeyela
Reviewer:
Michael Schukat
Contributors:

High level system description

DFD Model for Final Year Project that represents a roguelike dungeon crawler game that implements procedural content generation

PCG Game DFD


Executable Game Files
Request Information
Game Instructions
player
Input
Start Game
player status
Determine player alive
Yes
Load New Dungeon
Game Status
Set Dungeon Completed Flag
Set Flag
Dungeon Completed
Dungeon is seeded
Dunge
Notify that Dungeon has been seeded
No
player status
Display
Game Display
End Game
Display End Credits
No More Levels

Start Game (Process)

Description:
Game is ran

Malware tampering threat

Tampering, Mitigated, Medium Severity

Description:
Game can be tampered with Malware at runtime . This is escpecially true for third party publishers of the game.

Mitigation:
The introduction of authorization and authentication mechanisms for players should prevent tampering by malware

Trojan information disclosure threat

Information disclosure, Mitigated, Medium Severity

Description:
Trojans with keyboard recording function can be deployed at runtime in order to record the keystrokes and the sequence of the user's actions

Mitigation:
Cryptographic protocols are deployed to battle hackers who implements Trojan hacks

Local File Inclusion information disclosure threat

Information disclosure, Mitigated, High Severity

Description:
Local File Inclusion can deployed at runtime of the game by hackers to gain access to files stored on the server

Mitigation:
Add firewall to server to prevent Local File Inclusion from being deployed

player (ExternalActor)

Description:
User whos action determine game generation

Bot spoofing threat
Spoofing, Mitigated, Medium Severity

Description:
User could use bot in order to do well in the game.

Mitigation:
Add a Captcha test. As a Captcha test e''ectively removes bots that cannot read and supply a correct answer to the test

No Login repudiation threat
Repudiation, Mitigated, Medium Severity

Description:
The lack of a login system in the game means that the game does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation

Mitigation:
Add a login system for the game to track and log users' actions in order to prevent malicious manipulation

Determine player alive (Process)

Description:
Game determines if player is still alive

Player Health tampering threat
Tampering, Open, Medium Severity

Description:
Hackers could attempt to modify player character's health so that they never die and the game never ends

Mitigation:

player status (Data Flow)

Description:
The status of the player's health

No threats listed.

Executable Game Files (External Actor)

Description:
Source code of game

Admin spoofing threat

Spoofing, Mitigated, Medium Severity

Description:
User/bot who illegally gain access to admin rights can falsely pretend to be an admin and gain access to game's source code

Mitigation:
Require two-factor authentication tp gain access to admin right so that it is more di cult for hackers to gain access to source code

Request Information (Data Flow)

Description:
Game requested for info from executable files

No threats listed.

Game Instructions (Data Flow)

Description:
Instructions are sent so game can be run

Game Instructions tampering threat
Tampering, Open, Medium Severity

Description:
Game Instructions sent by game could be tampered with by hacker in order to meet hacker's desire

Mitigation:

Instructions information disclosure threat
Information disclosure, Open, Medium Severity

Description:
Inforamtion about the Instructions could possiby be accquired by hackers to manipulate the game for nefarious purposes

Mitigation:

Load New Dungeon (Process)

Description:
A new dungeon is loaded if player is still alive

Manipulating Dungeon Generation elevation threat
Elevation of privilege, Open, Medium Severity

Description:
Hackers could potentially manipulate the game's dungeon generation to continuously generate dungeons in order to eventually crash the player's system

Mitigation:

Yes (Data Flow)

Description:
Player is alive

No threats listed.

Set Dungeon Completed Flag (Process)

Description:
Flag is set for dungeon completed

No threats listed.

Game Status (Data Flow)

Description:
Determine the current status of game

Game Status DoS threat
Denial of service, Open, Low Severity

Description:
The game status could be prevented from updatibg thus preventing dungeoun generation

Mitigation:

Game Status tampering threat
Tampering, Open, Medium Severity

Description:
Status of Game could be tampered with in order to perfom a DoS threat

Mitigation:

Dungeon Completed (External Actor)

Description:
Player has completed the dungeon

No threats listed.

Set Flag (Data Flow)

Description:
Flag is set

No threats listed.

Display End Credits (Process)

Description:

No threats listed.

End Game (Process)

Description:
The end of the game has been reached as player is dead

No threats listed.

No (Data Flow)

Description:
Player is dead

No threats listed.

No More Levels (Data Flow)

Description:
Level Generation is halted after game has ended

No threats listed.

Display (Data Flow)

Description:
End Credits are displayed to player

No threats listed.

Game Display (Data Flow)

Description:
Game is diplayed to the player

Display tampering threat
Tampering, Open, Medium Severity

Description:
Hackers could attempt to manipulate the game's display for their own nefarious puposes

Mitigation:

Input (Data Flow)

Description:
Action inputted by Player

No threats listed.

Dungeon Database (Data Store)

Description:
Database for completed dungeons

SQL injection tampering threat
Tampering, Mitigated, High Severity

Description:
Hackers tamper with the database by using online forms to inject specific SQL code that can then compromise the database

Mitigation:
Input validation and parametrized queries including prepared statements should be used to prevent SQL Injection attack

Dungeon Generation DoS threat
Denial of service, Open, High Severity

Description:
Hackers can deny service of game by not allowing the database to notify the game that dungeon has been seeded thus halting dungeon generation

Mitigation:

Seeded Dungeon information disclosure threat
Information disclosure, Open, High Severity

Description:
Gaining access to database of seeded dungeons could provide hackers with vital info on how dungeons are generated thus giving further insight on how to manipulate game

Mitigation:

Dungeon is seeded (Data Flow)

Description:
Dungeon is seeded so it is not repeated for generation

No threats listed.

Notify that Dungeon has been seeded (Data Flow)

Description:
Notify the game that completed dungeon has been added to database

No threats listed.

player status (Data Flow)

Description:
The status of the player's health

No threats listed.