Table of Contents

[Section 1 – Analysis: 2](#_Toc150418347)

[Problem Identification: 2](#_Toc150418348)

[Introduction: 2](#_Toc150418349)

[Problem Recognition: 2](#_Toc150418350)

[Problem Decomposition: 2](#_Toc150418351)

[Stakeholders: 5](#_Toc150418352)

[Further Investigations: 6](#_Toc150418353)

[Fact Finding: 6](#_Toc150418354)

[Diagrammatic Representation: 7](#_Toc150418355)

[Existing Solutions: 8](#_Toc150418356)

[Possible approaches to solving the solution: 9](#_Toc150418357)

[Proposed solution: 10](#_Toc150418358)

[Scope of Solution: 11](#_Toc150418359)

[Feature of Solution: 12](#_Toc150418360)

[Limitations of Solution: 13](#_Toc150418361)

[Requirement Specification: 14](#_Toc150418362)

[Success criteria: 14](#_Toc150418363)

[Developer’s hardware/software spec: 16](#_Toc150418364)

[User’s hardware/software spec: 17](#_Toc150418365)

# Section 1 – Analysis:

## Problem Identification:

### Introduction:

This report outlines the creation of a top-down shooter game set within a procedurally generated environment. Within this dynamic environment, as players traverse the ever-changing landscape, the system will continuously generate terrain, obstacles, enemy placements, and various in-game elements unpredictably.

Additionally, to enhance player interaction and immersion, the game will feature a Graphical User Interface (GUI) designed for efficient inventory management, containing vital resources such as healing items, ammunition, weapons, and quest-related objects essential for achieving your escape. This document will provide an overview of the analysis of the problem, the design of the solution, the development of the solution and the evaluation of the problem.

### Problem Recognition:

The problem in this scenario is the approach to level generation in top-down shooter games, which typically relies on pre-designed fixed levels. In these games, players often encounter the same layout and elements in each playthrough, diminishing the element of surprise and potentially reducing replay value.

What the user wants is a more engaging and unpredictable gaming experience. The aim is to develop a system that implements procedural level generation, where the game generates levels dynamically as the player progresses through the game. This approach would introduce variability in level design, making each playthrough unique. The new system will incorporate procedural level generation algorithms that create levels on the fly. As the player navigates through the game, the system will generate terrain, obstacles, enemy placements, and other game elements in a randomised and unpredictable fashion.

Furthermore, it is worth noting that the current trend in game development often targets high-end hardware, which can lead to a less enjoyable gaming experience for those with less powerful systems. Players may find it costly and frustrating to achieve a satisfactory framerate by tweaking various video settings. This process consumes time and detracts from the overall user experience, potentially causing frustration. By contrast, the creation of a low-end computer game addresses this issue directly. The aim is to simplify the gaming experience, ensuring that players can enjoy the game smoothly, regardless of the hardware they possess. Having a low-end system requirement allows the game to reach more users. This approach provides a sense of escapism and overall enjoyability, allowing players to immerse themselves in the game world without technical barriers fully.

### Problem Decomposition:

In the development of our top-down shooter game, I will utilize a range of computational methods to boost efficiency, simplify development, and heighten the overall gaming experience. These methods not only optimize system performance but also prioritize resource efficiency, ensuring that the game runs smoothly on a wider range of computer systems for more users to be able to access the game.

Object Orientated Programming:

Object-Oriented Programming (OOP) is a paradigm that utilizes objects to embody entities within a program. These objects are instantiated from distinct classes, each of which defines the attributes and behaviours specific to the corresponding object. The use of OOP means that less code will have to be rewritten and introduces modularity, making it easier to add, remove and maintain features, which saves time and makes debugging easier as bugs are easier to pinpoint as code is structured into separate classes.

Inheritance:

In object-oriented programming, Inheritance is a technique that capitalizes on the concept of a subclass (child class) inheriting methods and attributes from a superclass (parent class). This approach allows similar classes to derive methods from a common parent class, eliminating the need for redundant code rewriting in the project. An example of this is through our player and enemy class, which are subclasses of a ‘character’ parent class that contains collision detection and movement methods.

Abstraction:

Abstraction is the process of removing unnecessary information so that the user only focuses on key details. This project will make use of abstraction as a computational approach by implementing multiple classes with defined behaviours using private attributes. This design enables child classes to access essential functions while preventing them from accessing unnecessary variables employed within those functions. For example, consider the player class, which can invoke the movement method from its parent character class without needing access to the specific values of variables like move speed and move angle, kept as private attributes. This design ensures that the player class is exposed only to the essential functions necessary for in-game movement. The implementation of such abstraction contributes to long-term code maintainability by reducing complexity and hiding unnecessary details, making the code more comprehensible and manageable.

Decomposition:

Decomposition is the process of breaking down a large complex problem into smaller, more manageable, and solvable problems. The solutions are merged to form a final solution to the main problem. This project will implement decomposition as a computational approach during the coding stage. For example, when building the levels system, there will be a randomly generated maze and procedurally generated rooms. Looking at procedural generation, I will first break down the problem into stages. This could include the number of rooms to generate, room dispersion and room contents. Once the procedural algorithm is fully tested and functioning as intended, I can move on to building the maze level. This should guarantee that the game satisfies the user's experience as there would be minimal to no bugs and errors for the user to encounter.

Backtracking:

Backtracking is the algorithm used to generate the procedurally generated levels and the randomly generated maze. The idea of backtracking is solving a problem step by step recursively until the correct outcome has arisen. Recursion is when a function is called upon itself, and performed repeatedly until a base case is met. This method can be used for the maze generation algorithm. An advantage of using recursion is the method attempts every possible solution that exists within the problem. In terms of the maze algorithm, utilizing recursion ensures the maze is complete, and every tile can be accessible. Furthermore, code is easier to write and debug for the developer, which will help speed up development time and ensure fewer bugs are present. However, using a recursive algorithm will have an impact on performance as recursion repeatedly calls upon the function which takes up more memory as variables and functions are replicated in memory and cleared once the solution is complete. Unlike iterative approaches which repeatedly do an action without requiring the replicate memory, doing an iterative solution for the maze generation algorithm will be much harder to write and take up much more development time. This impacts the project by reducing the time for bug fixing and testing.

Heuristics:

Heuristics refers to problem-solving strategies that use practical rules or shortcuts to quickly find approximate solutions, often sacrificing optimality for efficiency. Heuristics would be implemented into graph data structures which can be used when creating the procedurally generated levels, by creating hallways between each node and vertex. Weighted vertices allow for hallways of varying lengths to be generated between rooms. A\* and Dijkstra search algorithm will find the path between the start of the level and all the rooms to ensure that the level is fully accessible and complete. This use of heuristics will reduce unnecessary CPU usage which will increase the efficiency of the game and enable it to run on lower-end systems.

Pipelining and multithreading:

Pipelining is a technique where multiple stages of the FDE cycle are overlapped to improve throughput and overall performance. The project would be written in a language that allows low-level memory access which is useful to create efficient algorithms that will use minimal memory usage, reducing the need for the operating system to send programs and data currently not in use to virtual memory. Additionally, Multithreading is a computing technique that enables a program to simultaneously execute multiple threads, making better use of available CPU resources and potentially reducing overall execution time for tasks that can be concurrently processed.

Visualisation:

Visualisation refers to the graphical representation of data, which will be throughout the entire program when rendering sprites, levels, and graphs for simplifying how to produce the procedural generation algorithm and providing a graphical user interface.

## Stakeholders:

The stakeholders of my project include students seeking a sense of escapism and stress relief. The Computer Science department at Dormers Wells High School, acting as my client, is responsible for distributing the program to their students to cater to their requirements. The computer science department is responsible for providing students with comprehensive education and training in computer science principles.

Dormers Wells High School is in the Ealing borough, with approximately 1,500 students, with the majority of its students enrolled in Computer Science classes. The target audience for this solution primarily comprises students who are currently taking their GCSEs or A levels as they often experience high levels of stress. These students need to have the sensation of relaxation and escapism. The project has been designed with input from students and is intended to be a user-friendly, easily accessible tool that can help relieve the stress often associated with academic studies. Additionally, testing the program within the school environment ensures that it meets the specific needs and preferences of the target audience while also catering to the hardware and software configurations commonly found in educational institutions.

As a student at Dormers Wells High School, my classmates will form the internal testing group for this solution. Their proximity allows them to give me direct feedback, resulting in real-time improvements. Additionally, testing the solution on the school's diverse hardware and software configurations ensures compatibility with a range of systems.

DWHS possesses a range of hardware configurations on a native Windows 10 distribution:

|  |  |  |
| --- | --- | --- |
|  | School computer 1: | School computer 2: |
| CPU | Intel i5-13500T | Intel i5-12600 |
| RAM | 1 x 8 GB DDR4 | 1 x 8 GB DDR4 |
| GPU | Intel UHD Graphics 770 | Intel UHD Graphics 670 |
| Operating System | Windows 10 | Windows 10 |

This range of hardware allows many students to develop, test, and play games on any machine in the school. The final build can be distributed to other students within the school so that they can run it on their machine.

This project has been tailored to meet the unique needs and expectations of its primary stakeholders, the students at Dormers Wells High School who seek a sense of escapism and stress relief during their academic pursuits. By placing accessibility and user-friendliness at the core of its design, the game ensures that students can seamlessly integrate it into their daily routines. The success criteria, from resizable window options to a pause feature, settings customization, and multiple difficulty levels, are carefully chosen to enhance the overall gaming experience, making it both accessible and enjoyable for the target audience.

Moreover, the inclusion of procedural generation algorithms for level and maze creation adds an infinite replay factor, allowing students to play as long as they desire and continuously explore fresh challenges. The game's controls, with familiar WASD movement, right-click aiming, and left-click shooting, resonate with the conventions of shooter games, creating an immersive interaction that students can effortlessly adopt. The addition of an accessible inventory system with the TAB key, combined with the ability to save progress, further reflects a user-centric approach, accommodating longer play sessions without disrupting gameplay.

By actively involving the internal testing group, the creator's fellow students, and considering the diverse hardware and software configurations present in the school environment, the game is meticulously crafted to ensure compatibility across a wide range of systems. This approach aligns with the educational environment of Dormers Wells High School, where students need a game that not only offers relaxation and escapism but also seamlessly integrates with the hardware and software commonly found in the institution. In essence, this project strives to be a valuable tool for students, offering stress relief and enjoyment within the confines of their academic journey.

## Further Investigations:

To optimize the needs of our target audience for the project, I have undertaken a survey involving three representatives selected from our client base, each mirroring the characteristics of our intended users.

Survey Questions:

1. Please state your gender and age.
2. Do you prefer challenging games or easy games?
3. What art style do you prefer in 2D games?
4. Please state the specs of your device: CPU, GPU, RAM

Justification of the question:

Question 1 serves as an audience-oriented inquiry essential for understanding the demographic of the audience. This information is crucial because factors such as age and gender can significantly impact user expectations and, consequently, the overall success criteria for the project.

Questions 2 and 3 aim to determine the desired level of difficulty for the game, ranging from easy to challenging. By collecting data on attention spans, these questions provide valuable insights that inform decisions regarding maze generation, player movement speeds, enemy projectile patterns, and point distributions.

Question 4 delves into the requirements and gauges what specifications are present for the average PC.

### Fact Finding:

Question 1:

3 Males and the age range is 17-18.

These results indicate that the game should appeal to the male gender, although this isn’t significant enough to affect the development of the project. This will become more important as the sample size increases. Additionally, the results of the age range, the game can be a mature game which allows for the game to be relevant to the target demographic.

Question 2:

1 Difficult, 2 Mixture

These results establish the user threshold considering game difficulty and their experience with games in general. By prompting difficulty preferences, the project can base the development stage choices on the results. For instance, enemy difficulty, level generation difficulty, and number of lives to make the game easier or harder.

Question 3:

1 Pixel art, 2 no preference

To satisfy the clients, a simple pixel art style will be used to display graphics. These graphical assets do not have to be high quality or look perfect Which satisfies abstraction and allows greater focus on the features of more significance like the procedural generation algorithm and the maze generation algorithm.

Question 4:

|  |  |  |
| --- | --- | --- |
| Client 1: | Client 2: | Client 3: |
| Operating System: Windows 10 | Operating System: MacOS 13 | Operating System: Windows 10 |
| CPU: Intel i5-10500F | CPU: Intel i7-7820HQ | CPU: Intel i5-2500 |
| GPU: AMD RX 580 8GB | GPU: Intel HD Graphics 630 | GPU: GTX 760 |
| RAM: 1x16GB DDR4 | RAM: 2x8GB LPDDR3 | RAM: 2x4GB DDR4 |

These results provide the hardware requirement to set for the average user. This is a mixture of low-end and mid-end hardware which will not change much about the initial project proposal as the game is targeted for low-end hardware.

### Diagrammatic Representation:

Diagrammatic representation can be crucial for conveying complex concepts, system structures, and data flows visually. Flowcharts are essential for step-by-step outlining algorithms. Maze generation diagrams visually represent generated mazes, aiding in level design. Entity-relationship diagrams establish relationships between game entities, while Use-Case diagrams depict player interactions. Data Flow diagrams reveal data movement within game systems, and class diagrams organize code structure. Graph data structure diagrams can be used to illustrate the randomization process. Each diagram has a specific role in organizing, designing, or implementing game elements.

Top of Form

### Existing Solutions:

There is a wide array of games that provide a sense of escapism. Most of these games come in the form of shooter games or adventure games. Here are a few examples:

|  |  |
| --- | --- |
| Hotline Miami | Minecraft |
| Spend the weekend making 'Hotline Miami 2' levels | A Brief Explanation Of Why Minecraft Matters | TechCrunch |
| Developer: Dennaton Games  Minimum System Requirements:   * **CPU**: 2.8 GHz Intel Core 2 Duo or better. * **RAM**: 1 GB RAM. * **Storage**: 2 GB available space * **GPU**: OpenGL 3.2 compatible GPU with at least 512MB of VRAM * **OS**: 64-bit Windows Vista or later. * **Programming Language**: C++ | Developer: Mojang  Minimum System Requirements:   * **CPU:** Intel Core i5-4690, AMD A10-7800 * **RAM:** 4 GB RAM * **Storage:** 4 GB available space * **GPU:** GeForce 700 Series or later with OpenGL 4.5 * **OS:** 64-bit Windows 10, MacOS, Linux * **Programming Language**: Java |

Both Hotline Miami and Minecraft provide a high sense of escapism, however, they are very different approaches. Hotline Miami uses its fun gunplay and incredible locations. Minecraft uses procedural generation to create infinitely generated worlds. This always makes the game replayable as you will never be in the same location when you start a new playthrough. However, with Minecraft, Infinite terrain generation can be very intensive on the hardware the more you explore. Using a bird’s eye view keeps the system requirements low by having to render limited assets in a particular scene. Developing Minecraft on Java allows for the game to be run on multiple operating systems.

|  |  |
| --- | --- |
| Joy of Creation Reborn | Darkwood |
| The Joy of Creation: Reborn | Free Roam Teaser - YouTube |  |
| Minimum System Requirements:   * **CPU**: Intel or AMD Quad-core, 2.5GHz or faster * **RAM**: 4 GB RAM. * **Storage**: 2 GB available space * **GPU**: GeForce GTX 470, Radeon 6870 HD * **OS**: 64-bit Windows 7 or later. * **Programming Language**: C++ | Minimum System Requirements:   * **CPU**: 2.8 GHz Intel Core 2 Duo or better. * **RAM**: 4 GB RAM. * **Storage**: 6 GB available space * **GPU**: GeForce 8800GT or ATI Radeon HD 4850 * **OS**: 64-bit Windows Vista or later. * **Programming Language**: C# |

In the first level of Joy of Creation Reborn, your aim is to escape the house while being chased down in a randomly generated room, collecting quest items to help you escape the house. This use of the randomly generated maze makes the level always replayable and will always have the same level of intensity when getting chased down. Darkwood is a top-down horror game where you have to explore the woods while having limited vision. The lighting engine used amplifies the immersion of being lost in the woods. These games are both relatively easy to run on modern hardware.

## Possible approaches to solving the solution:

The approach that I propose must involve optimal memory efficiency and efficient use of processing power to maximise performance. Furthermore, the language must be able to utilize a graphics library to make the game possible. Here are the possible programming languages I can use with the included libraries:

|  |  |  |  |
| --- | --- | --- | --- |
| Python | C# | Java | Java |
| Pygame Library | Unity Game Engine | JavaFX Library | LWJGL3 Library |
| Python is an interpreted programming language that executes natively on the system.  Python can make use of OOP.  For implementing a GUI, the library Pygame will be used.  A game engine will have to be developed from scratch.  Additional libraries may be used to improve performance as Pygame does not make a lot of use with a GPU | C# is a compiled programming language that executes natively on the system.  C# can make use of OOP.  The Game Engine Unity can be utilized to help ease workload as the creation of a game engine will not be required. | Java is a compiled programming language which is run on a virtual machine.  Java is fundamentally OOP.  For implementing a GUI, the library JavaFX will be used.  A game engine will have to be developed from scratch.  JavaFX makes use of the system's GPU, but not much use of the GPU, allowing for less strain on the CPU by offloading work to the GPU, overall improving performance. | Java is a compiled programming language which is run on a virtual machine.  Java is fundamentally OOP.  For implementing a GUI, the library LWJGL 3 will be used.  A game engine will have to be developed from scratch.  LWJGL 3 makes more use of the system's GPU, allowing for less strain on the CPU by offloading work to the GPU, overall improving performance. |

## Proposed solution:

Python and Pygame:

Using an interpreted programming language such as Python will be less memory efficient, as an interpreter translates code line by line, meaning that the interpreter needs to keep more information about the program available at runtime and there is no garbage collection so more information about the program is kept in memory until the program is ended. Also, error codes are outputted when the line is executed which can lead to testing errors. For example, if the necessary conditions to execute a specific branch of code within functions or if-else statements are not met early in the program, that code remains inactive, and any potential logic errors within it won't be revealed until those conditions are satisfied later on. Additionally, the use of the external library Pygame will need to be utilized to provide a GUI, although no GPU resources will be used, meaning that all processing will be done on the CPU, creating the least optimal performance especially when the CPU is utilizing multithreading and pipelining. With Interpreted code, the source code is required for the project to run and there is more platform availability as source code is given, which is not architecture specific. However, I do not have much experience with Python so more time will be spent on understanding the language and its libraries.

C# and Unity:

Using a pure compiled programming language such as C# will be more memory efficient, as a compiler translates the entire code. Also, a list of error codes is outputted when the entire source code is complied with. The game engine Unity will be used to help with the workload as it provides GPU support and a fully functioning game engine. This helps reduce workload as less time is spent on developing the main project features like the procedural generation algorithm instead of having to develop a game engine from scratch and having to worry about external libraries. However, I do not have much experience with C# and Unity so more time will be spent on understanding the language and the Game Engine. Furthermore, I am likely to be developing the game on 3 different machines and only my personal computer would have access to Unity and C#. The school computers are only limited to C# which can make the testing and design process longer. Also, pure compiled code, source code is not required for the project to run, meaning that there is less platform availability as source code is not given, which is architecture specific.

Java and JavaFX:

Using a compiled programming language such as Java will be less memory efficient than a pure compiled language like C#, as a compiler translates the entire code within a virtual machine which is then run on a local machine, removing the barrier of architecture-specific code. Also, a list of error codes is outputted when the entire source code is complied with. Java is fundamentally OOP which suits the design of the design. Additionally, the use of the external library JavaFX will need to be utilized to provide a GUI, although minimal GPU resources will be used, meaning that the majority of processing will be done on the CPU, improving performance, but not by much. JavaFX makes designing the GUI much easier due to JavaFX having CSS and FXML incorporated into the library. This means that CSS and FXML can be used to display the GUI, making it much more easier to code, saving development time. Furthermore, I have much more experience with Java than any other language so time will be primarily spent on understanding the libraries.

Java and LWJGL 3:

However, using JavaFX as an external library does have its problems as there is minimal GPU support which can cause problems with performance. This is why I would rather use LWJGL 3, which has stronger GPU support due to the plethora of Graphics API add-ons available in the library such as OpenGL, OpenCL, Vulkan, and CUDA. This allows for much more processing to be offloaded to the GPU, reducing the load on the CPU, especially when the CPU is utilizing multithreading and pipelining. I would primarily use the OpenGL Graphics API as it is much easier to understand and write code for, unlike other APIs like Vulkan where you must write over a thousand lines of code just to display a triangle. Also, OpenGL is primarily used when developing low-end games.

JavaFX and LWJGL 3:

Using JavaFX for the frontend of the game makes it much easier to code as JavaFX has FXML and CSS support. This makes designing GUIs a similar process to designing websites, especially when using the application SceneBuilder which is a visual design tool that simplifies the creation of user interfaces for JavaFX applications by allowing developers to drag and drop components to build scenes. Using LWJGL for the backend to increase performance by utilizing the systems GPU via the OpenGL Graphics API. Using the GPU in parallel with the CPU, which will make use of Multithreading, should increase the performance of the game, allowing for the user to have a more enjoyable experience by having a much more Frames per Second (FPS).

Therefore, I will use Java with the LWJGL 3 library with the JavaFX library due to my current knowledge of Java, ease of frontend design with JavaFX’s CSS and FXML integration and the potential performance to be gained with the use of a GPU via OpenGL.

### Scope of Solution:

The scope of this project involves the development of a top-down shooter game with a primary focus on procedural level generation and innovative gameplay mechanics. The overall aim is to create an immersive gaming experience for students who seek a sense of escapism and stress relief. By embracing the implementation of procedural generation for terrain, obstacles, enemy placement, and in-game elements, each playthrough offers a unique experience. However, it is important to acknowledge the limitations inherent to an A-level project, which imposes constraints on factors such as time, complexity, and available resources. Therefore, while the project will focus on the essential elements of procedural generation, gameplay mechanics, and the unique arm mobility for the protagonist, it will not delve into highly detailed graphics but will focus on the performance of the game so it can run on a variety of hardware.

### Feature of Solution:

|  |  |
| --- | --- |
| Game Features | Justification |
| **Procedural Level Generation** |  |
| Choose and implement a procedural generation algorithm | Enhances replayability and unpredictability |
| Develop parameters and rules for terrain generation | Ensures varied and engaging environments |
| Implement dynamic obstacle placement | Creates challenging and immersive gameplay |
| Randomly generate mazes (1 out of 5 chances) | Adds a unique maze-like level for diversity |
| Generate points of interest (quest-related rooms, hidden passages) | Enhances storytelling and exploration |
| Randomize enemy spawn locations and density | Increases gameplay variety and challenge |
| Design lootable item distribution and rarity | Enhances in-game item variety and balance |
| Adapt procedural generation system to player progress | Maintains an engaging and cohesive world |
| **Game Mechanics** |  |
| Implement character movement mechanics | Provides essential player control |
| Develop responsive shooting mechanics | Ensures immersive gunplay and excitement |
| Create enemy AI behaviours (patrolling, pursuing, attacking) | Provides challenging and dynamic opponents |
| Implement health and damage systems for player and enemies | Adds tension, strategy, and risk management |
| Design inventory management systems | Enhances player resource management |
| Develop inventory management GUI | Provides an intuitive interface for players |
| Implement resource management (ammo, health kits, consumables) | Adds strategic depth to gameplay |
| Include quest-related object interactions | Encourages exploration and puzzle-solving |
| Pause button for game control | Allows players to take breaks and control game flow |
| **Gameplay Features** |  |
| Develop dynamic event system | Adds scripted events based on player actions |
| Integrate random events or encounters | Increases unpredictability and excitement |
| Create variations in enemy behaviour | Maintains suspense and challenge |
| Implement a dynamic lighting system | Enhances immersion and atmosphere |
| Design environmental hazards | Adds tension and tests player skill |
| Ensure unique attributes for points of interest | Enhances exploration |
| Balance distribution of lootable items and resources | Ensures balanced and rewarding progression |
| Customizable key binds stored in a config file | Provides user-friendly customization options |

### Limitations of Solution:

|  |  |
| --- | --- |
| Limitation | Justification |
| Compatibility Across Multiple Platforms | Extending compatibility to various platforms may entail additional development and debugging efforts. |
| Troubleshooting for Diverse Environments | Addressing issues across different OS environments, like Windows and MacOS, can be time-consuming and complex. |
| Demanding Procedural Generation | Implementing complex procedural generation algorithms can strain system resources, impacting performance on less powerful hardware. |
| Art and Sound Asset Creation | Developing art and sound assets for the game can be a time-intensive process, necessitating additional skills. |
| User-Friendly User Interface (UI) Development | Designing an intuitive and user-friendly UI is imperative for ensuring a positive user experience. |
| 2D Game Design | Opting for 2D graphics simplifies development and requires fewer resources, allowing for better performance on a broader range of hardware configurations. |
| School Audience Focus | Prioritizing Windows compatibility caters to the prevalent use of Windows-based systems in educational settings, broadening accessibility for students. |
| Integrated Graphics Hardware | A 2D game design is more suitable for school computers with integrated graphics cards, ensuring smoother gameplay and accessibility. |
| Virtual Machines | Users of Mac, Linux and other Operating Systems can still access the game via virtual machines, ensuring compatibility with their preferred environments, however GPU support may not work. |

## Requirement Specification:

Below are the specific requirements for the project regarding details necessary for features and hardware/software configurations. These requirements are concrete as they are based on research on the problem identified. Additionally, there is an outline of the hardware and software specifications required for development and implementation.

### Success criteria:

|  |  |
| --- | --- |
| Criteria | Justification |
| 1. Must display on a resizable window (640x480 to 1920x1080) or Fullscreen | Ensures usability and accessibility. Resizable window accommodates correct ratio display, and fullscreen supports compatibility for all displays. |
| 1. Start Game button to change from the Main Menu scene to the respective scene | Tests button functionality to avoid unexpected scenarios. Critical for navigating through scenes in a project with multiple scenes. |
| 1. Once the User presses the Start game button, they can select 5 difficulties | Enhances overall gaming experience by catering to different skill levels, providing a range of difficulty options. |
| 1. Load Game button to change from the Main Menu Scene to the game save selector scene | Enables seamless continuation of gaming progress, contributing to a user-friendly and enjoyable experience. |
| 1. Settings button to change from the Main Menu scene to the Settings Scene. The user can change their Graphics, Keybinds, or sound settings | Empowers users to customize their gaming environment, enhancing comfort during gameplay. |
| 1. Graphics settings scene | Offers control over visual aspects, ensuring smooth gameplay on various devices. |
| 1. Keybind setting | Allows users to personalize controls, contributing to a more intuitive gaming experience. |
| 1. Sound setting scene | Enhances immersion, accommodates players in different environments, and caters to those with specific preferences. |
| 1. About button to show information about how to play the game | Provides straightforward game instructions, reducing frustration, and improving enjoyment for players. |
| 1. Exit Button to close the game application | Offers a standard way for users to exit the game, ensuring a seamless conclusion to the gaming experience. |
| 1. The Player can press ESC to pause the game | Offers a user-friendly way to pause the game, providing control and flexibility during gameplay. |
| 1. When the Player pauses the game, they can click on the settings button to change settings | Allows players to customize their gaming experience, enhancing user-friendliness. |
| 1. When the Player pauses the game, they can save their game | Enables players to continue from where they left off, accommodating longer play sessions. |
| 1. Levels will consist of procedurally generated rooms using a recursive backtracking algorithm | Utilizes an algorithm for endless gameplay, allowing users to play as long as they want. |
| 1. Levels will consist of randomly generated mazes using a recursive backtracking algorithm | Utilizes an algorithm for endless gameplay, allowing users to play as long as they want. |
| 1. The player can move with WASD keys or whatever the user wants to configure | Allows users to configure controls for accessibility and playstyle preferences. |
| 1. Player can aim their equipped weapon with a right-click | Mimics shooter game controls, creating a more immersive interaction. Configurable for accessibility. |
| 1. The player can shoot by aiming and clicking left-click | Standard control for shooter games, providing a natural and intuitive gameplay experience. |
| 1. Access inventory with the TAB key | Allows quick management of items without disrupting gameplay. Configurable for accessibility. |
| 1. The player can kill enemies by shooting at them | Fundamental to core gameplay mechanics, providing a sense of achievement. |
| 1. Once the player dies, the game screen shows, showing score, kills, and levels completed | Displays game results, encouraging replay for a higher score. |

### Developer’s hardware/software spec:

Below are hardware and software requirements, with justifications, for developers:

|  |  |  |
| --- | --- | --- |
| Component | Minimum Requirement | Justification |
| Operating System | Latest 64-bit version of Windows, macOS, or Linux | A PC operating system is required for developing a game in IntelliJ as external libraries are required, IDE and other intensive computations. |
| CPU | Any modern 64-bit CPU with multiple cores and threads | Programming in a high-level language requires a CPU with at least more than 1 core or else the program will run poorly. Multithreading will be used to efficiently manage CPU resources, improving performance. |
| RAM | 8 GB of total system RAM | At least 8GB of primary storage is required as many programs such as the IDE, code documentation for OpenGL and more. RAM allows these programs to be open and run smoothly, ensuring a quicker and less frustrating development process for the developer. |
| GPU | OpenGL capable GPUs | OpenGL is a graphics API which allows for systems CPU and GPU to communicate with each other, improving performance by equalizing workload for the CPU and GPU. |
| Storage | At least 5 GB of free space | This is required so that IntelliJ can be installed properly with enough space for additional libraries and enough space to create projects. |
| Input device | Keyboard and Mouse | A keyboard will be required to write the code, needed for the development of the game. A mouse is required to navigate the IDE and files. |
| Software | IntelliJ, JDK 20 or newer, LWJGL 3.3.3 library, Git/GitHub, OneDrive | The IntelliJ IDE is required to access Java code. The Java JDK is required to compile and run Java code. I am using JDK 20 as that is what the school computers use and what my personal computer uses. LWJGL 3 is the external library that provides GUI and Graphics card support, allowing for increased performance. Git allows for the code repository to be uploaded online so I can access code on multiple computers without having to configure any settings. OneDrive allows me to upload any other documentation, or screenshots to my school account so I can access it on other computers |

Here are the 3 different computers I am likely to be developing with the specifications:

|  |  |  |  |
| --- | --- | --- | --- |
|  | School computer 1: | School computer 2: | Personal computer: |
| CPU | Intel i5-13500T | Intel i5-12600 | AMD Ryzen 7 4800H |
| RAM | 1 x 8 GB DDR4 | 1 x 8 GB DDR4 | 2 x 8 GB DDR4 (16GB total) |
| GPU | Intel UHD Graphics 770 | Intel UHD Graphics 670 | Nvidia RTX 2060 |
| Operating System | Windows 10 | Windows 10 | Windows 10 |

### User’s hardware/software spec:

Below are hardware and software requirements, with justifications, for users:

|  |  |  |
| --- | --- | --- |
| Component | Minimum Requirement | Justification |
| Operating System | The latest 64-bit version of Windows 10 | A PC operating system is required to run the game application for clients. |
| CPU | Any modern 64-bit CPU with multiple cores and threads | A CPU with 1 to 2 cores will cause the game to run poorly. Multithreading will be used to efficiently manage CPU resources, improve performance, and create a smooth gameplay experience |
| RAM | Minimum: 4 GB of total system RAM  Recommended: 8 GB of total system RAM | At least 4GB of primary storage is required to run the operating system itself and the game as RAM allows more programs to be open and run smoothly. 8 GB of RAM is recommended as Windows 10 takes up 4GB depending on what programs and background processes are running on the computer. |
| GPU | OpenGL capable GPUs | OpenGL is a graphics API which allows for systems CPU and GPU to communicate with each other, improving performance by equalizing workload for the CPU and GPU. |
| Storage | At least 100 MB of free space | The game is designed to run on a Windows platform. Non-volatile storage is required to store game-related files such as user settings and high scores are available next time the program is opened. |
| Input device | Keyboard and Mouse | A keyboard and mouse are normally the standard with 2D shooter games. Allowing for all the key binds to be configured allows for keyboard or mouse-only gameplay, in the case that a user has access to only a mouse or a keyboard. |
| Software | JDK 8 or newer, OpenGL | A JDK is required to run any Java application, these are commonly installed on Windows computers and within school environments. OpenGL drivers may need to be installed but this is unlikely as Windows is already pre-packaged with OpenGL support. |