Table of Contents

[Section 1 – Analysis: 2](#_Toc147416623)

[Problem Identification: 2](#_Toc147416624)

[Introduction: 2](#_Toc147416625)

[Problem Recognition: 2](#_Toc147416626)

[Problem Decomposition: 2](#_Toc147416627)

[Computational methods: 4](#_Toc147416628)

[Stakeholders: 6](#_Toc147416629)

[Further Investigations: 7](#_Toc147416630)

[Fact Finding: 7](#_Toc147416631)

[Diagrammatic Representation: 8](#_Toc147416632)

[Existing Solutions: 9](#_Toc147416633)

[Possible approaches to solving the solution: 9](#_Toc147416634)

[Proposed solution: 10](#_Toc147416635)

[Scope of Solution: 11](#_Toc147416636)

[Feature of Solution: 11](#_Toc147416637)

[Limitations of Solution: 12](#_Toc147416638)

[Requirement Specification: 13](#_Toc147416639)

[Success criteria: 13](#_Toc147416640)

[Developer’s hardware/software spec: 14](#_Toc147416641)

[User’s hardware/software spec: 15](#_Toc147416642)

# 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:

1. **Procedural Level Generation:**

* Choose and implement a procedural generation algorithm (Dijkstra and A\* shortest path algorithm and more).
* Develop parameters and rules for terrain generation.
* Implement dynamic obstacle placement, ensuring varied and challenging environments.
* Have 1 out of 5 chances for the level to be within a randomly generated maze.
* Define algorithms for generating points of interest (quest-related rooms, hidden passages)
* Establish rules for randomising enemy spawn locations and density.
* Design lootable item distribution and rarity to enhance gameplay variety.
* Ensure the procedural generation system adapts to player progress and maintains a cohesive game world.

1. **Game Mechanics:**

* Implement character movement mechanics.
* Develop responsive shooting mechanics, incorporating aiming, firing, and reloading.
* Create enemy AI behaviours, including patrolling, pursuing, and attacking.
* Implement health and damage systems for both the player and enemies.
* Design inventory management systems, including item pickup, use, and discard functions.
* Develop an inventory management GUI, featuring item sorting and quick access.
* Implement resource management, including ammunition tracking, health kits, and consumables.
* Include quest-related object interactions such as keycards to access rooms.

1. **Gameplay Features:**

* Develop a dynamic event system that triggers scripted events based on player actions.
* Integrate random events or encounters to surprise and challenge the player.
* Create variations in enemy behaviour to maintain suspense and unpredictability.
* Implement a dynamic lighting system to create an atmosphere.
* Design environmental hazards that add tension and require player skill to navigate.
* Ensure that generated points of interest have unique attributes.
* Balance the distribution of lootable items and resources to maintain gameplay progression.
* Customisable Key binds which are stored within a config file.

A screen shot of a computer screen

Description automatically generated

Computational methods:

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. The school has access to Dell OptiPlex AiO Computers configured with a 13th Gen Intel® Core™ i5-13500T (24 MB cache, 14 cores, 20 threads, 1.60 GHz to 4.60 GHz Turbo, 35W), Intel® UHD Graphics 770 and 8 GB: 1 x 8 GB, DDR4. Additionally, they have access to lower-end hardware like the older generations Dell OptiPlex AiO computers configured with a 12th Gen Intel® Core™ i5-12600 (18 MB cache, 6 cores, 12 threads, 3.30 GHz to 4.80 GHz Turbo, 65 W), Intel® UHD Graphics 670 and 8 GB: 1 x 8 GB, DDR4.

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.

## 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. 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.

Therefore, I will use Java with the LWJGL 3 library due to my current knowledge of Java and the potential performance to be gained with the use of a GPU.

### 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:

The primary feature of the project is the procedurally generated levels. This was chosen as a key feature to enhance the game’s replayability and deliver an unpredictable experience. The reason I chose this feature is to establish the recognition of the problem that traditional, fixed-level designs in top-down shooter games often lead to repetitive gameplay experiences. By implementing procedural generation algorithms, the game can dynamically generate terrain, obstacles, enemy placements, and in-game elements in a randomized fashion, ensuring that each playthrough is unique. This choice aligns with computational methods such as backtracking and heuristics, where recursive algorithms and practical rules are used to efficiently generate maze-like levels that are both challenging and engaging. These levels will progressively get harder and longer to complete, the more levels you complete. This creates an infinite game loop, allowing for unlimited replayability.

Additionally, the unique gunplay and inventory management can provide a sense of escapism, and enjoyability and can create some intense situations. To create an immersive environment, certain weapons will have to be reloaded manually by dragging onto each bullet and inserting it into the barrel of the firearm. Normally with shooting games, when you reload a weapon, you press a single button on the keyboard and an animation plays which then reloads your weapon, especially for weapons which do not have magazines. There is less immersion this way, which can deter away from the sense of escapism. Having an inventory system that is easy to use with minimal menus, simple to find what you need and that can show all vital information needed makes it easy for players to get as much information in a single button and get back to the game as soon as possible. This is possible by showing a health bar, healable items, key items, weapons, and ammunition. These items can be stored within a linked list to preserve the order of the inventory. Using an intuitive gunplay style and inventory management allows for a better experience for the players as they are more immersed in the game.

### Limitations of Solution:

The game can be programmed so it can be compatible with multiple devices if necessary. However, troubleshooting for different devices such as on a Windows 10 environment and MacOS environment can become difficult as the code might work perfectly fine on a Windows environment but might not run at all or have performance problems on a MacOS environment. Having to fix code for 2 completely different environments will take time away from development. If MacOS consumers want to play the game, they can run the game within a virtual machine which has a Windows install. Additionally, considering the target audience is students within an educational setting, Windows-based systems are more prevalent. Schools often rely on Windows machines due to their compatibility with various educational software and tools. By prioritizing Windows compatibility, the game becomes more accessible to the intended users, ensuring that a broader audience can enjoy the stress-relief experience.

Another limitation is that the game will be 2D. While 3D graphics can offer a more immersive and visually stunning gaming experience, it comes with significant challenges and resource demands. Developing a 3D game involves complex modelling, texturing, and rendering processes that require more time and expertise. Additionally, 3D games often demand higher computational power and better graphics hardware, limiting the accessibility of the game to players with lower-end systems, which is counter to the project's goal of providing a stress-relief game for students at the school. The school computers all have integrated graphics cards which are relatively underpowered and will struggle in creating 3D games. In contrast, opting for a 2D game design proves beneficial as 2D graphics are simpler to create and require fewer resources, making them more suitable for an A-level project. This allows the focus to be on procedural generation, gameplay mechanics, and overall performance, ensuring the game can run smoothly on a wider range of hardware configurations.

## 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 as the screen size must be large enough to display the scene at a correct ratio. Needs to be resizable or full-screen to allow compatibility for all displays. |
| 1. Start Game button to change from the Main Menu scene to the respective scene | Due to the project using several scenes, the button will need to be tested to ensure that it goes to the correct scene to avoid any unexpected scenarios or issues. |
| 1. Levels will consist of procedurally generated rooms using a recursive backtracking algorithm | Using an algorithm to generate a new level, makes the game endless, allowing the user to play as long as they want. |
| 1. Levels will consist of randomly generated mazes using a recursive backtracking algorithm | Using an algorithm to generate a new maze, making the game endless, allowing the user to play as long as they want. |
| 1. The player can move with WASD keys or whatever the user wants to configure. | Allowing the default controls to be like conventional movement controls allows users to get right into the game as they are already familiar with the controls. This control can be configured so that the user has more accessibility to whatever playstyle they prefer. |
| 1. Player can aim their equipped weapon with a right-click | The controls of the game are very similar to a shooter game as you aim and shoot but in some other 2D games, you just shoot and do not have to aim, creating a more immersive interaction. This can be changed within settings to allow more accessibility |
| 1. The player can shoot by aiming and clicking left-click | The control for the game is very natural for all shooter games as right-click to shoot is standard. |
| 1. Access inventory with the TAB key | It allows players to quickly manage their items and resources during gameplay without disrupting the action. This can be changed in settings. |
| 1. The player can kill enemies by shooting at them | Allowing the player to eliminate enemies by shooting at them is fundamental to the core gameplay mechanics. |
| 1. Once the player dies, the game screen shows, showing score, kills, and levels completed | Players can see the results of the game, rather than just return them to the menu with nothing. It also encourages users to play again to get 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. |