Computer Science Coursework Project

A LEVEL computer science nea project

Ilyasse Boudjemline  
candidate no: 8362  
centre: loreto college  
centre no: 32361

2024

Contents

[Analysis 1](#_Toc151983052)

[Problem Definition 1](#_Toc151983053)

[Background to the Problem 1](#_Toc151983054)

[Computational Approaches 1](#_Toc151983055)

[Stakeholders 3](#_Toc151983056)

[Identifying stakeholders 3](#_Toc151983057)

[Research the Problem 3](#_Toc151983058)

[Research of Existing Solutions: 3](#_Toc151983059)

[Stakeholder Feedback: 6](#_Toc151983060)

[Essential Features 8](#_Toc151983061)

[Limitations 8](#_Toc151983062)

[Proposed Solution 9](#_Toc151983063)

[Hardware and Software Requirements 9](#_Toc151983064)

[Success Criteria 10](#_Toc151983065)

[Design – Decomposition 12](#_Toc151983066)

[Top-Down design 12](#_Toc151983067)

[Explanation of Processes 14](#_Toc151983068)

[Character Movement: 14](#_Toc151983069)

[Progression: 14](#_Toc151983070)

[Character Actions: 15](#_Toc151983071)

[Graphics: 15](#_Toc151983072)

[Sounds: 15](#_Toc151983073)

[Menu: 15](#_Toc151983074)

[Justification 15](#_Toc151983075)

[Planning Ahead 15](#_Toc151983076)

[Iterative Development Approach 15](#_Toc151983077)

[Stages 16](#_Toc151983078)

[Prototype 1 17](#_Toc151983079)

[Planning and Design 17](#_Toc151983080)

[Overview of Structure 17](#_Toc151983081)

[Screen Designs & Usability Features 18](#_Toc151983082)

[Algorithm Designs 22](#_Toc151983083)

[Key Variables & Data Structures 31](#_Toc151983084)

[Validation Methods 35](#_Toc151983085)

[Testing Approach 36](#_Toc151983086)

[Development 40](#_Toc151983087)

[Stage 1 40](#_Toc151983088)

[Stage 2 41](#_Toc151983089)

[Stage 3 51](#_Toc151983090)

[Forms of Validation Used 53](#_Toc151983091)

[Stakeholder Feedback & Reviews 54](#_Toc151983092)

[Requirements 54](#_Toc151983093)

[Feedback & Summary 54](#_Toc151983094)

[Prototype 2 56](#_Toc151983095)

[Planning & Design 56](#_Toc151983096)

[Overview of Structure 56](#_Toc151983097)

[Screen Designs & Usability Features 59](#_Toc151983098)

[Algorithm Designs 62](#_Toc151983099)

[Key Variables & data Structures 68](#_Toc151983100)

[Development 71](#_Toc151983101)

[Stage 4 71](#_Toc151983102)

[Stage 5 72](#_Toc151983103)

[Stage 6 73](#_Toc151983104)

[Stakeholder Feedback & Reviews 74](#_Toc151983105)

[Requirements 74](#_Toc151983106)

[Feedback & Summary 75](#_Toc151983107)

[Prototype 3 76](#_Toc151983108)

[Planning & Design 76](#_Toc151983109)

[Overview of Structure 76](#_Toc151983110)

[Screen Designs & Usability Features 77](#_Toc151983111)

[Algorithm Designs 78](#_Toc151983112)

[Key Variables & data Structures 79](#_Toc151983113)

[Development 80](#_Toc151983114)

[Stage 7 80](#_Toc151983115)

[Stage 8 81](#_Toc151983116)

[Stage 9 82](#_Toc151983117)

[Stakeholder Feedback & Reviews 83](#_Toc151983118)

[Requirements 83](#_Toc151983119)

[Feedback & Summary 83](#_Toc151983120)

[Code Listing 84](#_Toc151983121)

[Post Development Testing 85](#_Toc151983122)

[Evaluation 86](#_Toc151983123)

# Analysis

The Analysis is the first stage of my project. Here I will carry out extensive research to make sure that I fully understand the problem. I will then be able to produce a set of success criteria for my solution.

## Problem Definition

In this stage of the Analysis, I will introduce my proposed application by describing and justifying the features that will solve the problem through computational methods.

### Background to the Problem

#### Description:

* I have a college friend who is a gaming enthusiast and would like me to develop a game for him to play in his free time abroad. Most games he usually plays require an internet connection to play the game at all, with no “offline mode.” For this reason, he would like a game that can be played offline but is also intriguing and fun to play.
* He has specified that he wants it to be a shooting game and have an element of survival. While I do not have a complete idea of exactly what I will be creating, I have a slight vision for a game that fits the genre perfectly, and that also fits with the current trends in gaming. My friend will love this as he is already currently playing the new popular games that are around today.

#### Explanation:

* The major features of the game I plan to implement are:
  + An increasing difficulty algorithm built into the game.
  + A points system
  + Some kind of event that causes the player to lose / “Game Over”.
  + Simple controls
* There will be a player character that the user controls, and they will be avoiding and killing other sprites. Points will be awarded for killing the enemies, and some sort of extra factor will be brought in to make sure that the player keeps moving; depletion of weapon ammunition or depletion of running stamina.
* I will be using Windows Forms with C# on a desktop computer to create the solution. And my chosen Integrated Development Environment will be Visual Studio 2019, as it can fulfil my requirements and is also an IDE, I am familiar with.

### Computational Approaches

#### Abstraction:

* Abstraction is the means of removing any unnecessary detail and keeping parts that are essential. It will simplify my solution, making it easier to develop, and have less requirement for hardware resources. This includes removing the full-detail aspects of a shooting game, such as animations for the entire player’s sprite, like a reloading animation.
* I will create the game in 2 dimensions only, as a 3rd will create unnecessary pressure on both me as a developer to program it, and on the computer to provide the necessary resources. 2D (2 Dimensional) Gameplay will still ensure that the game is still realistically playable, and I will still be able to implement all the things my stakeholder needs.
* On top of that, other features such as external environmental objects like plants and animals will be discounted. In addition, the sprites used for the player and other objects will be simple and not fully detailed (Minute details such as having visible fingers or a fully formed face)

#### Decomposition:

* The development of this game will certainly be complex. Therefore, decomposition is a computational method that is a necessity to be able to construct the algorithms and code to make this project work. Decomposition is used to break down a problem into multiple simpler ones that are much more manageable and approachable. It will split the workload on the developer into easier parts, which will make things much easier.
* Components will be able to be tested individually and can be maintained / edited easily. A way to achieve decomposition is by using a top-down design, this method includes looking at the problem as a whole and identify its main large components. I can then systematically split these components further into smaller ones until the basic building blocks are reached. For example, the player sprite can be split into movement and shooting, movement broken down into updating picture graphics and identifying when a button is pressed, and so on.
* This includes the use of separate components which are built separately and then put together to create a solution. I will be using classes and objects, as well as re-usable sub-routines and procedures to fulfil this. For example, the player’s object will be separate and will have its own functions that can be used, separate from the object for the background or other sprites. The code for movement will be a separate block of code to the code for shooting bullets. Overall, it will mean I can complete the project not only faster, but with better quality and efficiency.
* The use of top-down diagrams and flowcharts for algorithmic decomposition when developing my solution will allow me to identify key points when programming and designing the solution. In my case, I can use decomposition to create flowchart diagrams for individual procedures and methods that I will need to program. It will help me visualise my code and aid me in linking multiple procedures and functions together to create a fully functioning program.

#### Thinking Logically:

* To create the necessary algorithms for my solution, I will need to think logically. This entails organising and structuring an algorithm with a logical manner. Decisions are the heart of all algorithms; it is a point in a program where a different route can be taken depending on the conditions supplied. This makes use of conditional statements such as “if - then” statements and the use of Boolean values.
* Iteration and branches are also used and may be central to the algorithm. Iteration is where loops occur, if a decision leads to a loop, it may mean that new conditions need to be supplied. I will use iteration to continuously check for user input, and branches to call a certain procedure when a condition is met. Conditional statements will be used when the user inputs a movement key e.g., if its left, move the player sprite left. Ultimately, Thinking Logically is a requirement for producing algorithms for my solution efficiently.
* Iteration will be constantly used throughout my program; this is going to be with a timer that constantly loops when the program is ran, it will carry out certain procedures and functions so that the user can use the solution fluently.

#### Thinking Ahead:

* I will need to identify which inputs and data are required for my solution to work. I will also need to establish which outputs and information I need to produce for my objective to be completed. This includes identifying exactly what kind of sprite images to use, and exactly which keyboard inputs will be used by the user (e.g., Up, Left, Down and Right Keys).
* The use of pattern recognition is a crucial tool for computational thinking. It involves the idea that some processes and functions that have already been made / used can be re-used again throughout the solution. C# provides a large library of pre-built functions and classes that I can repeatedly use to solve complex problems without having to completely code them from scratch.
* I will be “Thinking Ahead” whenever I write code so that I can establish whether this will be re-used later in my code, if so, I will create a callable function for it to save time and keep my code bug-free. I will also be importing a range of libraries that will aid me in creating this solution. An example of this is a method that restarts the game, this will be used repeatedly.
* I will import all the resources required for the development of the prototypes at the beginning, rather than searching for novel resources during development, this will make my development more efficient. In addition, I will form classes for each object I plan to create, giving the attributes for each class and whether they hold sub-classes or not. This will prepare me for the actual development of the solution’s code.

## Stakeholders

In this stage of the Analysis, I will determine who will be using my solution that I am creating, and how they will use it.

### Identifying stakeholders

The end-users of this software will be young teenagers or young adults between the ages of 12 and 20. This is because it is a shooting survival game which fits into today’s popular gaming genre widely played by young people. As my product is designed for the gamer and not any other roles, I will identify a persona rather than distinct categories of end-users as this game will only be played by the user and nothing else.

As mentioned, the persona for this game will be young people that are already gamers who are always looking for something new to try. They represent the fast-paced shooting genre of games; these are games that require constant focus and real-time skill. The stakeholders for this game which fit the persona’s demographic are Maimoon Waheed and Hayder Nawaz, who are both close friends that are deeply into gaming. Especially Maimoon, who regularly travels abroad and has no access to the internet, meaning he usually misses his pump of adrenaline from playing fast-paced games like the one I am going to produce. This means he may be a genuine regular user of the game. As stated, the game will have offline capabilities, unlike many popular games today that are online-centred and have no single-player modes that do not require an internet connection.

Hayder is also a fervent gamer who loves reaching new goals / high scores whenever he plays games, my solution will supply this demand with a recorded score system. He also has “professional” experience in gaming, as he has participated in tournaments for games like “Rocket League” and “Counter Strike: Global Offensive.” His experience will be extremely valuable to me as he tries and rates my game, his insight into fast-paced shooting games like “CSGO” will give me the ability to create the perfect product.

## Research the Problem

In this stage of the Analysis, I will research the problem, and then identify and justify a set of suitable approaches for my solution.

### Research of Existing Solutions:

Here I will look at solutions to similar problems to identify and justify some suitable approaches, I will aim to do this alongside my stakeholders.

#### The Binding of Isaac

I need to research for similar solutions to mine that will also have the properties that I will include in my solution. An example is “The Binding of Isaac;” a top-down 2-dimensional game wherein the player controls a character which explores an underground dungeon and must eliminate enemy monsters. There are multiple levels with increasing difficulty, it also includes an inventory system where the character can store multiple weapons for use in combat. There are also “consumables” that can affect health and battle statistics such as power and defence capabilities.

A video game screen with cartoon characters

Description automatically generated

This is like what I have in mind; top-down, increasing difficulties, multiple levels and enemies which need to be eliminated. It also has a story line which is shown throughout the gameplay; however, this is something that was disliked by my stakeholders and interviewees, as it makes boring, slow gameplay and can make the user uninterested. “The Binding of Isaac” is a remarkably successful game, although it’ graphics do not meet the modern capabilities, it makes for simple gameplay and this game shows that high level graphics are not required for an immersive gaming experience.

A white paper with black text on it

Description automatically generatedA screenshot of a video game

Description automatically generated

My favourite graphical feature of this game is the menu which is incredibly unique and different from many other games in the current market. I would like to implement something similar in my solution, it will grab the attention and interest of the user.

Another game like what I have in mind is “Rampage Knights,” a game set in the medieval times where the player controls a character which goes through multiple castles and destroys its enemies. The character is a knight with a sword that becomes stronger with each level. Similarly, to The Binding of Isaac, it has a health system and increasing difficulty level system. The view is not exactly top-down like the Binding of Isaac, the camera view is at a slight angle which makes for more interesting gameplay, this is something that I could implement in my solution.

#### Rampage Knights

A video game screen with a chef and a sword

Description automatically generated

A feature which really captured my attention when researching this game is a “boss level” which occurs as an event every few levels or when a milestone is reached. The bosses are enemies which are much more difficult to eliminate and require much more skill and time to kill. This feature is something that I now plan to use in my solution as I believe it will make the gameplay experience exciting. One thing that I noticed among the gaming community of Rampage Knights was that the game was intensive on computer system requirements, this means that consumers with low-end PCs cannot run the game efficiently. This is due to having multiple extra animations and features such as bleeding and weather. I plan to develop my game in such a way that it can be run on most weaker computers to maximise the number of people who can try the product.

A video game screen with a group of people and a large monster

Description automatically generated with medium confidence

After analysing both games thoroughly, I have extracted features that I can use in my solution such as boss-levels and a detailed menu screen. These games show that high-quality graphics are not necessary for producing a high-quality and popular game. The base mechanics of both games are simple, easy to learn and understand for the user, as the controls are the usual most accessible controls – using arrow keys and another external button for the attacking event to occur. I have also taken from this research that the game *cannot* be repetitive as it will make for boring and uninteresting gameplay. As my game will be a “survival” game there will be a substantial number of repetitive events, but I must try to minimise this as much as possible to keep the player immersed.

A few other things that I have noted which I do not think will be suitable for the solution are the excess of HUD (Heads Up Display); on screen information such as inventory bar and health), like in the second game, Rampage Knights. I prefer the simplicity of a single health bar and two or three important labels that show the user the essential information. This is because I do not like a cluttered screen, and I think such a type of design will not be appealing for the user. Another is the excess of blood graphics in Rampage Knights, every enemy killed releases blood onto the floor which stays there for a noticeable amount of time. With multiple enemies it makes the background cluttered and ugly.

The last problem I faced from the research was the lack of direction from The Binding of Isaac, at some times, the aim of the game is completely unknown. There are an endless number of doors to go through in the game and there is not much explanation from the story cut scenes. There are also few signs for the user or hints to help them progress in the game. This does allow for user exploration and adventure, but this does not fit with me clients’ expectations and wants.

### Stakeholder Feedback:

I will be interviewing my two stakeholders, Maimoon and Hayder. They have different approaches to gaming, which is why this interview will be important, as it will help me understand the specifics of their needs, so that I can cater for them both with a balanced gaming experience. I will also be interviewing a *third* person, who will not be a stakeholder but will help give me an insight into what the perfect game should be like as they have an experience in developing games in the past. The interviewee’s name is Phillip Wright, a previous game developer for companies such as Ubisoft and Electronic Arts. They will aid me in creating a suitable solution for today’s gaming market.

#### Questions:

1. What is your current gaming set up like?
2. Which kinds of games do you prefer, casual or fast pace?
3. When you begin playing a new game do you prefer having easy-to-learn controls that you can utilise quickly, or a depth of multiple controls that require time and experience to master?
4. And what kinds of controls have you used in the past?
5. Do you prefer constant action gameplay or gameplay with small breaks with a story line?
6. Are high quality graphics a requirement for a good game in your view?
7. Which features are essential for you in a game menu?
8. (For Phillip) When developing a shooter game, which properties for gameplay do you think are necessary to include?
9. Do you have anything else to add?

#### Question backgrounds:

Question 1 and 2 are background questions for the interviewee, these are necessary for me to formulate an idea of what their general opinions are for gaming.

Question 3 is useful for me to decide on what kind of controls I will implement in the game so that they are not too difficult to use but also interesting enough. Question 4 also investigates their history with controls in other games.

Question 5 gives me an insight into their favourable gameplay so that I can create a satisfactory solution. The feedback from this question will heavily influence how I formulate the gameplay of the game, slow gameplay, or fast action.

Question 6 will inform me whether I need to make an update on the sprite’s graphics in-game so that they are higher quality and more interesting. But it will also give me an insight on the interviewee’s opinions on whether a game needs good graphics or if they just want good gameplay.

The menu is an important feature of the game that needs to be easily accessible and have all the required controls for the user, this is investigated in question 7 and will help me create a solid useful menu for the clients.

This question is aimed for the developer Philip only, his experienced insight into game development is invaluable and I need the core foundational features that he uses when creating a survival shooter game. The feedback on question 8 will help me build a solid foundation of the game with all the properties I need.

Question 9 is an additional question to ensure that I have made the most of the interviews and so that I extract all the thoughts and ideas from them, as every little piece of feedback will be especially useful. If I have missed any important questions out, this will fulfil them.

#### Interview Analysis:

Philip does not playgames, he only develops and tests them, so his answers for some of the background questions were null and not useful. Each bullet point explores the feedback given by the interviewees, in order. Hayder and Maimoon were interviewed in person, while Philip was spoken to over an online seminar / meeting. After completing these interviews, I have extracted the following feedback:

* Hayder has an advanced gaming set up; this is because he currently plays high end games. While Maimoon has a weaker computer with low-end system specifications, this means that I need to ensure that my game can run smoothly. I will test this using an old PC at home with similar specifications that of Maimoon’s.
* Hayder is a fast-paced game enthusiast, he loves playing high intensity games that require constant focus on playing the game. Maimoon has a similar attitude to this but would still like to see some kind of small story line built into the game.
* Philip explained to me that the controls for the game depend on the type of game, since this is a top-down 2D game the controls cannot be too complex. He also stated that some sort of inventory cycling control should be implemented If I do create an inventory system in the solution. Hayder prefers complex controls that require time to learn as he believes that they will bring depth to the game. While Maimoon prefers controls that he can pick up instantly and are easy to use.
* Both Hayder and Maimoon have used controllers for gaming consoles such as PlayStation and Xbox. They have also used keyboard controls and prefer keyboard for shooting games.
* This question explores similar ideas to question 2, but I also brought from it that the gameplay cannot be constantly repetitive. I got this feedback from Philip who says he has had experience with consumers becoming uninterested with a game because of its repetitiveness.
* Philip believes that a game of this level of development (2-dimensional game) does not require serious thought on graphics at all, it just needs to be clean looking and have some intricate details. Hayder on the other hand says that graphics are the first thing he looks at when experiencing a new game, as he personally believes that it means the game is “good.” However, he does understand that a game of this level is not really in need of high-quality graphics. Maimoon also thinks similarly to Philip.
* Maimoon says that the menu needs to be a base to access all parts of the game from one screen and says he does not like having multiple pages within menus. Hayder says that there should be a pausing menu function this is something that I should implement. Philip says a game menu only really needs to fulfil the functions available and does not need to be in depth and show everything. He also says the graphical design of the menu needs to be quite simple.
* This question was directed at Philip, and he has given me some in depth feedback. He states first that there needs to be some sort of milestone and score system as it will keep the player intrigued and wanting to play more. He also says there needs to be a limiting factor in the game – something that will cause the player to try to survive harder or they will lose. In addition to this he says the gameplay needs to be smooth and transitions between levels should not be instant / automatic and that the user should be able to choose to move on.
* Maimoon also added that it could be useful to make the game controller compatible as he is likely to be playing this game abroad as stated in the problem identification and stakeholders’ sections. It is easier for him to carry a controller rather than an entire keyboard. But I did point out to him that a laptop can fulfil this.

### Essential Features

My application will have many features that have been touched on in the interview stage and from the game research stage. One feature is a smart, clean menu system, this is needed for the user to navigate the game and be able to access essential functions such as seeing their high scores, exiting the game, restarting a level, accessing other levels, and editing game settings such as sound volume.

I will also implement a multi-level milestone score system, when certain milestones are reached, the player needs to fight harder bosses. It will keep the user interested in the game. I will also try to create an inventory system within the solution which means I will need to create an array of different weapons for the player character to use.

The controls will be simple keyboard controls – arrow keys for movement, one or two buttons for the attacking event, and a way to scroll/cycle through the inventory. I will also try to implement a “tutorial level” at the beginning of the game to teach the user the aim of the game, how to play and the controls.

There will be a “limiting factor,” which will cause the player to become under more stress leading to them losing the game. This will be done by implementing a health system for the player’s character, it decreases when attacked. Another factor could be the speed of the game, which can be simulated by increasing the enemy speed.

### Limitations

This will be a PC game and will not be compatible with anything other than a keyboard, this is due to the solution being programmed on a computer using C#, a windows .NET programming code which is not suitable for cross-play or for running on different devices. This will not be a problem though as my clients are both PC players. There will not be an in-depth array for the inventory system, it will not affect the solution very much as the user will still be able to experience different things from various levels.

The graphics will be simplistic and 2 dimensional, this is because I will not be using a high-spec graphic renderer to create a 3D high-quality graphics game. It is also due to the development engine I will be using, Visual Studio. As I will be programming using windows forms on visual studio, 3D graphics is unachievable. Not all animations will be rendered, as mentioned at the beginning of the analysis, animations such as reloading, having the character able to face every direction and the movement of feet will not be included. This is because it will mean that I will need an exceptionally substantial number of images for each frame, this leads to the need of more rendering and more code. This eventually leads to a slower processing time for the solution as well as using more storage space.

Combat with the enemies in the game will not be fully simulated, for example you will only be able to shoot the enemies rather than use hand-to-hand combat. Similarly, to the previous point about not all animations being rendered, this is because it requires more code and unnecessary work. In addition, the background story that I plan to implement will not be in full detail and will not show every single part of the story. This is due to the requirements and expectations of the clients, they do not want a long story that they need to follow, they want a fast-paced game that they can jump straight into.

## Proposed Solution

This final section of the Analysis is a product of all my research. I now have a good idea of what my proposed solution will be like. Therefore, I can create a success criterion, and also set out my likely requirements for my solution.

### Hardware and Software Requirements

The game will require certain levels of hardware and software to be able to run it. However, as the game will be developed using Visual Studio 2019 Community Edition, the requirements will be based on the requirements for Visual Studio rather than the game by itself.

|  |  |
| --- | --- |
| Requirements | Justification |
| Software |  |
| Operating system: WINDOWS 7 / 8 / 10 / 11 | Visual Studio is a Windows based IDE. However, it can be run using virtualisation on MACOS or Linux |
| Visual Studio 2019 | This is required to be able to run the game using the code |
| Visual Basic runtime libraries | These libraries are required to run certain aspects of the game, these will be automatically installed |
| .NET framework 4.5.2 or above | The .NET framework is a requirement to compile the code, this may also be automatically installed |
| Hardware |  |
| Processor: 1.8Ghz+ | The graphics and animations of the game require a processor with a clock speed of at least 1.8Ghz |
| Memory: 2GB+ | The tasks being ran will require a substantial amount of memory to work |
| Storage: Minimum 1GB | The game itself will take up significantly less storage space, however the IDE will require a substantial amount |
| Video card supporting minimum of 720p | The graphics made in visual studio are rendered in 720p |
| Computer Keyboard | Needed to control the game and character |
| Monitor / Screen | A way to see the game |
| Speaker | There will be in-game sounds such as shooting sounds and in-game music |
| Mouse | Required to navigate the menu in-game |

### Success Criteria

Here I will identify all my objectives for the proposed solution, they will cover all the expected functionalities of my system.

|  |  |  |
| --- | --- | --- |
| Requirement | Justification | Reference |
| The game will have a top-down, 2-dimensional view | This is the most preferable dimension and view for a player, and it is simple to implement | Proposed solution |
| There will be multiple enemies with a range of difficulties | The game needs an element that keeps the user interested, the enemies will become harder in difficulty, and some will be extremely hard | Proposed solution |
| There will be an ammunition system for the guns | This is a limiting factor that will keep the player “on their toes,” the ammunition will go down by 1 after every shot | Proposed solution |
| There will be a health bar system for the player | This is another limiting factor that will make the user change how they play. The health of the player will decrease when an enemy hit them | Proposed solution |
| There will be a score system based on eliminations | This is to keep the game competitive and make the user feel a sense of achievement if they reach a high score | Proposed solution |
| A menu system for navigation between high score tables, pausing, restarting the game etc. | This is purely navigational and is there for the user to be able to access various parts of the game and complete system functions such as exiting | Proposed solution |
| Keyboard controls – Up left down right and spacebar for shooting | This is so that the user can easily control their character and kill the enemies. | Proposed solution |
| A multi-level system | This is so that the user stays interested and keeps the game action filled. | Proposed Solution |
| Multiple weapons system | Gives the user a large variety of options to eliminate enemies | Proposed Solution |
| Inventory system | Allows easy access for the user to navigate between weapons | Proposed Solution |
| Easy to call procedures | Procedures will be named appropriately so they can be used throughout the program for quick and efficient code | Development |
| Flexible code | Code will be open to additions so that it can be improved easily, and new prototypes can be developed from the previous structure | Development |
| Campaign game mode | This gives the user a choice, and reflects a different type of gameplay; gradual increase in difficulty and a story line to follow | Proposed solution |
| Survival game mode | Gives the user a choice to try a different type of gameplay; fast paced gameplay where the aim is to eliminate as many enemies as possible | Proposed solution |
| Design features for user usability such as window maximisation | Allows user to navigate the game with ease, also makes everything look clean and appealing. | Development / proposed solution |
| User login / authentication system | This allows data for each user to be saved, such as their high score and which level they have reached | Development / proposed solution |
| Smooth graphics | The movement of sprites in the game must not be glitchy, for a smooth user experience | Proposed solution |
| A high score system for each user | Allows users to compete with each other / aim for higher targets, will empower them to want to play the game more | Proposed solution |
| Sounds | Introduces extra immersion into the game for the user | Proposed Solution |

# Design – Decomposition

Before I begin planning how I will complete my project, I will need to have a good idea of what is required for my entire solution and how to break it down into sub-tasks.

## Top-Down design

I have used decomposition to break down my program into a set of major areas: Character movement, Progression, Character actions, Graphics, Menu and Sounds.

## Explanation of Processes

### Character Movement:

Keys will be assigned for each direction (arrow keys) for up, down, left, and right. The program will be able to detect when these keys are clicked, record the direction that corresponds to that key, and update the player graphics so that it faces in the direction and moves in that direction.

### Progression:

Two modes – Campaign and Survival. Campaign will entail multiple levels with boss levels every few levels, enemies will get harder and harder to eliminate with each level. This can be done by increasing damage from enemies, increasing their movement speed, and increasing their “health” – the number of hits from the player will increase for them to die. A points / score counter will increment with each kill, and when several kills is reached the new level will begin.

Survival will entail constant waves of enemies that will become harder to kill over time. The score will be recorded and will contribute to the high score system.

### Character Actions:

The character can kill enemies by shooting them with their weapon, a bullet image will be created and will move in the direction that the character was facing. Collision detection will be able to see when the bullet has hit an enemy. There will be health pickups, ammo pickups and new weapons to choose. The events will occur when collision is detected between the character and the pickup. Health will increase the health by some factor, similarly ammo will increase the ammunition count by some factor. New gun pickups will provide the player with new weapons with different abilities.

### Graphics:

Enemies and characters will have their own images that can collide with other objects. These will be updated as the game timer ticks and movements occur. When the player dies, their character will assume a new image of a dead character. The bullet from shooting will need to disappear and be deleted when it reaches the end of the screen or hits an enemy. In addition, there will be bounds for the character and enemies so that they cannot leave the screen.

### Sounds:

There will be a sound played when the character dies, when zombies die, and gunshot sounds. I will also include menu music to add some interest into the game. A sound that is different to the others will be used when boss levels are reached to notify the user that they have reached a boss level.

### Menu:

The menu will be accessible upon start of the program and when the player dies. It will also be accessible through a “pause” function that the user can initiate to pause the game and access the menu. Within this menu you can exit the game, resume your game, restart the level, and check previous scores and your high score.

## Justification

I have chosen to use a top-down design to represent the decomposition of my program because of its ability to aid me in viewing the entire programme in a straightforward way, as well as be able to identify the smaller sub-components. It also makes the program more manageable as I can focus on one area at a time.

By using this method, I have been able to identify all the required sub-tasks that are needed to fulfil each area of the program. I now have a clear idea of which algorithms I need to create / use so that I can complete each task. It also allows me to “Think Ahead,” I can realise the inputs and outputs required for each sub-task. I have also been able to identify decisions needed in the program and what their affects are on the game by use of “Thinking Logically.”

## Planning Ahead

Explanation of my approach to design and development of my solution.

### Iterative Development Approach

I will be undertaking an iterative approach to the development of my solution. This includes creating parts of the solution at a time and forming prototypes that build on each other as time continues. Each prototype / version will fulfill an area of the game and will be designed in a way so that it can be built on easily for the next prototype.

### Stages

* Stage 1 – Creating the first game screen with player image, health bar, score count and ammo count.
* Stage 2 – Gameplay; Player character movement, adding enemies to the game that follow the player and damage them (health bar decreases). Shooting to kill enemies. Death when zero. Ammo drops.
* Stage 3 – Simple menu, appears after death and beginning of game, exit and restart.
* Stage 4 - Introduction of progression; First level created. Increasing difficulty of zombies. An introductory level
* Stage 5 – Health pickups. Game sounds (shooting, death sounds etc.)
* Stage 6 – Menu development; access to Campaign mode and Survival mode. Access the menu through a pause button while playing.
* Stage 7 – Boss levels, New Gun additions
* Stage 8 – Previous scores accessible from Menu + High score
* Stage 9 - User management system

Stages 1 – 3:

This will be the first version, mainly because it is the base functions of the game that are required first so that the rest of the game can be developed.

Stages 4 – 6:

Version two – This will introduce extra features that will be added to bring interest to the game. Extra features may be added within this prototype version that have not been stated, these will be small improvements that are thought of during development.

Stages 7 – 8:

Version three – These three stages bring a lot of complex features, and they will take a lot of effort. At the end of this final prototype, I believe that it will be as good as a game as other games on the market today.

Each version will have a plan and explanation of the development of the prototype. With each addition of a feature or code, a test will be carried out and presented. After each version is made, it will be tested by myself, the stakeholders, and external testers. This is to ensure that it works smoothly, and it provides the effect that I want it to produce. I will then evaluate the entirety of the prototype and signify which areas have been improved sufficiently and which areas need further development. This Iterative development cycle will be repeated with each prototype version and will overall improve the efficiency of the development of my solution.

# Prototype 1

## Planning and Design

In this section I will present my clear idea of how I will approach the problem. My initial ideas may be subject to change as I develop.

### Overview of Structure

I will use a class diagram to represent the different classes and objects in the solution. It is an ideal way to show how all the distinct parts work together, it will also indicate the attributes and methods that each part of the game uses. This is all for the first version of the solution.

A screenshot of a game

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screenshot of a menu

Description automatically generated

### Screen Designs & Usability Features

In this section I will be providing images of the forms I have created for the first prototype and will be describing, explaining, and justifying each part of every screen. Usability features of each form will also be pointed out and justified. This section will aid me in realising how user-friendly my forms will be and will also help me understand the processes that a user will go through when making use of this product.

In general, the aesthetics of the game will be simple, with reasonable font size so that it is easily readable and a simple font type so that it keeps its simplicity and ease of reading. I will incorporate the shooting survival theme colours throughout the game, however, will still make effort to retain cleanness and simplicity throughout, especially in the menu where there will be a lot of text and reading compared to the gameplay forms. I will go through each form in the solution, giving a description of the functions, explanation of each process and why each function is presented in the way it is.

#### Menu

My aim for the menu is to keep it simple. Easy to read and navigate, however I am still retaining an element of zombie survival theme so that it remains relatable to the game itself. Currently in the first prototype the menu is limited to Starting a game, exiting the game, and reading the previous score. The menu is also only accessible when the game is first opened and at game over. A screenshot of a video game

Description automatically generated

The Start and Exit buttons are simple buttons with events assigned to them when they are clicked. This form also links to the main gameplay form so it can display the most recent score. The menu background is a screenshot of the gameplay and gives the user an insight into what the game looks like

#### Game

For the first version, the gameplay screen will be quite simple, the main aim of the first prototype is to have a functioning game before thinking about aesthetics. The main functions are shown on the screen such as health bar, ammo count, kill count and a small label with an indication that you can click ESC to exit the game.

I have decided to use a progress bar for the health bar, this is because it accurately represents the health remaining of the player, in a visual way. I also added a label with “Health:” next to it so that the user understands what the bar is for. The ammo count and Kills count are represented as labels because they are editable during gameplay through the code, this is so that they can be updated when an enemy is killed and when ammo is used / picked up. They are also quite large, so they are visible enough for the user. An extra small info label is in the top right corner of the screen, this is a label telling the user that they can exit the game by clicking ESC.

A screenshot of a video game

Description automatically generated

The current background is simple, dark grey. This will be changed and improved to give a more immersive feeling for the player, however for now it is suitable to have the sprites on top of it so they can be seen and is also a good background for the white labels and progress health bar.

#### Gameplay

Below I will show the game screens when the game is ran, and the steps which players will go through, from when they start, to when they die.

Step 1:

A screenshot of a computer

Description automatically generated

This is an image of the first screen the player is met with once they click start. The game waits for the user to input the enter key on their keyboard before the game begins. I have made sure that the game sprites have clean edges and an invisible background, when they move on the screen It is smooth because of this. The sprites are just picture boxes that are added to the screen and their location is updated to simulate movement.

Step 2: A screenshot of a computer

Description automatically generated

When the user clicks Enter, the game timer is started, and the enemies begin chasing the player sprite. The player then uses the arrow keys to avoid the enemies, and the spacebar key to shoot at the enemies. The image above shows a bullet, and the enemies chasing the player.

Step 3:

A screenshot of a computer

Description automatically generated

When the enemies come in contact with the player sprite, the health bar on the top of the screen begins to decrease in value, as shown in the image above. Also, when the player has used all their bullets, the ammo label shows 0, and an ammo “crate” is added to the screen which the player can come in contact with, in order to replenish their ammunition count by 5, as shown in the image above.

Step 4:

A screen shot of a video game

Description automatically generated

When the player’s health decreases to zero, they die. The main game window closes, and the menu form opens, where the user can see their score, restart, or exit the game. Upon clicking the “Start” button, the game will display the main game form to the user and start from step 1 again.

#### Features

All the information required is on the same screen, so the user does not need to access a menu to use the simple features currently available to them. The menu is also simple and does not give the user too many options so that they do not become confused. Currently the number of functions in the game is low, as more updates are added, there will be more usability features that I need to incorporate so that the user can play the game easily. The menu shows before the game play begins when first opened, to allow the user some time to get ready and click start.

### Algorithm Designs

After having identified an overall structure, I will now give an outline of my algorithms required for each game feature. I will be using flowcharts to present my procedures, functions, and subroutines.

#### Key Events for Movement

A diagram of a game

Description automatically generated

The KeyIsDown() event handler detects when a key is pressed by the user. This algorithm shows how it processes certain key inputs for movement (Up, Down, Left and Right). When each key is pressed, its corresponding attribute for movement is set to the direction it represents. The variables for movement are goRight, goLeft, goDown, goUp, facing and the Picture box image for the player’s character is changed with each input to update the graphics so that the sprite is facing and moving in the correct direction.

A diagram of a flowchart

Description automatically generated

The function of the KeyIsUp() event handler is to ensure that the player’s character / sprite stops moving when the user stops holding down an arrow key for movement. When each arrow key Is let go, its corresponding movement variable is set to false. This will cause the sprite to stop moving on the screen. The variables are the same as that for KeyIsDown() – goRight, goLeft, goUp and goDown. With both KeyIsDown and KeyIsUp working together. I can now make character movement smooth and efficient.

#### Game Timer

The game will have a timer that ticks every few milliseconds, this is to update graphics and the score. I have done this to make graphics updating clean and smooth. Its name is gameLoopEvent().

A diagram of a game

Description automatically generated

First, the ammo label is updated with the remaining ammunition of the player. Then the Score / Kills label is updated to show the number of kills the player has. The first 4 decisions are checking if the user is trying to move their character and if they are at the boundaries yet. It also identifies which direction of movement, and then updates the sprites graphics correspondingly. It then checks if the player’s health is below 1 (0), if it is it will go through the series of processes to initiate the death and the end of game such as updating the sprite graphic to the dead image and initiating game Over. If not, the health bar graphic will just be updated accordingly.

#### Bullets

The bullets have their own class (Bullet). There are multiple methods for this class that can be used together to create a bullet and make it move.

A diagram of a make bullet

Description automatically generated

The MakeBullet() method uses the form as a parameter (make Bullet(Form form)), to add the bullet image picture box to the screen. It sets the properties of the bullet such as size, colour, and movement speed. Then it adds it to the form visually, sets up its timer and clock and starts it. A timer has been used to ensure smooth movement of the bullet image across the screen. This only contains processes as this method will be called when a decision has already been made.

A diagram of a bullet

Description automatically generated

This method “BulletTimerEvent()” controls the timer of the bullets. With every tick the following processes are carried out. It checks the direction of the bullet and begins moving the bullet in that direction using the bullet speed. This gives a smooth movement of the bullet on the screen. The method then checks if the bullet is on the edge of the screen by comparing its position on the screen to the size of the display. If it is, it is timer will be stopped and disposed so that it stops moving, then the bullet image will be removed and deleted from memory so that it is not seen by the player. It is completely removed from memory to avoid any duplication errors. This event occurs every few milliseconds, which means that the movement of bullets on the screen is exceptionally smooth, and the location of the bullet is checked constantly.

A diagram of a flowchart

Description automatically generated

This method ShootBullet() takes in the direction of the player as a parameter, it uses this to set the direction of the bullet. It first creates a new instance of the bullet using the Bullet Class, its coordinates on the screen are set in a way that the Bullet starts near the player’s character; it does this by using the sprite’s coordinates and its width to spawn the bullet In front of the gun. There are no decisions made in this method because it will only be called when the shooting event Is triggered. The MakeBullet() method is then used to add this bullet to the users display.

#### Dropping Ammunition

When the player’s ammo count is low, an ammo supply box will be dropped on the screen for the player to pick up and refill their ammo supply. This method will be called when the ammo is less than 1 in the KeyIsUp() event and will be checked when shooting a bullet (space bar is pressed). The supply is a picture box with an image and its location will be completely random on the screen.

A diagram of a diagram

Description automatically generated

First a new picture box instance is created, the ammo image is then added to this from resources. The sizing mode is set to auto size, this is because the image in resources is already the correct size for the game and this property will change the picture box dimensions to the correct size automatically. The x and y coordinates of the picture box will set the location of the image on the screen, they are set in a way so that they remain within the bounds of the screen. The picture box is then given a tag, “ammo,” this is so that it can be easily identified when removing the image from the screen. After adding the picture box to the screen, the ammo box will be brought to the front so that it can be seen, then the player picture box will be brought to the front so that the player sprite can walk over the ammo box.

#### Collisions

The game will automatically detect when the player’s sprite collides with a zombie or an ammo box, it will also detect when a bullet collides with a zombie sprite. There is an in-built function in C# from the imported libraries that can be used to detect intersection between picture boxes. It is used like this: “zombie.Bounds.IntersectsWith(player.Bounds)….” This is use of modular programming and reusable components; it comes from imported libraries and will save me time in development and is also already bug-free. The method for it is called CollisionDetect(), and it will be called in the GameLoopEvent event, meaning collision will be constantly checked while playing the game.

When this method is called, depending on which objects have collided, some processes will be made. When a zombie collides with a player, the player’s health will be decreased by 1 each time, when the zombie is constantly touching the player’s sprite the health will constantly decrease by one. Since this method is called from the game’s timer GameLoopEvent, the health will decrease every few milliseconds by one which means it will decrease in a uniform way. When an ammo picture box interacts with the player sprite, the ammo count will be incremented, and the ammo picture box will be removed from the form. I have used labels for each picture box such as “zombie” and “ammo.” This is done to allow me to search through the controls on the form and find the sprite / image I want to use. In addition, a bullet can be detected to collide with a zombie, when it does the kill score is increased by one and the zombie sprite is removed from the form. It will then initiate MakeZombies(), this method will spawn a zombie on the screen to make up for the killed one. This method is crucial to the project because it is the thing that handles killing enemies, refilling ammo, and getting attacked by enemies.

A diagram of a flowchart

Description automatically generated

#### Creating Enemies

The game needs to be able to create zombie picture boxes, assign the zombie image to it from resources and add it to the form. This will be in its own method that will be able to be called from anywhere in the program. It will be crucial to keeping the game moving and the user interested in the game.

First a new picture box is created, and it is assigned the tag “zombie” so that it is identifiable when searching through controls on the form, such as in CollisionDetect(). Then the zombie image is taken from resources and assigned to the picture box. The default image for the zombie when spawning is the facing down image “zdown.” Then the location of the picture box is set to a random location on the screen, this is done by using the rnd.Next function. This is a function from the randomizer library on Visual Studio. I will set the minimum and maximum values for the X and Y values of the picture box, and the function will return a random number within this limit. I have set the min and max values in a way such that the location will be within the screen limits. After the location is set, the size mode will be set to “Auto Size,” which is a setting for picture boxes on visual studio and it will make the picture box hold the zombie image in its natural form. After all the properties have been set for the picture box, it will be added to the form, and it will show up on the screen. To ensure that the zombie images do not overlap the player’s sprite image, the player’s image will be brought to the front using a C# OOP command called BringToFront().

A diagram of a zombie

Description automatically generated

This method is crucial for the solution as it is the procedure that will create the zombies and add them to the screen for the player to eliminate. It can be called from anywhere in the program.

#### Starting the Game

This method’s purpose is to initiate the start of the game so that the player can begin moving and eliminating enemies, and the zombies can begin chasing the player. It will be called when the user presses play on the menu.

A diagram of a game

Description automatically generated

Then the game Over variable is set to false so that the knows that the game is about to start again. After that the ammo and score count are set to their original values, 10 and 0, respectively. These processes are done to ensure that all values are reset when the game is starting again.

Then the player image is set to the starting image of it facing upwards, the image is pulled from resources and applied to the picture box. Then all the zombies remaining on the screen are removed by going through the zombies list and removing the picture boxes assigned to each zombie with the Remove() function. Then the zombie list is cleared completely so that there are no instances of zombies in the game at all.

The game now needs new zombies for the player to eliminate, it does this by calling the MakeZombies() procedure. It does this three times to have 3 randomly spawned zombies on the screen for the player to kill. The player needs to be still as the user is not clicking anything anymore, when the game ends from death, the direction of the player sprite may remain. So, the direction variables goup, godown, goleft and goright are all set to false.

This method is needed for the game because of Its importance when first starting the game, so it becomes easy to begin the timer, begin movement on the screen, and allow user input.

#### Enemies chasing player

I will use a method called ChasePlayer() which will be a procedure callable from anywhere in the program. Its purpose is to make the zombie sprites move towards the player sprite so that they can “attack” the player. It is a big function in the project and the game will not work without it. It will be called in GameLoopEvent, this is so that the zombies continuously chase the player, because the game uses a timer with a tick that occurs every few milliseconds. This will lead to smooth, constant movement.

A diagram of a diagram

Description automatically generated

This procedure has the most decisions, these decisions are important in knowing which direction to set the zombie’s movement to. As the procedure is called from the game timer GameLoopEvent, it will produce smooth movement of the zombies towards the player sprite. First the code will find the zombie picture boxes by searching through every control in the form with the criteria of being a picture box and having the “zombie” tag. Then for each zombie picture box it will check its location on the screen, and see whether it needs to be moved up, down, left, or right to get closer to the player sprite. It will then move the zombie in this direction using the zombie Speed, it does this by editing the coordinate on the screen so that it is moved however many pixels in a direction. The program will also change the image so that the zombie faces the direction it is moving in, it pulls the required image from resources and assigns it to the picture box.

This method means that zombie movement is automatic and does not require user input, neither do the zombies need their own movement variables such as goup godown etc. Like the player sprite needs.

#### Summary of Algorithms

All these algorithms act as representations of all the methods and procedures in the first prototype. They will all be coded and used throughout the development of the program. In upcoming prototypes, there will be many more new methods and procedures. These will also be illustrated, explained, and justified as I have done with this prototype

### Key Variables & Data Structures

In this section I will be presenting my key variables and data structures that I have identified during the formation of my solution’s algorithms. Since this is a game, there will not be a need for much data handling apart from when recording scores in future prototype versions. There also will not be many entities in this solution due to it being a game.

#### Entities

I have created an entity relationship diagram that illustrates the relationship between the different areas of my solution.

A diagram of a zombie

Description automatically generated

For the first prototype there are currently only two identifiable entities in the solution, the player, and the enemies. This is a one-to-many relationship, with there being one player with an aim to eliminate multiple enemies. I have listed the attributes for each entity, these will be explained and justified in detail in a later part within this section. The lack of entities, as well as lack of key attributes, is due to the nature of this solution; it is a game, and this game focuses on a player and their goal, eliminating enemies. There is not a wide berth of different entities within the game that can be linked to each other, however, in further prototypes there will be more, such as levels within the game.

#### Databases

The only database that exists in the first prototype is the Resources folder, which comes built into the Visual Studio development environment. I can store the graphics needed to run the game here.

|  |  |
| --- | --- |
| Image name: | Use: |
| up | Player sprite facing up image, starting position, used when player is moving up |
| right | Player sprite facing right image, used when player is moving to the right |
| left | Player sprite facing left image, used when player is moving to the left |
| down | Player sprite facing down image, used when player is moving down. |
| zup | Zombie sprite facing up image, used for when the zombie is moving upwards |
| zdown | Zombie sprite facing down image, used when the zombie is moving downwards |
| zright | Zombie sprite facing right image, used when the zombie is moving towards the right |
| zleft | Zombie sprite facing left image, used when the zombie is moving towards the left |
| menuBackground | Assigned to the background of the menu form. It is the menu background |
| dead | Player dead sprite, this is applied to the player’s picture box upon death |

#### Classes

Currently there are only three classes; a partial class for the form called frmGame, a partial class for the menu form called frmMenu and a Bullet class. The Bullet class is used for the creation and movement of bullets in the game, and it has its own methods and procedures which are called from the main program. The main form class is representative of the main program, and holds the attributes, methods and procedures required for the gameplay in general. The menu form class represents the main menu form, and it controls what happens when the user clicks buttons on the screen.

A screenshot of a computer program

Description automatically generated

Each procedure and method for each class or partial class will be explained and justified in the next part. These class diagrams are a visual representation of what each class holds, and their attributes.

#### Key Variables

I will go through the main variables for each class, this is to show the use of each variable and their justification. I will also show which subroutines and methods use each variable.

##### Game form

|  |  |  |
| --- | --- | --- |
| Variable: | Subroutine | Purpose & Justification |
| gameOver  Boolean | frmGameLoad()  keyIsDown()  keyIsUp()  RestartGame()  GameLoopEvent() | The gameOver variable is an important flag and identifier used throughout the solution. When set to true the game will understand that the player has died and that movement needs to be stopped, as well as other things. When set to false, the game knows that the player is alive, and the game can continue. It is checked with every tick of the game timer.  A flag like this is crucial for this game, as it controls whether the gameplay continues or stops. |
| facing  string | keyIsDown()  keyIsUp() | This variable holds the direction that the player sprite is moving in and facing. It is crucial for player movement and movement will not work properly without it. |
| speed & zombieSpeed  int | gameLoopEvent()  chasePlayer() | The player speed is faster than the zombie speed so that the player can run away from the enemies. They are used to move the player and zombie sprites on the screen, it does this by incrementing the location of the sprite on the screen by the speed with every tick. It is also a vital attribute for movement and graphics |
| zombiesList  List<PictureBox> | makeZombies()  RestartGame() | This list holds picture boxes only which will be used for the enemy’s sprites on the screen. I have used a list so that multiple zombies can be added at the same time. It also allows for simple removal of sprites upon death. The list is cleared upon restarting the game and new zombies are added to it when making zombies. |

##### Bullet

|  |  |  |
| --- | --- | --- |
| Variable: | Subroutine | Purpose & Justification |
| direction  string | shootBullet()  Bullet() – constructor  bulletTimerEvent() | This variable holds the direction of each bullet, this is an identifier for the program to know which direction the bullet should be travelling in. It is set to a certain direction (“up”, “left” etc…) when the bullets are created using the constructor. It is a useful variable because it is important for the bullet’s movement on the screen. |
| speed  int | makeBullet()  bulletTimerEvent() | The graphics of the bullets are moved across the screen by the speed value. It does this by using the location of the picture box and incrementing it by the speed. It is also assigned to the timer so that when the timer ticks each time it will move the bullet by this small fraction. This means that the graphics are smooth for the bullets, and they move uniformly. |
| bulletTimer  Timer | makeBullet()  bulletTimerEvent() | This timer is used when creating each bullet and it ticks every few milliseconds and it is used to move the bullet on the screen. The bullet will not move on the screen without it. |

##### Menu form

|  |  |  |
| --- | --- | --- |
| Variable: | Subroutine: | Purpose & Justification |
| labelText  string | frmMenu() – constructor  frmMenuLoad() | This is used to hold the score of the previous game so that it can be displayed on the menu after the player dies. It is used as a parameter when loading the menu form from the main game form. The score can be seen by the user when they die on the menu because of this variable. |

### Validation Methods

There are not any significant validation methods in the first prototype of this solution. This is because most functions are straightforward and there is usually a true or false outcome of each function. In addition, there is not much data handling required due to the lack of a database in this version, therefore there are not validation checks such as a Lookup check. The following table will go through the few validations checks that are used in my solution.

|  |  |  |
| --- | --- | --- |
| Validation Check: | Description: | Example: |
| Type check | Checks the type of a variable so that it can be used within a certain function or method. Data types include integers (int), Booleans (bool) and many more. | Checking if a control is a picturebox: |
| Collision check (method) | This is presented through the CollisionDetect() method, however there is a main function used within the method called IntersectsWith() which is a Microsoft forms built in function that I am using. | Checking if sprites are colliding |
| Range check | Checks if a variable’s value is within a given set of values. In my case it is used with double (data type) numbers. | Checking if the health is above one, so that the program can identify whether the player is dead or not. |
| Location check | This is used throughout the solution and is crucial for producing and updating graphics on the screen. It is a check that also involves a range check to identify the x and y values of the image’s location on the screen. | Checking if bullet has reached the edge of the screen through it’s location |
| Direction check | Used in multiple methods throughout the solution to identify the direction that a bullet, player or zombie needs to move in. Takes in variables that are made specifically for setting directions for each entity. | A white background with black and green text  Description automatically generated  Checking goLeft, returns true or false. If true it will check the location and then move the picturebox accordingly. |
| User Input check | Through the use of built in functions such as Key is down / up events, I can check the user inputs and then act accordingly. The key pressed by the user is taken in by the software and then the code I have written will complete processes to make things occur on the screen | Checking I the key entered is the left arrow key. |

I will also need to describe how all the inputs and variables will go through validation checks in my solution. The following table shows the input value and which validation checks it uses, as well as describing and justifying the process for each variable.

|  |  |
| --- | --- |
| Input Variable & Validation Method: | Description & Justification: |
| Goup, godown, goleft, goright  Direction Check | These Boolean variables hold either a true or false value. When each variable Is true, the direction check will inform the program that the picture box in question needs to be moved in that direction. A direction check on these variables is crucial to identifying which direction sprites need to be moved in. |
| playerHealth, ammo  Range check | A range check is completed on the player health with every tick of the game timer, this is to identify whether the player is dead or not. The ammo is also checked with every tick and is useful for the program to know whether a new ammo supply box needs to be spawned onto the screen for the user. |
| bulletLeft, bulletTop  Location check | The location of the bullets that are on the screen at any time are checked to see if the bullets are on the edge of the screen so that they can be removed. The location check is vital for graphics processing in this game |
| pboBullet, pboPlayer, pboZombie  pboAmmo  Collision Check  Type check | The picture boxes that hold the zombie, player, bullet, and ammo supply sprites can collide with each other. They need to go through a collision check so that the program can identify when zombies need to be removed from the screen, or when health needs to be decremented or when the ammo count needs to be incremented. These are controls on the form; therefore, they need to go through a type check so that they can be identified as picture boxes. These are also vital to the game’s graphics. |
| Key is down / up events  User Input Check | The program needs to know what to do when a user clicks a button on their keyboard, so a user input check is used within the keyboard click events. This is so that the key clicked can be identified and the game can produce actions accordingly. It builds up to player movement and is essential for player input |

In general, validation methods will allow me to produce an effective solution that is bug free and smooth. It is also an essential part of the development of my solution because of its close relation to all the variables in my program and the processes that take place.

In my program there will not be many errors or exceptions due to the nature of the code; it consists of straightforward and iterative functions, and user input is discrete, meaning it can only be certain things (arrow keys and spacebar), and will not accept any other keyboard input. In future versions with the introduction of score recordings, amongst other things, exception handling will be a useful method that will be required to stop errors from occurring.

### Testing Approach

My game will be tested as it is developed, this is a required approach to testing because this is a game, and I cannot continue developing a game without testing it with each addition to the code, or else I will not know if it works or not, which is a dangerous approach to programming. When I program some new code, I will test it so that I know it works and I can move onto the next section of programming code.

This is not necessarily a testing approach, more of a base method for development, however it is the most efficient form of testing for a solution of this nature. The definitive version of the game however, will be tested using the black box approach, this includes testing the program regarding the specification.

#### Test Data

The test data refers to the inputs from the user in this game. The following tables present the test data in question and its validity. The validity refers to whether this data is a used parameter in the topic in question. The topics will be – Movement, Shooting, Menu and General, each will have a table of its own.

##### Movement

|  |  |
| --- | --- |
| Test Data | Type |
| Left Arrow | Valid |
| Right Arrow | Valid |
| Down Arrow | Valid |
| Up Arrow | Valid |
| Spacebar | Invalid |
| Enter Key | Invalid |
| Escape Key | Invalid |
| W A S D | Invalid |
| Mouse clicks | Invalid |

##### Shooting

|  |  |
| --- | --- |
| Test Data | Type |
| Left Arrow | Invalid |
| Right Arrow | Invalid |
| Down Arrow | Invalid |
| Up Arrow | Invalid |
| Spacebar | Valid |
| Enter Key | Invalid |
| Escape Key | Invalid |
| W A S D | Invalid |
| Mouse clicks | Invalid |

##### Menu

|  |  |
| --- | --- |
| Test Data | Type |
| Left Arrow | Invalid |
| Right Arrow | Invalid |
| Down Arrow | Valid |
| Up Arrow | Valid |
| Spacebar | Invalid |
| Enter Key | Valid |
| Escape Key | Invalid |
| W A S D | Invalid |
| Mouse Click | Valid |

##### General

This topic refers to the background test data that can be inputted during gameplay, such as exiting the game.

|  |  |
| --- | --- |
| Test Data | Type |
| Left Arrow | Invalid |
| Right Arrow | Invalid |
| Down Arrow | Invalid |
| Up Arrow | Invalid |
| Spacebar | Invalid |
| Enter Key | Invalid |
| Escape Key | Valid |
| W A S D | Invalid |
| Mouse clicks | Valid |

#### Beta Testing

Beta testing for the first prototype involves identifying what each test is for, explaining the test and its expected outcome. It will help me realise errors in the solution and help me evaluate the improvements needed. These tables show the beta testing plans for each topic.

##### Movement & Character

|  |  |  |
| --- | --- | --- |
| What the test is for: | Explanation: | Expected Outcome: |
| Testing if user arrow key input makes the player move | When the user enters the left, right, up, or down arrow key, the program should take this and identify which direction to move the player picture box on the screen, and then move it by the player speed. | Player sprite moves across screen when using arrow keys |
| Testing If enemies move on the screen | During gameplay the zombies will move towards the player, the program identifies the location of the player sprite, and then moves the zombie sprites towards this location. | Zombies chase the player continuously |
| Testing if player stays within bounds of screen | I will move the player sprite to the edge of the screen, and the sprite should not be able to go past it. | Sprite gets stuck at the edge of the screen and cannot get past |
| Testing collision with enemies | The game has a collision detection system that checks if picture boxes are intersecting with each other on the game form. The program then acts accordingly depending on which sprites intersected | Health bar begins to decrease |
| Testing collision with ammo supply | For the player to refill their ammunition count they need to pick up the ammo supply boxes that spawn on the map when the ammo count is low. Again, using the collision system, the game can detect this | Ammo supply disappears and ammo count increments by 5 |
| Testing death | By colliding with enemies until the health bar goes to zero. The game will detect this and act accordingly | Dead image replaces player sprite, all gameplay pauses |

##### Shooting & Ammunition

|  |  |  |
| --- | --- | --- |
| What the test is for: | Explanation: | Expected Outcome: |
| Testing if clicking the spacebar shoots a bullet | The program will create a bullet picture box when the spacebar is initiated. This should be next to the player sprite and in the direction its facing. Should not be initiated if the ammo is 0. The process should be able to be called multiple times consecutively and smoothly | Bullet ejects from player sprite |
| Testing If bullet moves across screen | After the bullet shot is initiated, the newly created bullet picture box needs to move across the screen, in a linear fashion, in the direction the player was facing upon initiation. It also needs to move across the screen at a constant speed, and smoothly. | Bullet moves across the screen in a single direction |
| Testing if zombie dies, score is incremented, and the bullet is removed upon collision with enemy | The game should be calling the collision detection method constantly throughout gameplay, when a bullet hits an enemy, the game should remove both the bullet and the enemy from the form and increment the score. I can do this by shooting and aiming at enemies and observing the result. | Bullet kills the enemy, and the score goes up by 1 |
| Testing if bullet is removed when it reaches end of screen | The bullet image needs to be removed from the form by the program when it reaches the end of the screen and has not collided with any other sprite along the way. I can test this by shooting at the sides of the screen | Bullet disappears when shooting at edge of screen |
| Testing if ammo box is created when ammo is 0 | A function of the game is ammunition replenishing, this is done by spawning an ammo crate on the screen for the user to collide with so that the ammo is replenished, and they can shoot again. I can test this by shooing bullets until ammunition runs out. | I should not be able to shoot bullets anymore, and there should be a new ammo crate sprite on the screen |
| Testing if ammo count depletes | A “Limiting Factor” of the game is the ammunition system. The player has a limited amount of ammo, and it should decrement by 1 each time the player shoots a bullet. I can test this by clicking the spacebar and watching the ammo count | The ammo label should update to have a value 1 less than before shooting |

## Development

### Stage 1

Creating the first game screen with player image, health bar, score count and ammo count:

For the first stage of development, I need to be able to produce a user interface for the player that shows all the essential information that is required during gameplay on the screen. No functions will be made just yet, this stage is purely there to set a base for the next stages.

I first imported all the images I need for the project (Ammo supply, menu background, Zombie sprite images, player sprite images etc.) from an online copyright checked resource (OpenGameArt.com). I added these into the resources panel where all game resources are stored in the game and fetched from. They all have appropriate and relatable names so that they can be called with ease from the main program.

A screenshot of a video game

Description automatically generated

I now must begin setting up the interface, Visual Studio forms has a very feasible system that can be used to create graphical interfaces. I can drag and drop controls from the built-in toolbox, such as labels and picture boxes. They all have different properties and purposes. The player sprite will be in a picture box and will be shown from the start. I must name each control in a way that they can be found and called easily from the main program. They also must be relevant to what the control is. The image below shows a screenshot of the interface that I have created, with annotations outlining its properties and its purpose.

**A screenshot of a computer

Description automatically generated**

I can assign an image from resources to the player picture box within the IDE. There is also a game timer which I brought from the toolbox as well, it is also classed as a control, which can be utilized from the main program. The image below outlines what its purpose is and its name.

A screenshot of a computer

Description automatically generated

### Stage 2

Gameplay; Player character movement, adding enemies to the game that follow the player and damage them (health bar decreases). Shooting to kill enemies. Death when zero. Ammo drops:

In this second stage, the aim is to initiate the actual gameplay, through player movement, enemy movement, health decreasing and shooting at enemies. The top-down diagram below illustrates which parts of the entire game this will cover.

A diagram of a game

Description automatically generated

I first created the KeyIsDown() event, which will automatically detect keyboard input from the user. Depending on the key used, the program will set the player sprite direction of movement, set the direction it is facing, and change the image of the sprite accordingly depending on the direction. At the beginning of the code, a flag is checked for game over. This is to ensure that the player cannot move when the game has ended, and they have died.

A screenshot of a computer program

Description automatically generated

Then I created the KeyIsUp() event for the same keys, this is so that when a movement key is released, the player stops moving.

A screen shot of a computer program

Description automatically generated

Now I need to create the event for the timer, which ticks every few milliseconds, it is crucial to the updating of all graphics in the game. The sprite picture boxes’ locations are incremented with every tick by the movement speed, so that it looks like they are moving smoothly on the screen.

A screenshot of a computer program

Description automatically generated

I have tested the player movement, and it works as expected. I tested this section altogether, as the movement would not work by itself with only the KeyIsUp() event. My only initial problem was having the directions reverted (Clicking down key would make the player move up). I fixed this with ease by just correcting all the directions in the code.

A blue and white box with white text

Description automatically generated

I now need to develop the shooting function, first I must create the bullet class:

A computer screen shot of a program code

Description automatically generated

I then need to create the method for making the bullets:

A computer screen shot of a program code

Description automatically generated

The bullets will have their own timer so that they move on the screen. So, I have set up the timer event which controls bullet direction and checks if bullet has reached the edge of the screen.

Then I also created the ShootBullet() method outside of the class, its purpose is to link the main program with the bullet class and create an instance of a bullet and set it is coordinates on the screen.

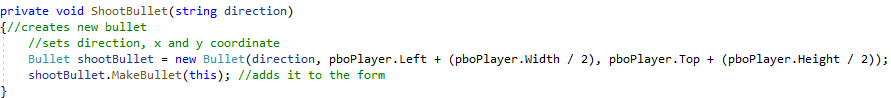
Now to have all this work together, I need to update the KeyIsUp() event so that when the spacebar is clicked, a bullet is shot using the bullet class and its methods

Screenshots of code for these three sections below, in order:

A computer screen shot of a computer code

Description automatically generated

Added as a new method:



Added to KeyIsUp():

A screenshot of a computer

Description automatically generated

The ammunition needs to be replenished when it reaches 0, this is done through ammo drops. The code below shows the method for dropping ammo on the screen

A screen shot of a computer code

Description automatically generated

This is then called from the KeyIsUp() event:

A screenshot of a computer program

Description automatically generated

Player movement, shooting and the ammunition system have been set up. I tested throughout the making of the code, but after these parts I began a larger test with actual criteria. It has worked as expected after a few minor tweaks such as changing the spawn location of the ammo pickups so that they do not spawn on top of the controls at the top of the screen (ammo label, health bar etc.), and making the bullets spawn closer to the player sprite.

I have created the Collision Detection method, it checks collision between ammo pickups and the player, for now:

A computer code with text

Description automatically generated

I then must call this method in the game loop event:

A screen shot of a computer program

Description automatically generated

I have tested this by depleting ammo by constantly shooting until it reaches 0, a supply box needs to have spawned, and upon collision with the player it disappears. However, on the first test round, the supply box would not disappear, I realised that it was due to missing an “x” on the line ((PictureBox)x).Dispose().

The next job is to start making zombies on the screen for the user to eliminate, they will be spawned in random locations on the screen:

A screen shot of a computer code

Description automatically generated

I now need to create a method used for restarting the game, which can be used whenever the game is first started, and when restarting the game again after death. It also spawns zombies on the screen using the makeZombies() method I just created, and resets all attributes:

A screen shot of a computer program

Description automatically generated

After successful testing, the zombies spawn on random places on the screen.

Now I will add an if statement to the collision detection method that checks if a player has hit a zombie. When detected the player will begin losing health:

A screen shot of a computer code

Description automatically generated

I then needed to make this work visually by updating the Game Loop Event so that the Health Bar value reflects the playerHealth attribute value with each tick of the clock, and death / gameOver is initiated when the health is 0:

A screenshot of a computer program

Description automatically generated

I tested this by running the program and moving the player sprite to the spawned zombies and colliding with them. At first the health did not decrease because I had again, missed the x on the 3rd line. After adjustments, the health bar then gradually decreased as expected, and the player image turned into the dead player image.

I added a selection statement into the collision detection method for bullets colliding with zombie sprites:

A screenshot of a computer program

Description automatically generated

This was tested by running the game, aiming at zombie sprites, and hitting the spacebar button, when the bullet hit the zombies, they first stopped at the zombie, but neither the bullet nor the zombie had disappeared from the form. So, I had to add the “.Dispose()” function for the controls to fully remove them from the form.

Next is making the zombies chase the player, this took a lot of thought, as determining the direction of movement depending on the location of the player sprite was difficult:

A screenshot of a computer program

Description automatically generated

I added it to the Game Loop event so that they move towards the player constantly and smoothly:

A screenshot of a computer

Description automatically generated

I tested this by running the game and letting the zombies move towards the player, initially they moved the wrong way in the vertical direction, which meant I had mixed up some of the directions in the code, it was an easy fix. They all moved simultaneously towards the player sprite.

Now the main base of the gameplay has been completed; player movement, shooting, ammo supplies and enemies. I again tested the entire program; however, it is not yet robust because you cannot restart the game from the same form, rather you need to close and re-open the game.

### Stage 3

Simple menu, appears after death and beginning of game, exit and start:

Next is the menu form. It is a simple form with only a few controls, and an uncomplicated design, with a simple, but with an effective purpose; to allow the user to navigate the game easily. I can create a second form in the IDE and add controls to it. Both forms can be called from the main program at the same time, or individually.

A diagram with red circle and white text

Description automatically generated

A screenshot of a computer

Description automatically generated

The three controls available are a start button, exit button and score label. The annotated image above shows a screenshot of the menu interface, and it outlines each control.

There is not a requirement for code apart from setting up events. Events are assigned to buttons and the IDE automatically creates code in the main form program for me to add my code in. When the user clicks a button, the event will be called. This can be done from the built-in properties window in the IDE:

A screenshot of a computer

Description automatically generated

I have created events for each button in the menu, called btnStart\_Click and btnExit\_Click. I have made the start button carry out the following code. Its purpose is to create a new instance of the main game form and show it to the user. Then close and hide the current menu from the user’s screen. Using “this.Close()” I can easily close the game and any instances with one line of code from the Exit button event:

A screen shot of a computer code

Description automatically generated

I have set the program to run the menu form first from the main program code. The parameters in frmMenu() are for the previous score, and a Boolean flag showing whether this is the first time the game is being ran.

A screen shot of a computer code

Description automatically generated

I then added code to the Game Loop Event in the game form code file that causes the menu to open upon death:

A screen shot of a computer code

Description automatically generated

I have tested this stage with each addition of code, rather than writing all the code and testing it because this can lead to large errors. So far it has been successful with only 1 error occurring when I had to open the main game form when the start button is clicked on the menu form. The program was fixed by adding “newGame.ShowDialog()”, which shows the form on the user’s screen, rather than only initiating an instance of the game form

### Forms of Validation Used

Since the only user inputs are through keyboard keys during gameplay, and mouse clicks on the menu in the first prototype, there are not any catch methods used because the user can use any keyboard key, and the program will automatically ignore the input if it has not been registered by me in the code. So therefore, there are not any try-catch methods used, and data is loaded throughout the game as discrete, set values that will not throw errors because of the nature of the game.

Throughout my code I have used the validation checks from the Validation section in this coursework, such as range checks within locations checks for the player sprite, so that zombies can chase the player.

## Stakeholder Feedback & Reviews

### Requirements

These are my requirements for the first prototype, from the success criteria. The table shows the requirement, why it is needed, and whether it has been met or not.

|  |  |  |
| --- | --- | --- |
| Requirement: | Justification | Met? |
| The game will have a top-down, 2-dimensional view | This is the most preferable dimension and view for a player, and it is simple to implement |  |
| There will be an ammunition system for the guns | This is a limiting factor that will keep the player “on their toes,” the ammunition will go down by 1 after every shot |  |
| There will be a health bar system for the player | This is another limiting factor that will make the user change how they play. The health of the player will decrease when an enemy hit them |  |
| There will be a score system based on eliminations | This is to keep the game competitive and make the user feel a sense of achievement if they reach a high score |  |
| A menu system for navigation between high score tables, pausing, restarting the game etc. | This is purely navigational and is there for the user to be able to access various parts of the game and complete system functions such as exiting |  |
| Keyboard controls – Up left down right and spacebar for shooting | This is so that the user can easily control their character and kill the enemies. |  |
| Easy to call procedures | Procedures will be named appropriately so they can be used throughout the program for quick and efficient code |  |
| Flexible code | Code will be open to additions so that it can be improved easily, and new prototypes can be developed from the previous structure |  |

### Feedback & Summary

I have fulfilled most of the requirements that I needed to complete for this stage of development. I have created the base structure of the game, which can be further built on in further development. So far, this base includes player movement, shooting enemies, enemies chasing the player and a simple menu system. I had planned to implement a pausing system, but I could not integrate this properly for this first prototype. So, I therefore plan to add this in further development for another prototype.

I have now pulled the game executable from the IDE and sent it to the stakeholders Maimoon and Hayder. After a period of in-person testing, (I was there with them as they tested the prototype), I received some valuable feedback on the game from my stakeholders.

They both think that this basic structure is solid, and more than usable. They also understand that future versions of the game will fulfil the rest of their requirements and the remaining success criteria. They said this first version is, for obvious reasons, “boring to play,” however again they said that they understand that this is a basic prototype version of the game. I also noted that both Hayder and Maimoon said that the health decreases too quickly when in contact with the zombies. I will therefore slightly decrease the rate at which the health diminishes. Hayder said that they would “like to see a pausing system in the game” for the next development stage, as it is something they expect in an endless shooter game with a score system.

So far, the progress I have made is good for a project of this magnitude and length. My next steps in code development will include fulfilling the requirements for the second prototype, as well as adjusting a few values such as health, and the player/zombie speed too.

# Prototype 2

## Planning & Design

In this section I will present my clear idea of how I will approach my second prototype. My initial ideas may be subject to change as I develop.

### Overview of Structure

I will use a class diagram to represent the different classes and objects in the solution. It is an ideal way to show how all the distinct parts work together, it will also indicate the attributes and methods that each part of the game uses. This is all for the second version of the solution.

These class diagrams are built from the original class diagrams from the first prototype. Many attributes and methods will be added to the classes that have already been implemented. New features, or edited features will be highlighted in Yellow:

A screenshot of a computer program

Description automatically generated

The survival mode will almost be the same in terms of classes, procedures and methods used throughout the code. This is because I have already laid out the base for the game, which is the survival mode. Any new additions to survival will be in the third prototype. For now only visual changes to the form will be made for survival.

A screenshot of a computer

Description automatically generated

The menu will now be updated to account for the new campaign mode. It will also be given background music. It will also make use of a parameter when the menu is opened, that flags whether the menu was opened from pausing gameplay, or if it’s after game over, or the first time the game is ran. A second form will be created, hence the new partial class for frmMenuCampaign. This will host the access to the different levels of the campaign game mode.

The campaign game mode sees the introduction of a new class – Zombie. This class will be used to implement different difficulties for the enemies, as well as different sprites, and a whole new enemy elimination system. The method of making enemies on the screen will now change because of this. The new feature of health pickups will see the use of a new method – dropHealth().

A screenshot of a computer program

Description automatically generated

### Screen Designs & Usability Features

In this section I will be providing images of the forms I have created for the second prototype and will be describing, explaining, and justifying each part of every screen. Usability features of each form will also be pointed out and justified. This section will aid me in realising how user-friendly my forms will be and will also help me understand the processes that a user will go through when making use of this product. Screen designs from the first prototype will be amended in this second prototype, I will show these changes here too.

#### Menu

The main menu is still very similar, just with the addition of a “Campaign” button which takes them to another window. It has also been reworked in terms of design – it is now a lot more aesthetic than before.

A screenshot of a video game

Description automatically generated

The plan is for the menu to be also accessible when the user pauses the game. This requires a “Resume” button so they can continue to play. This is the view for that button – it will replace the campaign and survival buttons:

A screenshot of a video game

Description automatically generated

The second window I have created is a new form, for the campaign mode access. It has the same theme. It gives user access to the 3 levels of the campaign mode – of which only 1 will be developed for this prototype. The aim of having the other two visible for the user, is so that my clients can understand that there will be further development for the campaign mode for the next prototype. The user can return to the “Main Menu” with the return button.

A screenshot of a game

Description automatically generated

#### Game

The survival game mode will have the exact same look.

The first level will have walls:

A screenshot of a video game

Description automatically generated

* Smaller sprites to fit a larger map.
* HUD remake
  + Countdown+
  + Wave No.+
* Picture box walls

Similar information on screen as survival mode, with addition of timer and wave number.

### Algorithm Designs

After having identified an overall structure of my second prototype, I will now give an outline of my algorithms required for each new feature. I will be using flowcharts to present my procedures, functions, and subroutines.

#### Resizing Window

This algorithm shows the steps that need to be taken to automatically change the size of all controls on the form to the correct ration whenever the user resizes their window.

A diagram of a window

Description automatically generated

This all occurs within the event caller for resizing the form, which is a built-in function in visual studio winforms.

#### Pausing

This algorithm goes through the steps taken by the program whenever the user initiates the pause feature.

A diagram of a software algorithm

Description automatically generated

#### Navigating Menu

This algorithm goes through the possible actions that the user can take, and how the program should respond, when navigating the game menu

A diagram of a menu navigation

Description automatically generated

#### Creating & Removing Enemies

With the new difficulty system that I will develop, the set of tasks that need to be carried out by the program will have to differ from the original system to make up for the new features. This Algorithm presents how the enemies will be added to the user’s screen.

A diagram of a game

Description automatically generated

This algorithm shows how enemies will be removed from the user’s display, and the tasks carried out to complete the process.

A diagram of a system

Description automatically generated

#### Dropping Health Pickups

This flowchart is for the steps taken to drop a health pickup on the screen for the user to pickup.

A diagram of a process

Description automatically generated

This flowchart is for when the user comes in contact with the health pickup.

A diagram of a program

Description automatically generated

#### Collisions

Here I will go through the new types of collisions – new features such as the new enemy system mean that collisions will work differently.

Below is an algorithm flowchart that follows the steps taken when the player collides with physical game objects, walls, or enemies. The aim is to remove the possibility of images overlapping. This makes for a smoother game experience and will make more sense in terms of how such a game should work.

A diagram of a game

Description automatically generated

This is ran within the collision method, but has a separate algorithm to that of the algorithm that goes through the actual effects on player and score, health etc.

This algorithm is still within the collision method – however this algorithm goes through the steps taken to affect the player, and the score / health etc. Each type of enemy has a different effect, hence objects are used with an enemy class, so that each enemy has a different effect on player’s health.

A diagram of a game

Description automatically generated

### Key Variables & data Structures

In this section I will be presenting my key variables and data structures that I have identified during the formation of my solution’s algorithms/

#### Entities

A diagram of a game

Description automatically generated

This relationship diagram outlines the solution’s structure. The main menu links to the campaign menu, both of which link to the game. The game contains player and enemies, one player to many enemies. Circular structure – from the game to the menu.

#### Databases

The only databases are the enemies list, and resources. The enemies list contains the multiple enemies on the screen. It is a dynamic data structure – meaning it can be added and removed from dynamically without needing to set a specific size to the array.

The Resources folder contains all the images and sound files that I will use for development:  
Coloured green are the resources that have already been specified in the first prototype

|  |  |
| --- | --- |
| Resource name & type: | Use: |
| Up, image | Player sprite facing up image, starting position, used when player is moving up |
| Right, image | Player sprite facing right image, used when player is moving to the right |
| Left, image | Player sprite facing left image, used when player is moving to the left |
| Down, image | Player sprite facing down image, used when player is moving down. |
| Zup, image | Zombie sprite facing up image, used for when the zombie is moving upwards |
| Zdown, image | Zombie sprite facing down image, used when the zombie is moving downwards |
| Zright, image | Zombie sprite facing right image, used when the zombie is moving towards the right |
| Zleft, image | Zombie sprite facing left image, used when the zombie is moving towards the left |
| menuBackground, image | Assigned to the background of the menu form. It is the menu background |
| Dead, image | Player dead sprite, this is applied to the player’s picture box upon death |
| ammo, image | ammo drop image for when an ammunition box pickup is spawned on screen |
| health drop, image | Health drop image for when a heath pickup is spawned on screen |
| Menu music, sound file | Ambient background music to be played while accessing menu |
| Player death, sound file | Sound that runs when player reaches game over |
| Shot, sound file | Sound that runs when player uses shot function -spacebar |
| Reload, sound file | Used when user picks up ammunition box |
| Zombie1 – up, down, right,left images | First type of campaign zombie |
| Zombie2 | Second type of campaign zombie |
| Zombie3 | Third type of campaign zombie |
| Zombie4 | Fourth type of campaign zombie |

#### Key Variables

I will go through the main variables for my classes, this is to show the use of each variable and their justification. I will also show which subroutines and methods use each variable. I will only present NEW variables that are new to the second prototype.

|  |  |  |
| --- | --- | --- |
| Variable: | Relevant subroutines / methods / events | Purpose & Justification |
| controlOriginal – main form | resizeWindow()  formResize() | Used to store original size of all controls in the form, so that they can be later used to calculate a ratio, and expand controls and window correctly. |
| Paused – menu form | loadMenu() | Used as a Boolean flag so that the program knows whether the menu has been opened from pausing the game, or if it has been opened from the start / game over |
| Music – all forms | playMusic() | Used to hold sound player class file type so ambient background menu music can be played. |
| Level, speed, damage, health – enemy class | makeZombie() | Attributes of new zombie class – will be sued to distinguish different types of zombies because they will have different qualities and characteristics. |
| Image – enemy class | RemoveZombie()  makeZombie() | Used for holding the different images of the different enemies, because different levels of enemies will look different. |

## Development

This section will present my actual game code, how it is written, explanations for what code is doing, and proof of testing. Not all sections of code will be shown, rather only new additions will be presented here.

### Stage 4

**Features to be coded and added in this stage:**

* Automatic form + control maximisation
* Enemy difficulty system (new class)
* Introductory level creation (1st Campaign level)

#### Automatic Maximisation

To do this I must use a basic maths technique – using ratios; I will calculate ratios for control-to-form objects when the game is first run. Then use this ratio to calculate new locations and sizes for every control whenever the window size is changed.

* Create variables for original control sizes – Rectangle Class used for shapes

A close up of text

Description automatically generated

* When the game is first loaded, all the original variables I created earlier are fulfilled

A screen shot of a computer

Description automatically generated

* Creating a new procedure – resizeControl(). Gets control-to-form ratios, sets new X and Y coordinates (location) with this ratio. Sets new width and height values (size) with this ratio.

A screen shot of a computer code

Description automatically generated

* Create event procedure that is automatically called whenever the user resizes their window. In this procedure the recently made procedure (resizeControl()) is called multiple times with all controls in the form

A screen shot of a computer code

Description automatically generated

These pieces of code need to be updated all the time with each new control that is added to the game. This will also be added to the campaign game mode forms. I tested this feature by running the game, and resizing the window to different sizes:

<https://screenpal.com/watch/c0XUXjVmXi5>

#### Creating First Campaign Level form

* Create new form – lvlOne. Copy screen design from survival. Add wave No. label – starts at 0/3. Add countdown label. Add picture boxes with wall image, arrange to create an immersive gaming experience.
* I need a smaller sized player and zombie sprites – to have a larger map for the player. These will be new resources added to the resource database – they have names with a hint to campaign version of sprite in the name.

See the previous Screen Designs & Usability Features section - [Game Screen Design](#_Game) - for the screen image.

* First, I will set the gameplay to be like that of the survival mode – code is copied from survival form.
* Edits are made to the sprites used.  
  New images for enemies and player picture boxes that are used only for the campaign game mode. They are smaller – this is to allow for a larger map (smaller sprites and larger form size = illusion of more space on screen)
* New controls are added to the automatic control resize method.

A screen shot of a computer program

Description automatically generated

* New Timer is added to form – this will be used for countdown.

A screenshot of a computer

Description automatically generated

* Countdown label is linked to timer.

New variable time left created to keep track of countdown time:



Timer event created for the countdown timer, and following code added:

A screen shot of a computer code

Description automatically generated

* Timer starts from 1 minute, and decreases, when the player begins playing.

Testing:



Counting down and the label is updated every 1 second.



* For now, enemies are spawned one at a time for the first wave, once one is eliminated, another is spawned.
* When 10 enemies have been eliminated within the countdown a new wave begins.
* Game waits 10 seconds to announce new wave, before adding more enemies.
* Wave label updated to 1/3.
* Enemies spawned twice at a time.
* Until 16 enemies eliminated within 2 mins.
* New wave, label updated, 10 second wait.
* Countdown starts from 3 mins this time.
* 3 enemies at a time for third wave until 21 eliminated within countdown.
* If countdown is surpassed for each wave, then notify user with a large label – pause 5 secs.
  + End game – display menu with restart button and return to main menu button.

#### Menu Development

* New form with name frmMenuCampaign – this form will be used as the campaign menu section which will be displayed to the user when they select Campaign mode from the menu. The screen design has been shown previously in Screen Designs & Usability Features section: [Campaign Menu Screen Design](#Design)
* The buttons are added, and the corresponding images are placed in the backgrounds of the buttons – I will no longer be using text, instead images that contain text which have a better look.
  + Properties of the Buttons – names btnLvlOne, btnReturn etc.

A screenshot of a computer

Description automatically generated

* A campaign button is added to the main menu. When clicked the following code runs:

A screenshot of a computer program

Description automatically generated

* + A new instance of the campaign section form is created using its partial class.
  + The background music is paused.
  + The current main menu form is hidden and closed.
  + The new campaign form is displayed to the user and activated.

This was tested by clicking the Campaign button on the main menu after running the solution:

A screenshot of a video game

Description automatically generatedA screenshot of a game

Description automatically generated

Testing concluded successfully.

* The return button is set up, by creating its user click event, and adding the following code to it:

A screenshot of a computer program

Description automatically generated

* + A new instance of the main menu form is created using its partial class.
    - Parameters used are flags to show it’s not a pause menu and is also a first run.
  + The background music is paused.
  + The current campaign menu form is hidden and closed.
  + The new main menu form is displayed to the user and activated.

This is tested by running the solution, then moving to the campaign menu, then clicking the return button:

A screenshot of a game

Description automatically generated A screenshot of a video game

Description automatically generated

* Then the Level One button is set up, using its user click event. The following code is added:

A screenshot of a computer code

Description automatically generated

* + A new instance of the level one game form is created.
  + The background music is stopped.
  + The menu is closed, and the new game form is displayed to the user.

This was tested by running the solution, navigating to the campaign menu section, and then clicking on the level one button:

A screenshot of a game

Description automatically generated

As a result, the first level game form was opened successfully:

A screenshot of a video game

Description automatically generated

The menu development has now been completed.

[Insert testing video link of MENU ONLY]

### Stage 5

**Features to be coded and added in this stage:**

* Implementation of health pickup drops
* Addition of game sounds throughout the solution:
  + Shot sound.
  + Background menu music
  + Reloading ammunition sound
  + Player death
  + Health regeneration (health pickup)
* Enemies class for different enemy levels

#### Health Drops System

The algorithm used for dropping health supplies is very similar to the algorithm for dropping ammunition supplies. The only changes are that the player’s health increases rather than the ammunition count, and a different image is spawned on the user’s screen.

The following procedure was added to the level One form code:

A computer screen shot of a program

Description automatically generated

This is the procedure DropHealth(), it takes the following steps to fulfil this new feature:

* Creates a new picture box instance and assigns the new health drop image to this instance.
* Its location is set using the random variable – it returns a random number between the values given (inside the play area in this example)
* The size mode is set to auto size – It will be the size of the original image, which is a small 25x25 pixel square.
* It’s given the tag “health” so that it is identifiable when it is collided with.
* The image is added to the screen, and the player image is brought to the front of the screen to ensure it does not cover the player sprite.

This procedure now needs to be called when its needed – which is when the player has low health, the following code has been added to the timer ticking event:

A screenshot of a computer code

Description automatically generated

If the players health is lower than 15, then the procedure is called. A flag dropped health is used to know whether there is currently a health pickup on the screen. It is set to true. It will be useful in the upcoming code.

Now collisions between the player and the health pickup needs to be set up:

A screen shot of a computer code

Description automatically generated

This is added to the Collision detection method – when the player sprite intersects with a picture box that has the health tag:

* The health pickup image is removed from the screen.
* The player’s health is incremented by a fair value – 25.
* The dropped health flag is set to false now – this allows for another health drop to be spawned because there isn’t already currently one on the screen – this stops multiple drops being spawned at the same time.

The health pickup feature can now be tested by running level one by navigating to the previously developed campaign menu.

* Beginning the gameplay by clicking the enter key.
* Purposefully reducing player health by colliding with enemies
  + This is done until the health decreases to a point at which the drop health procedure is called.
* A single health drop should spawn on the screen.
  + Approaching it using arrow key movement

#### Game Sounds

Game sounds are easy to implement especially with the windows forms Media library which can be used to easily play and stop sounds from resources. The tests for all these sounds will be shown at the end within one video.

* Sounds are imported into solution resources:

[Image of sound files]

* Shot sound.

Whenever the procedure for making a bullet is called, the system will play the shooting sound:

[insert code image of shot sound]

* Background menu music

The form menu load procedure which is called whenever the menu first opens is amended to include the playing of background menu music:

[start music code]

Its then stopped when the form is closed.

[stop music code]

This has been done for both the main menu and the campaign menu forms.

* Reloading ammunition sound

When the collision detection method is called, and a collision between the player sprite and an ammunition box is identified, the sound will be played:

[insert reload sound code]

* Player death

When game over is initiated after the player loses all their health, the player death sound is induced:

[Insert death sound code]

* Health regeneration (health pickup)

Like the reloading sound this sound will be played when the player sprite collides with a health drop image:

[Insert health reg sound code]

#### Enemies Class

The aim of this class is to give a wide range of enemy difficulties that each have different characteristics and properties, to give a wider range of experiences for the user.

* A new class is created, called Enemy:

[Insert class image]

* It has attributes for enemy speed, health, damage, level, and image.
* It has an instantiation procedure that is used when creating a new enemy, it only takes in the level of the enemy that is needed to be made.
* It has an add enemy subroutine used to add enemies to the form:

[Insert addEnemy() code]

Depending on the level of enemy that is added in as a parameter, different images, values of damage, health, and speed are assigned to the enemies. It is then added to the screen within the make zombies method which has been remade for the campaign mode:

[make zombies amendments]

* It has a remove enemy subroutine which is used to remove the enemy from the form:

[insert removeEnemy() code]

### Stage 6

**Features to be coded and added in this stage:**

* Full separation of Campaign and Survival modes
* Second menu form – specifically for Campaign access
* Pausing function during gameplay
* Even further refinement of First level (explained more later)

#### Pause Gameplay Function

* Event for user input of Escape key:

[Insert e code]

Creates new menu form. Score and true for pausing passed in as parameters. Then within the menu form, this is noticed and the score label is changed to show the current score. The buttons are changed to show resume and exit.

#### Enemy Collisions – Prevent overlaps

* Collision detect method is amended to stop the enemies from overlapping with each other and the player sprite:

[code]

When with player sprite it is also moved 10 pixels away from the player. When with another enemy it is 5 pixels away from each other

#### Walls & Objects Collisions (1st Level Refinement)

Movement in direction of wall stopped when intersect.

## Stakeholder Feedback & Reviews

Here I will essentially evaluate the completion of my second prototype – whether it has fulfilled my proposed solution – and whether it appeals to the stakeholders. Split into two sections: “Requirements”, and “Feedback & Summary”.

### Requirements

These are my requirements for the second prototype, from the success criteria. The table shows the requirement, why it is needed, and whether it has been met or not.

|  |  |  |
| --- | --- | --- |
| Requirement | Justification | Requirement met? |
| There will be multiple enemies with a range of difficulties | The game needs an element that keeps the user interested, the enemies will become harder in difficulty, and some will be extremely hard |  |
| A menu system for navigation between game modes, and pausing, | This is purely navigational and is there for the user to be able to access various parts of the game and complete system functions such as exiting |  |
| A multi-level system | This is so that the user stays interested and keeps the game action filled. |  |
| Easy to call procedures | Procedures will be named appropriately so they can be used throughout the program for quick and efficient code |  |
| Flexible code | Code will be open to additions so that it can be improved easily, and new prototypes can be developed from the previous structure |  |
| Campaign game mode | This gives the user a choice and reflects a different type of gameplay; gradual increase in difficulty. |  |
| Design features for user usability such as window maximisation | Allows user to navigate the game with ease, also makes everything look clean and appealing. |  |
| Smooth graphics | The movement of sprites in the game must not be glitchy, for a smooth user experience. Refers mainly to the new no-overlap system for sprites. |  |
| Sounds | Introduces extra immersion into the game for the user |  |

### Feedback & Summary

I will now package up my solution into a runnable windows execution file and send it to my stakeholders – Maimoon and Hayder. They will give me feedback on the new features added and will also point out any unseen problems / errors / bugs in the solution. They will also tell me about any previously not noted features that they think would be beneficial for my solution – I will evaluate these and decide if they should be implemented into my success criteria, for it to be developed in my third prototype.

Due to their unavailability, Philip will not be able to test or look at my second prototype. The summary will give a light overview of what has been added, and their relative success for the project.

#### Stakeholder 1 – Hayder

Hayder likes the new additions and is excited to see further progression with more levels. He thinks the menu works as it should and allows proper navigation for him between sections of the game. They don’t like the basicness of the player sprite and their weapon.

#### Stakeholder 2 – Maimoon

# Prototype 3

## Planning & Design

In this section I will present my clear idea of how I will approach my third prototype. My initial ideas may be subject to change as I develop.

### Overview of Structure

I will use a class diagram to represent the different classes and objects in the solution. It is an ideal way to show how all the distinct parts work together, it will also indicate the attributes and methods that each part of the game uses. This is all for the third version of the solution.

### Screen Designs & Usability Features

### Algorithm Designs

### Key Variables & data Structures

#### Entities

#### Databases

#### Classes

#### Key Variables

## Development

### Stage 7

Boss levels in campaign mode. Gun pickups

A second level was made, like the first, it just had a different layout of objects (walls and crates), and the introduction of the 3rd level of enemy (the fast one with low health but high damage). The third level was then made, a boss level. The 4th level of enemy, high health, high damage, but relatively low speed. Less objects and walls. Larger enemy sprite. A singular specified health bar at the bottom of the screen indicating the health of the boss enemy. A new attack for the boss – a sort of “orb” that was short from the enemy sprite in all four directions every 20 seconds. The user had to avoid this else it would take half of the player’s health. Simple contact would take a quarter of health. Once the enemy was eliminated, the level would continue with a single wave of a random multitude of lower-level enemies, for about 30 seconds, user must survive this short high intensity wave until the countdown ends – then the game ends if successful.

Only a second gun was implemented – a simple rifle, which could hold more ammunition. It was placed on the ground as a simple image in the first and third level, once picked up, it could not be changed back with the original gun. So very simple not much of a good of a system for gun pickups. No inventory. The sprite was changed though to accommodate for the new gun picked up.

### Stage 8

Previous scores accessible from Menu and High score (for a single user, no accounts yet)

Scores were saved when a user dies in survival mode. As well as their stats being saved for each campaign level upon completing it. Was saved in a csv (comma separated value) text file. A sorting algorithm was made to sort these values in order, and they were then visualised in a page in the menu with a grid showing date and time of score, and the score.   
  
The highest score at top and highlighted.

### Stage 9

Continuation of stage 8 – accounts system, scores recorded and saved, login and registration system.

CSV file was updated to hold the records for username, password, high score, and the list of the rest of the scores. The first index of scores is specifically set for the high score only. This allows for easy retrieval, just specifying the index [3], rather than having to do a sort of all the values every time.

New menu page accessible from the first running of the game – login and registration. Simple text boxes with labels above them showing which were for registration and which were for login. Once password and username entered, it was saved into the csv file on a new line when registering. When logging in, the username is looked for throughout the entire csv file – a simple binary search. Once found, its next index is checked against the password entered by the user. If any are incorrect / don’t exist, an error window is displayed to the user saying, “incorrect username / password’. Once logged in successfully, the login window is closed, and they’re sent to the main menu.

When they click the statistics button, they’re sent to the statistics page which pulls the scores as said in the previous stage.

## Stakeholder Feedback & Reviews

### Requirements

### Feedback & Summary

# Code Listing

Here my full program’s code will be copied and pasted. There are different forms for the different pages of the game, and therefore this section will be split into further sections indicating which form the code belongs to.

## Main Menu

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Media;

namespace Zombie\_Survival

{

public partial class frmMenu : Form

{

private string labelText;

private bool paused;

public SoundPlayer Music = new SoundPlayer(Zombie\_Survival.Properties.Resources.Menu\_Music);

public frmMenu(string text, bool first, bool pause)

{

InitializeComponent();

labelText = text;

if (first == true)

{

lblScore.Hide();

}

if (pause == true)

{

paused = true;

btnStart.BackgroundImage = Zombie\_Survival.Properties.Resources.btnResume;

btnCampaign.Hide();

}

}

private void btnStart\_Click(object sender, EventArgs e)

{

if (paused == true)

{

Music.Stop();

this.Hide();

this.Close();

}

else

{

Music.Stop();

frmGame newGame = new frmGame();

this.Hide();

newGame.ShowDialog();

this.Close();

}

}

private void btnExit\_Click(object sender, EventArgs e)

{

System.Windows.Forms.Application.Exit();

}

private void frmMenu\_Load(object sender, EventArgs e)

{

lblScore.Text = "Score: " + labelText;

Music.PlayLooping();

}

private void btnCampaign\_Click(object sender, EventArgs e)

{

frmMenuCampaign newCampaignMenu = new frmMenuCampaign();

this.Hide();

Music.Stop();

newCampaignMenu.ShowDialog();

this.Close();

}

private void btnMainMenu\_Click(object sender, EventArgs e)

{

}

}

}

## Campaign Menu

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Media;

namespace Zombie\_Survival

{

public partial class frmMenuCampaign : Form

{

public SoundPlayer Music = new SoundPlayer(Zombie\_Survival.Properties.Resources.Menu\_Music);

public frmMenuCampaign()

{

InitializeComponent();

}

private void btnLvlOne\_Click(object sender, EventArgs e)

{

lvlOne newLvlOne = new lvlOne();

this.Hide();

Music.Stop();

newLvlOne.ShowDialog();

this.Close();

}

private void frmMenuCampaign\_Load(object sender, EventArgs e)

{

Music.PlayLooping();

}

private void btnReturn\_Click(object sender, EventArgs e)

{

frmMenu newMainMenu = new frmMenu("", false, false);

this.Hide();

Music.Stop();

newMainMenu.ShowDialog();

this.Close();

}

}

}

## Survival Mode

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Media;

namespace Zombie\_Survival

{

public partial class frmGame : Form

{

//creates variables/attributes

bool goUp, goDown, goLeft, goRight;

bool gameOver = false;

string facing = "up"; //used to guide bullets

double playerHealth = 100;

int playerSpeed = 10, ammo = 10, zombieSpeed = 4, score = 0;

//variables for resizing controls when going fullscreen

private Rectangle frmGameOriginal, lblKillsOriginal, lblHealthOriginal, lblAmmoOriginal,

prgHealthOriginal, pboPlayerOriginal, lblEnterToStartOriginal, pboLabelAmmoOriginal,

pboLabelKillsOriginal, pboLabelHealthOriginal;

//Random variable used for spawning pickups

Random rnd = new Random();

private void frmGame\_Resize(object sender, EventArgs e)

{

resizeControl(lblEnterToStartOriginal, lblEnterToStart);

resizeControl(lblKillsOriginal, lblKills);

resizeControl(lblHealthOriginal, lblHealth); //This procedure will resize all of the controls on the screen

resizeControl(lblAmmoOriginal, lblAmmo); //when the user changes the size of the window

resizeControl(prgHealthOriginal, prgHealthBar);

resizeControl(pboPlayerOriginal, pboPlayer);

resizeControl(pboLabelAmmoOriginal, pboLabelAmmo);

resizeControl(pboLabelHealthOriginal, pboLabelHealth);

resizeControl(pboLabelKillsOriginal, pboLabelKills);

}

private void pboPlayer\_Click(object sender, EventArgs e)

{

}

private void resizeControl(Rectangle r, Control c)

{

float xRatio = (float)(this.Width) / (float)(frmGameOriginal.Width);

float yRatio = (float)(this.Height) / (float)(frmGameOriginal.Height); //This gets the ratio of control sizes

//so that they can be expanded correctly

int newX = (int)(r.Location.X \* xRatio);

int NewY = (int)(r.Location.Y \* yRatio);

int newWidth = (int)(r.Width \* xRatio);

int newHeight = (int)(r.Height \* yRatio); //sets their new widths and location, using the the ratio

c.Location = new Point(newX, NewY);

c.Size = new Size(newWidth, newHeight); //completes the process

}

private void frmGame\_Load(object sender, EventArgs e)

{

lblEnterToStartOriginal = new Rectangle(lblEnterToStart.Location.X, lblEnterToStart.Location.Y, lblEnterToStart.Width, lblEnterToStart.Height);

frmGameOriginal = new Rectangle(this.Location.X, this.Location.Y, this.Size.Width, this.Size.Height); ;

lblKillsOriginal = new Rectangle(lblKills.Location.X, lblKills.Location.Y, lblKills.Width, lblKills.Height);

lblHealthOriginal = new Rectangle(lblHealth.Location.X, lblHealth.Location.Y, lblHealth.Width, lblHealth.Height);

lblAmmoOriginal = new Rectangle(lblAmmo.Location.X, lblAmmo.Location.Y, lblAmmo.Width, lblAmmo.Height);

pboPlayerOriginal = new Rectangle(pboPlayer.Location.X, pboPlayer.Location.Y, pboPlayer.Width, pboPlayer.Height);

prgHealthOriginal = new Rectangle(prgHealthBar.Location.X, prgHealthBar.Location.Y, prgHealthBar.Width, prgHealthBar.Height);

pboLabelAmmoOriginal = new Rectangle(pboLabelAmmo.Location.X, pboLabelAmmo.Location.Y, pboLabelAmmo.Width, pboLabelAmmo.Height);

pboLabelHealthOriginal = new Rectangle(pboLabelHealth.Location.X, pboLabelHealth.Location.Y, pboLabelHealth.Width, pboLabelHealth.Height);

pboLabelKillsOriginal = new Rectangle(pboLabelKills.Location.X, pboLabelKills.Location.Y, pboLabelKills.Width, pboLabelKills.Height);

//all above are saving all the control sizes and locations so that they can be changed when user resizes window

this.WindowState = FormWindowState.Maximized;

gameOver = true; //stops game from running

lblEnterToStart.Show(); //Shows informational label so user can start by clicking enter

tmrGameTimer.Stop(); //stops the game from running too

}

private void lblKills\_Click(object sender, EventArgs e)

{

}

List<PictureBox> zombiesList = new List<PictureBox>();//List of zombies to be added/removed

public frmGame()

{

InitializeComponent();

RestartGame();

}

private void KeyIsDown(object sender, KeyEventArgs e)

{//left

if (gameOver == false)//makes sure cant move when dead

{

if (e.KeyCode == Keys.Left)

{

goLeft = true;

facing = "left";

pboPlayer.Image = Properties.Resources.left;//changes player image to left one

}

//right

if (e.KeyCode == Keys.Right)

{

goRight = true;

facing = "right";

pboPlayer.Image = Properties.Resources.right;//changes player image to right one

}

//up

if (e.KeyCode == Keys.Up)

{

goUp = true;

facing = "up";

pboPlayer.Image = Properties.Resources.up;//changes player image to up one

}

//down

if (e.KeyCode == Keys.Down)

{

goDown = true;

facing = "down";

pboPlayer.Image = Properties.Resources.down;//changes player image to down one

}

}

if (gameOver == true)

{

if (e.KeyCode == Keys.Enter)

{

RestartGame();

lblEnterToStart.Hide();

}

}

}

private void CollisionDetect()

{//loops through every control in the form

foreach (Control x in this.Controls)

{//if its a pbo and has the ammo tag

if (x is PictureBox && x.Tag == "ammo")

{//check if it is hitting player pbo

if (((PictureBox)x).Bounds.IntersectsWith(pboPlayer.Bounds))

{//when player picks it up, the pbo will be removed from program and add 5 to ammo

this.Controls.Remove((PictureBox)x);

((PictureBox)x).Dispose();

ammo += 5;

//Plays reloading sound

SoundPlayer Reload = new SoundPlayer(Zombie\_Survival.Properties.Resources.Reload\_Sound);

Reload.Play();

}

}

if ( x is PictureBox && x.Tag == "zombie")

{//if hits a zombie

if (((PictureBox)x).Bounds.IntersectsWith(pboPlayer.Bounds))

{//decreases health by 0.25

playerHealth -= 0.25;

}

}

foreach (Control j in this.Controls)

{//nested loop to differentiate zombie from player

if ((j is PictureBox && j.Tag == "bullet") && (x is PictureBox && x.Tag == "zombie"))

{

if (x.Bounds.IntersectsWith(j.Bounds))

{//if bullet hits zombie, increment score, remove bullet and zombie and add another zombie

score++;

this.Controls.Remove(j);

j.Dispose();

this.Controls.Remove(x);

x.Dispose();

makeZombies();

}

}

}

}

}

private void chasePlayer()

{//method for zombies to chase the player around the screen

foreach (Control x in this.Controls)

{

if (x is PictureBox && x.Tag == "zombie")

{//moves zombie toward player pbo

if (((PictureBox)x).Left > pboPlayer.Left)

{

((PictureBox)x).Left -= zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zleft;//make zombie face left

}

if (((PictureBox)x).Top > pboPlayer.Top)

{

((PictureBox)x).Top -= zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zup;//make zombie face left

}

if (((PictureBox)x).Left < pboPlayer.Left)

{

((PictureBox)x).Left += zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zright;//make zombie face left

}

if (((PictureBox)x).Top < pboPlayer.Top)

{

((PictureBox)x).Top += zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zdown;//make zombie face left

}

}

}

}

private void KeyIsUp(object sender, KeyEventArgs e)

{

//left

if (e.KeyCode == Keys.Left)

{

goLeft = false;

}

//right

if (e.KeyCode == Keys.Right)

{

goRight = false;

}

//up

if (e.KeyCode == Keys.Up)

{

goUp = false;

}

//down

if (e.KeyCode == Keys.Down)

{

goDown = false;

}

else

{

//pause game (1st)

//closes game (2nd)

if (e.KeyCode == Keys.Escape)

{

tmrGameTimer.Stop();

frmMenu pauseMenu = new frmMenu(Convert.ToString(score), false, true); //score, first time?, pausing?

pauseMenu.ShowDialog();

tmrGameTimer.Start();

//this.Close();

}

}

if (gameOver == false)//makes sure can't shoot when dead

{

//When space bar is released shoot a bullet

if (e.KeyCode == Keys.Space && ammo > 0)//checks if not out of ammo

{

SoundPlayer ShotSound = new SoundPlayer(Zombie\_Survival.Properties.Resources.Shot\_Sound);

ShotSound.Play();

ammo--;//decreases ammo

//Will shoot the bullet in the direction the player is facing

ShootBullet(facing);

if (ammo < 1)

{//when ammo is low an ammo box is dropped

DropAmmo();

}

}

}

}

private void DropAmmo()

{//will create an image of ammo

PictureBox ammo = new PictureBox();//creates instance of pbo

ammo.Image = Properties.Resources.ammo\_Image;//ammo immage assigned to the pbo

ammo.SizeMode = PictureBoxSizeMode.AutoSize;

ammo.Left = rnd.Next(10, this.ClientSize.Width - ammo.Width);//sets x to random value

ammo.Top = rnd.Next(80, this.ClientSize.Height - ammo.Height);//sets y to random value

ammo.Tag = "ammo";

this.Controls.Add(ammo);//adds it to the screen

ammo.BringToFront();

pboPlayer.BringToFront();

}

private void makeZombies()

{//when called it makes zombies

PictureBox zombie = new PictureBox();

zombie.Tag = "zombie";

zombie.Image = Properties.Resources.zdown;//default facing down

zombie.Left = rnd.Next(0, this.ClientSize.Width - zombie.Width);//sets x to random value

zombie.Top = rnd.Next(0, this.ClientSize.Height - zombie.Height - 80);//sets y to random value

zombie.SizeMode = PictureBoxSizeMode.AutoSize;

zombiesList.Add(zombie);//adds to list of zombies

this.Controls.Add(zombie);//adds the pbo to the screen

pboPlayer.BringToFront();

}

private void RestartGame()

{//used when game starts or is restarted

playerHealth = 100;

gameOver = false;

pboPlayer.Image = Properties.Resources.up;//makes sure player is facing up

ammo = 10;

score = 0;

tmrGameTimer.Start();

foreach (PictureBox i in zombiesList)

{//removes any zombies from previous game

this.Controls.Remove(i);

}

zombiesList.Clear();

for (int i = 0; i < 3; i++)

{//spawns 3 zombies on screen

makeZombies();

}

//makes player still

goUp = false;

goDown = false;

goLeft = false;

goRight = false;

}

private async void GameLoopEvent(object sender, EventArgs e)

{

if (playerHealth > 1)

{//asssings the progress bar to the playerHealth

prgHealthBar.Value = Convert.ToInt32(playerHealth);

}

else

{//when the player's health is below 1

//Play death sound

SoundPlayer DeathSound = new SoundPlayer(Zombie\_Survival.Properties.Resources.Player\_Death\_Sound);

DeathSound.Play();

pboPlayer.Image = Properties.Resources.dead;//player dead image shows

prgHealthBar.Value = 0;

gameOver = true;

tmrGameTimer.Stop();

await Task.Delay(3000);//Pauses entire program for 3 seconds. (Shows dead player and full death

frmMenu newMenu = new frmMenu(Convert.ToString(score),false, false);//sound can be played

this.Hide();

newMenu.ShowDialog();

this.Close();

}

lblAmmo.Text = "Ammo: " + ammo;//Updates ammo

lblKills.Text = "Kills " + score;//updates kill count

if (goLeft && pboPlayer.Left > 0)

{//moves player left

pboPlayer.Left -= playerSpeed;

}

if (goRight && pboPlayer.Left + pboPlayer.Width < this.ClientSize.Width)

{//moves player right

pboPlayer.Left += playerSpeed;

}

if (goUp && pboPlayer.Top > 100)

{//moves player up

pboPlayer.Top -= playerSpeed;

}

if (goDown && pboPlayer.Top + pboPlayer.Height < this.ClientSize.Height)

{//moves player down

pboPlayer.Top += playerSpeed;

}

CollisionDetect();

chasePlayer();

}

private void ShootBullet(string direction)

{//creates new bullet

//sets direction, x and y coordinate

if (direction == "right" || direction == "up")

{

Bullet shootBullet = new Bullet(direction,

pboPlayer.Left + (pboPlayer.Width - (pboPlayer.Width / 3)),

pboPlayer.Top + (pboPlayer.Height - (pboPlayer.Height / 3)));

shootBullet.MakeBullet(this);

}

if (direction == "down" || direction == "left")

{

Bullet shootBullet = new Bullet(direction,

pboPlayer.Left + (pboPlayer.Width / 3),

pboPlayer.Top + (pboPlayer.Height / 3));

shootBullet.MakeBullet(this);

}

//adds it to the form

}

}

class Bullet

{

//attributes

private string direction;

private int bulletLeft, bulletTop, speed = 20;

private PictureBox bullet = new PictureBox();

private Timer bulletTimer = new Timer();

//constructor

public Bullet(string direct, int left, int top)

{

direction = direct;

bulletLeft = left;

bulletTop = top;

}

//methods

public void MakeBullet(Form form)

{

//Creates picture box for bullet

bullet.BackColor = Color.White;

bullet.Size = new Size(5, 5);

bullet.Tag = "bullet";

bullet.Left = bulletLeft;

bullet.Top = bulletTop;

bullet.BringToFront();

//adds the created pbo to the form

form.Controls.Add(bullet);

//Uses timer to control bullet actions

bulletTimer.Interval = speed;

bulletTimer.Tick += new EventHandler(BulletTimerEvent);

bulletTimer.Start();

}

private void BulletTimerEvent(object sender, EventArgs e)

{//controls bullets timer

//control direction

if (direction == "left")

{

bullet.Left -= speed;

}

if (direction == "right")

{

bullet.Left += speed;

}

if (direction == "up")

{

bullet.Top -= speed;

}

if (direction == "down")

{

bullet.Top += speed;

}

//makes bullet dissapear when reaches edge

if (bullet.Left < 1 || bullet.Left > frmGame.ActiveForm.Width || bullet.Top < 1 || bullet.Top > frmGame.ActiveForm.Height)

{

bulletTimer.Stop();

bulletTimer.Dispose();

//removes from ram

bulletTimer = null;

bullet = null;

}

}

}

}

## Level 1

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing.Imaging;

using System.Media;

namespace Zombie\_Survival

{

public partial class lvlOne : Form

{

//creates variables/attributes

bool goUp, goDown, goLeft, goRight;

bool gameOver = false;

string facing = "up"; //used to guide bullets

double playerHealth = 100;

int playerSpeed = 10, ammo = 10, zombieSpeed = 4, score = 0;

bool droppedHealth = false;

TimeSpan timeleft1 = TimeSpan.FromMinutes(1);

TimeSpan timeleft2 = TimeSpan.FromMinutes(2);

TimeSpan timeleft3 = TimeSpan.FromMinutes(3);//Used for countdown

int WaveNo = 1;//keeps track of wave number

//Random variable can be used for dropping pickups

Random rnd = new Random();

//List of zombies to be added/removed

List<PictureBox> zombiesList = new List<PictureBox>();

private void lvlOne\_KeyDown(object sender, KeyEventArgs e)

{

//left

if (gameOver == false)//makes sure cant move when dead

{

if (e.KeyCode == Keys.Left)

{

goLeft = true;

facing = "left";

pboPlayer.Image = Properties.Resources.leftCampaign;//changes player image to left one

}

//right

if (e.KeyCode == Keys.Right)

{

goRight = true;

facing = "right";

pboPlayer.Image = Properties.Resources.rightCampaign;//changes player image to right one

}

//up

if (e.KeyCode == Keys.Up)

{

goUp = true;

facing = "up";

pboPlayer.Image = Properties.Resources.upCampaign;//changes player image to up one

}

//down

if (e.KeyCode == Keys.Down)

{

goDown = true;

facing = "down";

pboPlayer.Image = Properties.Resources.downCampaign;//changes player image to down one

}

}

if (gameOver == true)

{

if (e.KeyCode == Keys.Enter)

{

RestartGame();

lblEnterToStart.Hide();

}

}

}

//variables for resizing controls when going fullscreen

private Rectangle frmGameOriginal, lblKillsOriginal, lblHealthOriginal, lblAmmoOriginal,

prgHealthOriginal, pboPlayerOriginal, lblEnterToStartOriginal, pboLabelAmmoOriginal,

pboLabelKillsOriginal, pboLabelHealthOriginal, lblCountdownOriginal, lblWaveOriginal;

private void tmrCountdown\_Tick(object sender, EventArgs e)

{

if (WaveNo == 1)

{

timeleft1 = timeleft1.Subtract(TimeSpan.FromSeconds(1));

lblCountdown.Text = "Time left: " + timeleft1.ToString("mm\\:ss");

}

else if(WaveNo == 2)

{

timeleft2 = timeleft2.Subtract(TimeSpan.FromSeconds(1));

lblCountdown.Text = "Time left: " + timeleft2.ToString("mm\\:ss");

}

else if (WaveNo == 3)

{

timeleft3 = timeleft3.Subtract(TimeSpan.FromSeconds(1));

lblCountdown.Text = "Time left: " + timeleft3.ToString("mm\\:ss");

}

}

private void lvlOne\_Resize(object sender, EventArgs e)

{

resizeControl(lblEnterToStartOriginal, lblEnterToStart);

resizeControl(lblKillsOriginal, lblKills);

resizeControl(lblHealthOriginal, lblHealth); //This procedure will resize all of the controls on the screen

resizeControl(lblAmmoOriginal, lblAmmo); //when the user changes the size of the window

resizeControl(prgHealthOriginal, prgHealthBar);

resizeControl(pboPlayerOriginal, pboPlayer);

resizeControl(pboLabelAmmoOriginal, pboLabelAmmo);

resizeControl(pboLabelHealthOriginal, pboLabelHealth);

resizeControl(pboLabelKillsOriginal, pboLabelKills);

resizeControl(lblWaveOriginal, lblWave);//new control

resizeControl(lblCountdownOriginal, lblCountdown);//new control

}

private async void tmrGameTimer1\_Tick(object sender, EventArgs e)

{

if (playerHealth > 1)

{//asssings the progress bar to the playerHealth

prgHealthBar.Value = Convert.ToInt32(playerHealth);

if (playerHealth < 15 && droppedHealth == false)

{

DropHealth();

droppedHealth = true;

}

}

else

{//when the player's health is below 1

//Play death sound

SoundPlayer DeathSound = new SoundPlayer(Zombie\_Survival.Properties.Resources.Player\_Death\_Sound);

DeathSound.Play();

pboPlayer.Image = Properties.Resources.dead;//player dead image shows

prgHealthBar.Value = 0;

gameOver = true;

tmrGameTimer1.Stop();

await Task.Delay(3000);//Pauses entire program for 3 seconds. (Shows dead player and full death

frmMenu newMenu = new frmMenu(Convert.ToString(score), false, false);//sound can be played

this.Hide();

newMenu.ShowDialog();

this.Close();

}

lblAmmo.Text = "Ammo: " + ammo;//Updates ammo

lblKills.Text = "Kills " + score;//updates kill count

if (goLeft && pboPlayer.Left > 0)

{//moves player left

pboPlayer.Left -= playerSpeed;

}

if (goRight && pboPlayer.Left + pboPlayer.Width < this.ClientSize.Width)

{//moves player right

pboPlayer.Left += playerSpeed;

}

if (goUp && pboPlayer.Top > 100)

{//moves player up

pboPlayer.Top -= playerSpeed;

}

if (goDown && pboPlayer.Top + pboPlayer.Height < this.ClientSize.Height)

{//moves player down

pboPlayer.Top += playerSpeed;

}

CollisionDetect();

chasePlayer();

}

private void lvlOne\_KeyUp(object sender, KeyEventArgs e)

{

//left

if (e.KeyCode == Keys.Left)

{

goLeft = false;

}

//right

if (e.KeyCode == Keys.Right)

{

goRight = false;

}

//up

if (e.KeyCode == Keys.Up)

{

goUp = false;

}

//down

if (e.KeyCode == Keys.Down)

{

goDown = false;

}

else

{

//pause game (1st)

//closes game (2nd)

if (e.KeyCode == Keys.Escape)

{

tmrGameTimer1.Stop();

frmMenu pauseMenu = new frmMenu(Convert.ToString(score), false, true); //score, first time?, pausing?

pauseMenu.ShowDialog();

tmrGameTimer1.Start();

//this.Close();

}

}

if (gameOver == false)//makes sure can't shoot when dead

{

//When space bar is released shoot a bullet

if (e.KeyCode == Keys.Space && ammo > 0)//checks if not out of ammo

{

SoundPlayer ShotSound = new SoundPlayer(Zombie\_Survival.Properties.Resources.Shot\_Sound);

ShotSound.Play();

ammo--;//decreases ammo

//Will shoot the bullet in the direction the player is facing

ShootBullet(facing);

if (ammo < 1)

{//when ammo is low an ammo box is dropped

DropAmmo();

}

}

}

}

private void lvlOne\_Load(object sender, EventArgs e)

{

tmrGameTimer1.Stop();

tmrCountdown.Stop();//stops the game and countdown from running too

lblEnterToStartOriginal = new Rectangle(lblEnterToStart.Location.X, lblEnterToStart.Location.Y, lblEnterToStart.Width, lblEnterToStart.Height);

frmGameOriginal = new Rectangle(this.Location.X, this.Location.Y, this.Size.Width, this.Size.Height); ;

lblKillsOriginal = new Rectangle(lblKills.Location.X, lblKills.Location.Y, lblKills.Width, lblKills.Height);

lblHealthOriginal = new Rectangle(lblHealth.Location.X, lblHealth.Location.Y, lblHealth.Width, lblHealth.Height);

lblAmmoOriginal = new Rectangle(lblAmmo.Location.X, lblAmmo.Location.Y, lblAmmo.Width, lblAmmo.Height);

pboPlayerOriginal = new Rectangle(pboPlayer.Location.X, pboPlayer.Location.Y, pboPlayer.Width, pboPlayer.Height);

prgHealthOriginal = new Rectangle(prgHealthBar.Location.X, prgHealthBar.Location.Y, prgHealthBar.Width, prgHealthBar.Height);

pboLabelAmmoOriginal = new Rectangle(pboLabelAmmo.Location.X, pboLabelAmmo.Location.Y, pboLabelAmmo.Width, pboLabelAmmo.Height);

pboLabelHealthOriginal = new Rectangle(pboLabelHealth.Location.X, pboLabelHealth.Location.Y, pboLabelHealth.Width, pboLabelHealth.Height);

pboLabelKillsOriginal = new Rectangle(pboLabelKills.Location.X, pboLabelKills.Location.Y, pboLabelKills.Width, pboLabelKills.Height);

lblCountdownOriginal = new Rectangle(lblCountdown.Location.X, lblCountdown.Location.Y, lblCountdown.Width, lblCountdown.Height);

lblWaveOriginal = new Rectangle(lblWave.Location.X, lblWave.Location.Y, lblWave.Width, lblWave.Height);

//all above are saving all the control sizes and locations so that they can be changed when user resizes window

this.WindowState = FormWindowState.Maximized; //Auto set window size to maximised so that game is seen properly

gameOver = true; //stops game from running

lblEnterToStart.Show(); //Shows informational label so user can start by clicking enter

}

private void CollisionDetect()

{//loops through every control in the form

foreach (Control x in this.Controls)

{//if its a pbo and has the ammo tag

if (x is PictureBox && x.Tag == "ammo")

{//check if it is hitting player pbo

if (((PictureBox)x).Bounds.IntersectsWith(pboPlayer.Bounds))

{//when player picks it up, the pbo will be removed from program and add 5 to ammo

this.Controls.Remove((PictureBox)x);

((PictureBox)x).Dispose();

ammo += 5;

//Plays reloading sound

SoundPlayer Reload = new SoundPlayer(Zombie\_Survival.Properties.Resources.Reload\_Sound);

Reload.Play();

}

}

if (x is PictureBox && x.Tag == "health")

{//check if it is hitting player pbo

if (((PictureBox)x).Bounds.IntersectsWith(pboPlayer.Bounds))

{//when player picks it up, the pbo will be removed from program and add 5 to ammo

this.Controls.Remove((PictureBox)x);

((PictureBox)x).Dispose();

playerHealth += 25;//increases health

droppedHealth = false;//allows for another health drop to be spawned if needed

//Plays reloading sound

SoundPlayer GainedHealth = new SoundPlayer(Zombie\_Survival.Properties.Resources.Star\_Sound);

GainedHealth.Play();

}

}

if (x is PictureBox && x.Tag == "zombie")

{//if hits a zombie

if (((PictureBox)x).Bounds.IntersectsWith(pboPlayer.Bounds))

{//decreases health by 0.25

playerHealth -= 0.25;

}

}

foreach (Control j in this.Controls)

{//nested loop to differentiate zombie from player

if ((j is PictureBox && j.Tag == "bullet") && (x is PictureBox && x.Tag == "zombie"))

{

if (x.Bounds.IntersectsWith(j.Bounds))

{//if bullet hits zombie, increment score, remove bullet and zombie and add another zombie

score++;

this.Controls.Remove(j);

j.Dispose();

this.Controls.Remove(x);

x.Dispose();

makeZombies();

}

}

}

}

}

private void chasePlayer()

{//method for zombies to chase the player around the screen

foreach (Control x in this.Controls)

{

if (x is PictureBox && x.Tag == "zombie")

{//moves zombie toward player pbo

if (((PictureBox)x).Left > pboPlayer.Left)

{

((PictureBox)x).Left -= zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zleftCampaign;//make zombie face left

}

if (((PictureBox)x).Top > pboPlayer.Top)

{

((PictureBox)x).Top -= zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zupCampaign;//make zombie face left

}

if (((PictureBox)x).Left < pboPlayer.Left)

{

((PictureBox)x).Left += zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zrightCampaign;//make zombie face left

}

if (((PictureBox)x).Top < pboPlayer.Top)

{

((PictureBox)x).Top += zombieSpeed; //moves toward left of player

((PictureBox)x).Image = Properties.Resources.zdownCampaign;//make zombie face left

}

}

}

}

private void resizeControl(Rectangle r, Control c)

{

float xRatio = (float)(this.Width) / (float)(frmGameOriginal.Width);

float yRatio = (float)(this.Height) / (float)(frmGameOriginal.Height);

//Ratios of controls sizes ^

int newX = (int)(r.Location.X \* xRatio);

int NewY = (int)(r.Location.Y \* yRatio);

//Sets new location using ratio ^

int newWidth = (int)(r.Width \* xRatio);

int newHeight = (int)(r.Height \* yRatio);

//Sets new Size using ratio ^

c.Location = new Point(newX, NewY);

c.Size = new Size(newWidth, newHeight);

//completes the process ^

}

private void DropAmmo()

{//will create an image of ammo

PictureBox ammo = new PictureBox();//creates instance of pbo

ammo.Image = Properties.Resources.ammo\_Image;//ammo immage assigned to the pbo

ammo.SizeMode = PictureBoxSizeMode.AutoSize;

ammo.Left = rnd.Next(10, this.ClientSize.Width - ammo.Width);//sets x to random value

ammo.Top = rnd.Next(80, this.ClientSize.Height - ammo.Height);//sets y to random value

ammo.Tag = "ammo";

this.Controls.Add(ammo);//adds it to the screen

ammo.BringToFront();

pboPlayer.BringToFront();

}

private void DropHealth()

{

//will create an image of the health

PictureBox healthImage = new PictureBox();//Instance of pbo created

healthImage.Image = Properties.Resources.healthDrop;//ammo immage assigned to the pbo

healthImage.SizeMode = PictureBoxSizeMode.AutoSize;

healthImage.Left = rnd.Next(10, this.ClientSize.Width - healthImage.Width);//sets x to random value

healthImage.Top = rnd.Next(80, this.ClientSize.Height - healthImage.Height);//sets y to random value

healthImage.Tag = "health"; //identifier

this.Controls.Add(healthImage);//adds it to the screen

healthImage.BringToFront();

pboPlayer.BringToFront();

}

private void makeZombies()

{//when called it makes zombies

PictureBox zombie = new PictureBox();

zombie.Tag = "zombie";

zombie.Image = Properties.Resources.zdown;//default facing down

zombie.Left = rnd.Next(0, this.ClientSize.Width - zombie.Width);//sets x to random value

zombie.Top = rnd.Next(0, this.ClientSize.Height - zombie.Height - 80);//sets y to random value

zombie.SizeMode = PictureBoxSizeMode.AutoSize;

zombiesList.Add(zombie);//adds to list of zombies

this.Controls.Add(zombie);//adds the pbo to the screen

pboPlayer.BringToFront();

}

private void RestartGame()

{//used when game starts or is restarted

playerHealth = 100;

gameOver = false;

pboPlayer.Image = Properties.Resources.upCampaign;//makes sure player is facing up

ammo = 10;

score = 0;

tmrGameTimer1.Start();

tmrCountdown.Start();

foreach (PictureBox i in zombiesList)

{//removes any zombies from previous game

this.Controls.Remove(i);

}

zombiesList.Clear();

for (int i = 0; i < 3; i++)

{//spawns 3 zombies on screen

makeZombies();

}

//makes player still

goUp = false;

goDown = false;

goLeft = false;

goRight = false;

}

public lvlOne()

{

InitializeComponent();

RestartGame();

}

private void ShootBullet(string direction)

{//creates new bullet

//sets direction, x and y coordinate

if (direction == "right" || direction == "up")

{

Bullet shootBullet = new Bullet(direction,

pboPlayer.Left + (pboPlayer.Width - (pboPlayer.Width / 3)),

pboPlayer.Top + (pboPlayer.Height - (pboPlayer.Height / 3)));

shootBullet.MakeBullet(this);

}

if (direction == "down" || direction == "left")

{

Bullet shootBullet = new Bullet(direction,

pboPlayer.Left + (pboPlayer.Width / 3),

pboPlayer.Top + (pboPlayer.Height / 3));

shootBullet.MakeBullet(this);

}

//adds it to the form

}

private void lblEnterToStart\_Click(object sender, EventArgs e)

{

}

private void pboPlayer\_Click(object sender, EventArgs e)

{

}

class Zombie

{

//atributes

private int level, speed, health, damage;

private PictureBox image;

Random rnd = new Random();

//constructor

public Zombie(int setLevel)

{

if (setLevel == 1)

{

level = 1;

speed = 5;

health = 100;

damage = 5;

}

else if (setLevel == 2)

{

level = 2;

speed = 7;

health = 50;

damage = 15;

}

else if (setLevel == 3)

{

level = 3;

speed = 3;

health = 200;

damage = 40;

}

else if (setLevel == 4)

{

level = 4;

speed = 5;

health = 250;

damage = 25;

}

}

//methods

public void makeZombie(int Level)

{//when called it makes a zombie of corresponding level

Zombie zombie = new Zombie(Level);

zombie.image = new PictureBox();

zombie.image.Tag = "zombie";

zombie.image.Image = Zombie\_Survival.Properties.Resources.zdown; //default facing down

zombie.image.Left = rnd.Next(0, frmGame.ActiveForm.Width - zombie.image.Width);//sets x to random value

zombie.image.Top = rnd.Next(0, frmGame.ActiveForm.Height - zombie.image.Height - 80);//sets y to random value

zombie.image.SizeMode = PictureBoxSizeMode.AutoSize;

//zombiesList.Add(zombie);//adds to list of zombies

//this.Controls.Add(zombie);//adds the pbo to the screen

//pboPlayer.BringToFront();

}

}

}

}

# Post Development Testing

In this section I will undergo vigorous post development testing. Much of this testing will be carried out by stakeholders, this is so that a different perspective can be considered, and so that I can evaluate whether the requirements by the stakeholders have been met.

## Test Plan

I will use my original success criteria to create a test plan for post development testing. Each test will be numbered, and then cross referenced in the next section.

|  |  |  |  |
| --- | --- | --- | --- |
| No | Test Description | Test Data | Expected Result |
| 1 | User Input test – checking the reaction from the program when the user inputs something | Keyboard Keys  Mouse Clicks | An action occurs on the user’s display or movement of player sprite or changes in forms etc. |
| 2 | Enemy creation test – the program is observed, and events are triggered for creation of enemies by either eliminating current enemies, or just starting a new instance of gameplay | Game form  Sprites | Different images shown on screen in campaign. Multiple images at the same time on the screen |
| 3 | Updating of variables or databases – Running the game, completing actions such as eliminating enemies, or just idling. | Base variables (Countdown, kills ammo etc.) | The labels connected to variables in the program change when events occur, |
| 4 | Collisions test – Colliding with pickups or other sprites / objects | Sprites | Variables changed when contact with pickups. No overlaps with objects and enemies. |
| 5 | Code review test – Checking for number of pieces of codes repeated, how many classes used, how many procedures etc. | Program code | Its expected to see a minimal amount of code, through the use of subroutines. |

## Evidence of Testing for Criteria

1 entire test video will be used to cover the entire solution. Each criterion will be visited in the video. Here I will then evaluate the testing of each criterion – the occurrence of the feature implemented to meet the criterion will have a timestamp in the video, this will be written here next to it.

Video File Name:  
Ilyasse Boudjemline Coursework Test Video.mp4

The reference column contains the measurability factor of the criterion. The video timestamp for each feature is presented, this will allow the examiner to navigate to the part of the test video where the testing of the requirement in question is shown.

|  |  |  |
| --- | --- | --- |
| Requirement | Justification | Reference |
| The game will have a top-down, 2-dimensional view | This is the most preferable dimension and view for a player, and it is simple to implement | Functionality |

As shown in the testing video, the entire game is in the second dimension. Both the campaign and survival game modes implement the top-down view. Therefore, the requirement has been met.

|  |  |  |
| --- | --- | --- |
| There will be multiple enemies with a range of difficulties | The game needs an element that keeps the user interested, the enemies will become harder in difficulty, and some will be extremely hard | Functionality |

In the survival mode there is not a range of difficulties for the enemies. However multiple enemies are presented to the player. In the campaign mode, there are multiple enemies, that have different difficulties. Campaign game mode is run; different types of enemies should be spawned. This requirement has not been met. Test Number 2. [ 00:29 ]

|  |  |  |
| --- | --- | --- |
| There will be an ammunition system for the guns | This is a limiting factor that will keep the player “on their toes,” the ammunition will go down by 1 after every shot | Functionality |

Ammunition is depleted by continuously shooting – using the spacebar. The ammunition count decreases with each shot, the ammunition count is visible at the top of the screen. Philip (the ex-games developer) pointed out that the ammunition system is too basic, at least for the campaign mode. Reducing and refilling from 10 or 5 is said to be too simple, and repetitive. This requirement has therefore been met, but not entirely successfully because of its lack of uniqueness. Test Number 3, Test Number 1. [ 00:32 ]

|  |  |  |
| --- | --- | --- |
| There will be a health bar system for the player | This is another limiting factor that will make the user change how they play. The health of the player will decrease when an enemy hit them | Functionality |

This is tested in the video. The player sprite is moved towards the enemies. The health bar at the top of the screen does change – it decreases. When the health pickup is picked up, it increases. Different enemies will cause the health to decrease by different factors. This health system has been successful. Stakeholder Maimoon however says the health bar graphic itself is basic – it’s a simple green progress bar – possibly a different design would introduce some uniqueness to it. Test number 3. [ 00:51 ]

|  |  |  |
| --- | --- | --- |
| There will be a score system based on eliminations | This is to keep the game competitive and make the user feel a sense of achievement if they reach a high score | Functionality |

As tested, elimination of enemies causes the score value to increment. This is done my eliminating enemies; it works in both the game modes. However, it was pointed out by stakeholder Hayder during the testing that there was no excitement felt once a new score was reached. Something like an announcement of reaching a certain score would have fulfilled this. Requirement met, but no extra features implemented with it, which would otherwise make the game more immersive. Test Number 3.

|  |  |  |
| --- | --- | --- |
| A menu system for navigation between high score tables, pausing, restarting the game etc. | This is purely navigational and is there for the user to be able to access various parts of the game and complete system functions such as exiting | Usability |

This is explored at the start of the test video. It entirely works, the layout and design has been said to be relatively appealing for the stakeholders. Return button allows for returning to the main menu. Exiting the game can be done using an actual exit button rather than using a control command or manually closing the window. This exhibits one of the usability features. Menu testing is successful. Test Number 1. [ 00:04 onwards ]

|  |  |  |
| --- | --- | --- |
| Keyboard controls – Up left down right and spacebar for shooting | This is so that the user can easily control their character and kill the enemies. | Functionality |

Tested by me, this is a basic test - implements the use of test data – arrow keys are clicked, and held, and used at the same time to thoroughly test movement. The player sprite successfully moves across the screen, in the correct direction when the arrow keys are used. Shooting is also a basic test, holding and clicking spacebar. Bullets come out of the player sprite’s gun and move across the screen in a straight line in the direction that the sprite was facing when the spacebar was clicked. Maimoon says that there should be at least one gun with a different or extra control, for example a gun that required the spacebar to be held to “power up the gun”, then let go to release the bullet. Successful testing. Test Number 1.

|  |  |  |
| --- | --- | --- |
| A multi-level system | This is so that the user stays interested and keeps the game action filled. | Functionality - Usability |

This is implemented through the campaign game mode. Each level is unique to the other. they are accessible easily through the menu system. The difficulty intensifies with each level, and the stakeholders agree that 3 levels is a good number of levels for an at least immersive gaming experience. Not Met. Test Number 1 (navigating through menu and accessing different levels shows that this criterion has been met). [ 01:11 ]

|  |  |  |
| --- | --- | --- |
| Multiple weapons system | Gives the user a large variety of options to eliminate enemies | Functionality |

Test 4, with test 1 to move sprite towards gun pickups. The player sprite changes, to the corresponding image that has the player holding the new gun. Not met

|  |  |  |
| --- | --- | --- |
| Inventory system | Allows easy access for the user to navigate between weapons | Functionality |

This was not met. Different weapons can be picked up, but only one at a time.

|  |  |  |
| --- | --- | --- |
| Easy to call procedures | Procedures will be named appropriately so they can be used throughout the program for quick and efficient code | Robustness |

As seen in the code Listing, most of the code is just procedures subroutines and functions which are called throughout the program. This has meant that code is not repetitive, it’s also clean, and easier for me to further develop new features just by using prebuilt procedures. Test number 5

|  |  |  |
| --- | --- | --- |
| Flexible code | Code will be open to additions so that it can be improved easily, and new prototypes can be developed from the previous structure | Robustness |

Test number 5. Libraries have allowed to me to implement the same features through different campaign levels, and the survival mode. So, the code is flexible, it can be removed and added from without the need of a lot of edits. Proof of this is the fact that the stages throughout development had been very easily built onto each other, and existing code was just being amended and added to for new features to be developed.

|  |  |  |
| --- | --- | --- |
| Campaign game mode | This gives the user a choice, and reflects a different type of gameplay; gradual increase in difficulty and a story line to follow | Functionality |

Test 1 – clicking through menu to navigate to campaign mode. This criterion has been successful – there is a separate mode to survival just for campaign which has 3 levels. [ 00:04 ]

|  |  |  |
| --- | --- | --- |
| Survival game mode | Gives the user a choice to try a different type of gameplay; fast paced gameplay where the aim is to eliminate as many enemies as possible | Functionality |

Test 1 – Test 2. Successfully met, this was the base of the game originally, and allowed to me to further develop levels for the campaign mode. [ 00:28 ]

|  |  |  |
| --- | --- | --- |
| Design features for user usability such as window maximisation | Allows user to navigate the game with ease, also makes everything look clean and appealing. | Usability |

Test 1 – navigating the game forms and the menu. Return buttons between menus allow returning to previous screen without restarting the entire software. User can expand their game form without changing how it looks. Player can pause their game while playing. From there they can also access different levels and the main menu. [ 00:20 ]

|  |  |  |
| --- | --- | --- |
| User login / authentication system | This allows data for each user to be saved, such as their high score and which level they have reached | Functionality |

This criterion was not successful, this would have required a full online database to keep track of users and their passwords – however I simply used a local comma separated value file. This meant that mirrors between the databases of different people playing the game were not the same. Wherein multiple users use the same computer to play the game, which isn’t realistic, however it can be built on to integrate a full accounts system.

|  |  |  |
| --- | --- | --- |
| Smooth graphics | The movement of sprites in the game must not be glitchy, for a smooth user experience | Robustness |

Test 2, Test 4. No overlaps between any sprite or object, other than pickups. Enemies don’t overlap with the player, and the walls are physical objects that cannot be surpassed. The movement through initiating test 1 is also smooth – sprites move across the screen without it glitching or breaking up. While windows forms are not the best for graphical solutions, this criterion was somewhat met.

|  |  |  |
| --- | --- | --- |
| A high score system for each user | Allows users to compete with each other / aim for higher targets, will empower them to want to play the game more | Functionality - Usability |

This was bult with the local accounts system. Simply by assigning each account a high score value, and it being updated after a user logged in and then reaches a high score in survival. Test 3.

|  |  |  |
| --- | --- | --- |
| Sounds | Introduces extra immersion into the game for the user | Functionality |

Sound is available throughout the entire test video – clear sounds are played when events occur, or actions are made. There is also the background music. Successful, but could be more detailed (sounds of movement, sounds of enemy death etc.).

# Evaluation

## Success Criteria Evaluation

Here my original success criteria will be evaluated. The tests have been shown in the previous section, and it has been noted whether each was successfully met or not successfully met. The evaluation of each criterion explains in detail each requirement.

|  |  |  |
| --- | --- | --- |
| Requirement | Justification | Reference |
| The game will have a top-down, 2-dimensional view | This is the most preferable dimension and view for a player, and it is simple to implement | Proposed solution |
| There will be multiple enemies with a range of difficulties | The game needs an element that keeps the user interested, the enemies will become harder in difficulty, and some will be extremely hard | Proposed solution |
| There will be an ammunition system for the guns | This is a limiting factor that will keep the player “on their toes,” the ammunition will go down by 1 after every shot | Proposed solution |
| There will be a health bar system for the player | This is another limiting factor that will make the user change how they play. The health of the player will decrease when an enemy hit them | Proposed solution |
| There will be a score system based on eliminations | This is to keep the game competitive and make the user feel a sense of achievement if they reach a high score | Proposed solution |
| A menu system for navigation between high score tables, pausing, restarting the game etc. | This is purely navigational and is there for the user to be able to access various parts of the game and complete system functions such as exiting | Proposed solution |
| Keyboard controls – Up left down right and spacebar for shooting | This is so that the user can easily control their character and kill the enemies. | Proposed solution |
| A multi-level system | This is so that the user stays interested and keeps the game action filled. | Proposed Solution |
| Multiple weapons system | Gives the user a large variety of options to eliminate enemies | Proposed Solution |
| Inventory system | Allows easy access for the user to navigate between weapons | Proposed Solution |
| Easy to call procedures | Procedures will be named appropriately so they can be used throughout the program for quick and efficient code | Development |
| Flexible code | Code will be open to additions so that it can be improved easily, and new prototypes can be developed from the previous structure | Development |
| Campaign game mode | This gives the user a choice, and reflects a different type of gameplay; gradual increase in difficulty and a story line to follow | Proposed solution |
| Survival game mode | Gives the user a choice to try a different type of gameplay; fast paced gameplay where the aim is to eliminate as many enemies as possible | Proposed solution |
| Design features for user usability such as window maximisation | Allows user to navigate the game with ease, also makes everything look clean and appealing. | Development / proposed solution |
| User login / authentication system | This allows data for each user to be saved, such as their high score and which level they have reached | Development / proposed solution |
| Smooth graphics | The movement of sprites in the game must not be glitchy, for a smooth user experience | Proposed solution |
| A high score system for each user | Allows users to compete with each other / aim for higher targets, will empower them to want to play the game more | Proposed solution |
| Sounds | Introduces extra immersion into the game for the user | Proposed Solution |

In the end, the project is not fully completed. The solution that is left is not suitable, but the original was.

## Usability

Visualisation with the health progress bar, rather than just a number indicating health left is a usability feature. The pausing function is a usability feature allowing ease of use for the user. The menu system is a usability feature, within it, return buttons, a sub-feature allowing users to navigate with ease. An exit button for the game, rather than having to close the game manually is a usability feature. A log in, and account registration page in the menu is a usability feature allowing for easy entrance and saving of statistics. There is unfortunately no “log out” button – which would hypothetically take the user back to the login / registration page.

## Maintenance

The game requires the maintenance of login details for each user – it needs to be updated within each user’s game files. This means the project needs to be exported again and again whenever a new user registers, and have it sent to all the users, and expect them to use this new version – this is not ideal at all and is also a limitation.

Otherwise, the game in its current state does not really require much maintenance, unless it were updated to include new features such as a server-based system for the game.

## Limitations

As stated before, login system is locally saved for each user, meaning that if someone were to want to play the game on another device, with the same account, they would have to export their data to the new device’s game files.

## Future Improvements

Log out button, takes to user login / registration page. A server-based accounts system, rather than local, will allow users to log into and register accounts using different versions of the game, and removes the need for having to update each user’s game files to account for the new accounts. Also allows for playing the game with personalised statistics on multiple devices. This could be implemented either through an online repository system which is accessed by the games master server which would also have to be implemented. This could then be further expanded to implement online play, and remote application updates.   
  
Another way it could be implemented is a simple SQL server system, a database on an external server storing all the users’ information. This could also be further expanded to allow for the saving of customisable settings for the game (sprites, sound levels) – to be saved and ported upon running of the game / logging in (also requires a master server for the game).