|  |
| --- |
| Name: Laura Gandia |
| Computer science project |
| H446-03 |
| Candidate number:  School name: Horsforth School  Centre ID: 37625  Title project |

Contents

[**Analysis 2**](#_h5adh1sdoeuj)

[Problem identification 2](#_qxhas8v0opmw)

[What is the problem? 2](#_nj48h21ngnwe)

[- Decomposition 3](#_jm1odff3n6ou)

[- Abstraction 4](#_fd3h18r1iljw)

[- Trial and error: 0](#_lvgoanbei1km)

[- Thinking ahead 0](#_5ezyj94g00n8)

[- Thinking concurrently 0](#_w1v70gk7xhb7)

[- Thinking logically 0](#_xbgpx9jc42wm)

[- Thinking procedurally 0](#_ku6ccorpulmk)

[Stakeholders 0](#_sfhmqc9wlz46)

[Research the problem 0](#_ovyze3rsykd5)

[Researching Geometry Dash 0](#_2pojyph5n7fn)

[Description of Geometry dash 0](#_vjsjjccf8ciz)

[Researching Google Dinosaur 0](#_7j16uds5ccv9)

[Description of Google dinosaur 0](#_ao2t7owjsja6)

[Researching Vanguard (Atari 2600) 0](#_r473r325ps7i)

[Description of Vanguard (Atari 2600) 0](#_l3telis7rpgv)

[Interview 0](#_wga5hkucgkms)

[Specify the proposed solution 0](#_b7sipam5f1r6)

[Aesthetics 0](#_6q6xl0qsu2u0)

[Future success criteria 0](#_qtecpap4wmg)

[**Design 0**](#_y6sc78l0c9eb)

[Decompose the problem 0](#_azva7rytm12x)

[Describe the solution 0](#_3rkxe7hpocid)

[Systems diagrams 0](#_8511h7q3l8ky)

[Game flowchart 0](#_uqs6yac9adkg)

[Game state diagram 0](#_fe1dd5lb1dc4)

[Class diagram 0](#_gjjka67thhia)

[Describe the approach to testing 0](#_uxm3xzgtemai)

[Prototyping and testing approach 0](#_srei9ixk4pat)

[Prototypes plan 0](#_osqev3mfsiwr)

[Overall testing required 0](#_9g3pay8jw3q6)

[**Developing the coded solutions 0**](#_ryzx7atms73p)

[Prototyping 0](#_av1zhkgtgsdv)

[Overall testing and developing 0](#_bvnno4o1bvnu)

[Final aesthetic design 0](#_a72giad47xfb)

[Testing for evaluation 0](#_lwygn7jzljc2)

[User acceptance testing 1 0](#_njz7d448wqau)

[User acceptance testing 2 0](#_lm8tpdkaheqa)

[User acceptance testing 3 0](#_agl6ebjgrefc)

[Improvements for usability 0](#_x56h7w1etuu7)

[Code and files 0](#_fb0nvdrqyfdp)

[Game files 0](#_4nax7w9b86g4)

[File inside ‘assets’ 0](#_73jgvt9hidku)

[Files inside ‘graphics’ 0](#_2b281ly7n0uf)

[Assets code 0](#_euqbf4jzj3lt)

[Constants code 0](#_8huowzfx0ntl)

[Sprites code 0](#_glax28sb3eh)

[Game code (test\_07) 0](#_37fxc7kl66th)

[**Evaluation 0**](#_qw3d0wl872e0)

[Success criteria review 0](#_bc3ynbhygtf)

[White box interview (test 13) 0](#_w8dze7tqkdut)

[Black box interview (test 14) 0](#_17bwubb3x4mj)

[Usability review 0](#_qhimoa2cim37)

[Overall review 0](#_d9la2w7focs4)

[Future potential features 0](#_jtmrxkrhm2xg)

[**Project appendixes 0**](#_4e9dom9oia40)

# **Analysis**

## Problem identification

### What is the problem?

For this project, I am making an obstacle runner game, where the character must move at a certain speed through an obstacle course, hitting an object will result in the death of the character and making the level restart while some objects may make them bounce, or help the player jump through the level.

In the following games that will be researched, there are also portals or gates throughout the obstacle course which offer new types of game modes within the map switching through the level (such as making the level be upside down or the character run from right to left instead of left to right, placing the character in a spaceship in a flappy bird style of gaming).

The game would need to have the following: an accuracy indicator, a progress bar to point how long of the map is left or a highscore indicator in the case that the game is infinite; appearing obstacles; and the player’s characters to control. Other additional things that I could add would be some type of item which could be collected throughout levels from which the player could trade for different aesthetic items, maps or power ups.

The following methods can be used to solve the current problem as each of them could take part in the different sections of the game.

As mentioned before, the game would need to have the following:

* an accuracy indicator,
* a progress bar to point how long of the map is left/high score counter
* appearing obstacles and the player’s characters to control

Other additional things that could be added would be some type of item which could be collected throughout levels from which the player could trade for different aesthetic items, maps or power ups.

**This project is solvable by computational methods such as[[1]](#footnote-1):**

#### **Decomposition**

* + What are the different parts of the problem you are trying to solve?
  + What are the main sections/parts of the problem you are solving?
  + How could this problem be divided into smaller parts?

Decomposition can be used to break down a problem into smaller parts, this can help make this project easier to understand and carry out, making smaller goals to know where to start and getting smaller goals to get things done as big projects may be overwhelming and become tedious. By breaking down the problem it is easier to know which parts must be done so they can come together and work together to create a program.

**Why this problem is amenable to this computational approach**

Decomposition in this game to more easily develop objects and gameplay scenarios more easily in this game. Decomposition can be used to break objects down within the game, as each object has a class and certain attributes and criteria it must meet for the game to work, for example an obstacle should have a hitbox, a height/width and if it moves or not, and the parameters of where it moves.

* **Pattern recognition**
  + Are there any patterns you can observe?
  + Do you notice any similarities between this problem and something else you have already solved?
  + Do any of the parts of this problem share qualities?
  + Does anything repeat?

Pattern recognition could be used to recognise objects and features which may have similar properties to each other which may overlap with each other. This means that to make work simpler when developing this project, certain objects could be replicated and certain properties be added, removed or changed to create a whole different object or entity.

Pattern recognition can also be used to create objects and functions and be able to call these functions and objects when needed instead of having to create new ones every time, shortening the code and making it easier to maintain or change as you would not have to individually change every single object in the game and just edit the template.

**Why this problem is amenable to this computational approach**

In this particular project, pattern recognition may be used to more easily create objects and replicate them through the obstacle course and create new ones as they are fairly similar.

An example of this could be a box that could instead of being an obstacle, it could be jumped on safely. Since many of these games are rhythm based, patterns in the music used could fit a certain series of obstacles in the course creating a pattern along with the music, following the beat and making it more satisfying to play.

* **Thinking abstractly**
  + What is abstraction?
  + How can this problem be abstracted?

#### **Abstraction**

* + What are you trying to solve?
  + Which details are important in solving this problem?
  + What can you leave out? What information is unnecessary?
  + Can you describe the problem as something more basic?

Abstraction is the process of filtering out the characteristics of problems that are not needed in order to concentrate on those that are needed[[2]](#footnote-2).Abstraction can be used to highlight what you are actually doing.

Big projects may feel overwhelming with many different details and parts, however, by abstracting the problem you might be able to figure out what the most important parts of the problem are and be able to simplify the problem into less complicated problems, allowing you to find a simpler and easier solution to the problem. Combining abstraction with decomposition in a project will help to break down problems and make them simpler, therefore making it easier to solve and possibly setting deadlines to fix the problem.

**Abstracting “Space run”**

I need to figure out what features are essential for the game to work, this is where abstraction comes in.

* 2 dimension gaming
* Add obstacles to the course
* Add a timer/progress bar
* Highscore counter
* Add a retro style for simplicity
* Add a menu for the user to navigate through different levels.
* Have a character that has to be used to progress through the game

**Why is this problem amenable to this computational approach?**

In this particular project, abstraction may be used to highlight key features of the game, and rule out less essential elements of the game. All of this makes it easier to develop a basic outline of the project to later be improved, as if you start to work on the wrong things of the game such as the aesthetic design rather than the functionality you may encounter time constraints as the game might look good but not be playable by the time the deadline arrives.

Examples of essential elements of this game that have been obtained through abstraction could be controls of the player, map, possible obstacles that the player must work through and the character movement and controls.

* **Algorithms** 
  + What’s the first step you can take in solving this problem?
  + What are the steps that you need to do to solve this problem?
  + In what order should you complete those steps?

An algorithm is[[3]](#footnote-3) a process or set of rules to be followed in calculations or other problem solving operations. Algorithms and algorithmic thinking can be paired together with thinking procedurally as algorithms tend to have procedural thinking aspects as they must be carried out in a certain order and depending on particular conditions, in algorithms this is known as selection. To be able to develop a coded solution you must be able to understand algorithms. This is because the way that computers work is defined by algorithms as they must be able to follow precise and detailed instructions, as computers will not understand generic instructions.

**Why this problem is amenable to this computational approach**

This problem must be solved using algorithms as the solution to this problem can be solved through algorithms in a coded program. Examples of how algorithms could be used in this problem, would be that the object to be used in the game to be controlled by the player would be coded along with its attributes in object oriented programming and algorithms would be used to dictate how the object interacts with other objects and how it is controlled by the player. Another example would be that algorithms would be used to code the artificial intelligence of the enemies and other objects that may be in the game; such as the menu.

#### **Trial and error:**

* + Does this work? Why yes or why not?
  + Is there anything you could change for this to work?

Trial and error[[4]](#footnote-4) is finding out the best way to reach a desired result or a correct solution by trying out one or more ways or means and by noting and eliminating errors or causes of failure. If something is not working, it is worth using trial and error to figure out what is working and what is not working, this helps you improve the solution to the problem you are trying to solve.

Trial and error may take time, and not be very efficient, however if there is a problem you might not be able to solve, trial and error is a good technique to find the source of the problem and tracing algorithms to find where the errors take place.

An example of this would be running a code with different values to see if it validates the right values that should be allowed and rejects the values that should not be allowed by the program, e.g. “YEADON1” is inputted by the user as the month of the year, which should be rejected by the program and would proceed to prompt the user to input a valid month. All of this trying to make the program fail is part of the trial and error to ensure a code is working properly.

**Why this problem is amenable to this computational approach**

This problem is amenable to trial and error to make sure you can calibrate different features of the game such as the starting positioning of the character, collision with objects and what controls may need to be used for the solution to work. An example of this for this particular solution to the problem would be to test that the collision detection works and ends the game if the user crashes into an object they should have not, the code could also be run multiple times to test all the objects are generated properly and the user does not lose the game by going on objects they should not be dying to or goes through the objects completely.

#### **Thinking ahead**

* + What factors will you have to consider for this project?
  + What problems might you run into?
  + How will you solve these problems?
  + After completing a section, what will be your next goal?
  + How will different sections come together to make a final project?

Thinking ahead can be paired with thinking concurrently and thinking procedurally. As you need to plan how you will complete the project which is typically using combined techniques of computational thinking, breaking down the problem into simple small phases and completing one after the other until everything is put together to solve the problem. By thinking ahead and concurrently, you are considering problems you might have and figure out a set of solutions that will fix the problem so the development of the solution can be finished.

**Why this problem is amenable to this computational approach**

Thinking ahead in this problem has to be used to be able to understand and know how different objects within the game/solution will work and interact with each other, this could be that the player dies to a certain obstacle but on the other hand, a different object they can jump on top. Enemies may be designed to get in the way of the player and go after them, posing a larger threat than a normal obstacle. Lives or second chances may be put in place. All this thinking ahead will allow the programmer to more easily code the game as everything that needs linking together can be easily coded to ensure everything works as it has been planned and that most if not all factors have been taken into account.

#### **Thinking concurrently[[5]](#footnote-5)**

Thinking concurrently means to think about things at the same time. This helps you to solve problems faster and relate them to each other, all components of a large problem will have similarities and differences. Similarly to thinking ahead, thinking concurrently has to be used to be able to understand, and predict how different objects within the game/solution will work and interact with each other as you are thinking about different elements of the solution at the same time.

**Why this problem is amenable to this computational approach**

This problem requires us to use thinking concurrently, this is because we need to think about many different elements simultaneously. Examples of this would be that the objects that we are using to create the game must interact with each other; such as an interface for the user to view, like a background, and an entity that the player must control. These two elements have to work together as the player’s entity must or should not be able to leave the viewable interface of the game; as the player would not be able to see it as it would be off screen and not being easily able to control.

#### **Thinking logically**

Thinking logically is used to solve problems that need to be solved using algorithms or procedures, this could also involve problems that must be solved using a sequence. This could be solving a maths problem that has many parts or carrying out a task that needs to be done in a set of certain order. Thinking logically allows us to improve the efficiency of our solution as we would think about what are the most logical steps to solve the problem, and how doing these steps could improve the efficiency of our problem solving. Computers will use code to function, but they only work logically and binary, making strictly ruled out decisions based on the data that has been given. Computers also work using binary, which is boolean as it is a yes or no system; this means that no other answers can be produced to be interpreted by the computer and outcomes are limited.

**Why this problem is amenable to this computational approach**

This problem needs to be solved using logical thinking because logic must be used when using a coded solution to solve a problem due to the limitations of coded solutions. Examples of this would be to make decisions on whether the player’s character has collided with an object, and depending on the object, react accordingly to what the object is, if it's an obstacle the player would lose and the game would reset, or it would lose a life. Logical procedures and thinking must be used also when taking into account how different objects interact with the game and the overall experience.

#### **Thinking procedurally**

Thinking procedurally means to think about a solution sequentially, thinking about what operations must be performed one after the other, being able to hierarchically as to how operations should take place and which ones will have to be called multiple times. Thinking procedurally allows you to be able to see the world from a computer like perspective, as actions must be performed in a certain order with certain requirements to achieve the desired result.

**Why this problem is amenable to this computational approach**

In this game, thinking procedurally is important because the way solutions are coded generally require to be procedural, as computers work this way, using explicit instructions to work. Examples in this solution that requires thinking procedurally would be how from the menu the game is started and what settings the game starts with as it gradually gets faster and harder, thinking procedurally allows you to apply logic to the coding of the game as it would not function properly or be chaotic if the start menu played after the players starts but before the player dies, interrupting the game.

## 

## Stakeholders

This project will have a very simplistic, cartoonish look. It will also most likely seem pretty retro and pixelated, making it as simple as possible and no speech or text, other than from the tutorial that would be present in the game, meaning it would make it easily understandable by everyone.

The nature of obstacle racing games is that they are very easily understandable by everyone regardless of language, ethnicity, culture or age. They also tend to have simple ways of movement and controls. All of this makes it easy to play, however as speed of the game increases and more objects are added the person must have fast reaction times and use trial and error many times to memorise the level to learn what obstacles are tricky.

However, the game will be mostly aimed at a younger audience such as teenagers and children, due to the retro and cartoonish aesthetic, with vibrant colours it will be given to catch the eyes of children and teenagers, rather than adults. This is because teenagers and children will most likely become more easily engaged with a runner and obstacle game rather than an adult, as the music to be used most likely be dubstep/hip hop themed music with strong beats, in contrast to the more considered adult music such as classical or overall calmer music.

Companies could also benefit since advertisements could be included into the game, this could catch the attention of certain users of the game who might be interested in the company’s product or service being advertised, therefore potentially creating a revenue for them.

## Research the problem

## Researching Geometry Dash

##### Description of Geometry dash

“Geometry dash” is an action rhythm-based game. In this game you can control a block and make it jump up (and sometimes down) by tapping once or clicking.

The block must move at a certain speed through an obstacle course, hitting an object will result in the death of the block and making the level restart while some objects may make it bounce through the level.

There are also portals or gates throughout the obstacle course which offer new types of game modes within the map switching through the level (such as making the level be upside down or the character run from right to left instead of left to right, placing the character in a spaceship in a flappy bird style of gaming).

| **Main features of Geometry Dash** | | | |
| --- | --- | --- | --- |
| **Features** | **Feature included?** | **Purpose of feature** | **Why has it (not) been included?** |
| **In game** | | | |
| Player character |  | It allows the player to control a character within the game to sort through the obstacles | Due to the nature of the game i am developing it is useful to have a character that must be controlled, a third person perspective also allows the player to have a more detailed sense of distance and space to be able to calculate where collisions may occur and how far the character can move and jump without colliding with any objects |
| obstacles |  | Gives the game a purpose. Makes the player have to jump through obstacles and loose if the block hits an obstacle, making them reset the game | Obstacles are one of the main features of a runner-obstacle game, as the whole purpose of the game is to go through the level dodging and jump the obstacles in the level. |
| Background music |  | Give more ambience to the game as well as letting the user know when to click as obstacles are placed in a manner that matches with the beat of the music. | The game will not be rhythm based, however background music will be added for ambience, enhancing the experience of the user as they could get more absorbed into the game. |
| Decorative elements |  | Similarly to background music, it adds ambience to the whole environment and energy of the game, allowing the user to go deeper into the game experience. | Decorative elements are not particularly essential to a game, these may be added to the game to enable the user to have a better experience of the game. However, since the initial solution of the problem is highly abstracted initially to start developing the solution more easily, things such as decorative elements are not considered to be added until later on in the development of the solution |
| Pause menu |  | Lets the user pause the game in case the user must leave the game for a few minutes to do something. It also allows the user to access settings to change music volume or change level. | This feature is not essential, however it could be added in a later date as it is important for a user to be able to leave or pause the game, as well as let them change the settings if they do not feel comfortable with the default ones. |
| Portal that changes gamemode |  | Used to add more interesting elements to the game, making it less repetitive and harder to play, adding more game mechanics | This feature is not to be included as the solution to be planned is meant to have as simple mechanics as possible |
| Jumpable objects |  | Similarly to obstacles, they give purpose and function to the game as the player must use these objects to sort through the game, they might also serve as both an obstacle and a jumpable object, depending how the player collides with the object. | Similarly to obstacles, jumpable objects are one of the main features of a runner-obstacle game, as the whole purpose of the game is to go through the level dodging and jump the obstacles in the level.However, the game will run in open space therefore there it would fly, not jump. |
| Phantom jumpable objects |  | Objects that are used to jump through the obstacle course, however, they can only be jumped once. In geom | Initially there is not a plan to put objects which can only be jumped once, however in the future it would be a good addition for the game to be included as it would increase the amount of skill required to advance into the game, making it harder and more entertaining as the game goes on. |
| **Main menu** | | | |
| Sound effects |  | Add ambience to the actions of the player, and helps indicate when an action has been performed, such as jumping or going through a portal, as well as dying. | Sound effects would be used in the game to make the game more entertaining to play and allow the player to feel more immersed into the experience. |
| Level editor |  | Allows users to create their own levels with their own music and obstacle course. Helps innovation and essentially, once the community of the game is large enough, it ends up doing the work of the game developers for them as they are developing the levels and not the company. Saving time and effort. | This game will not have an option to edit levels as it will be an infinite runner, where random objects will be generated so the player can have an unique experience each time they play. However as the score increases new obstacles will be added or made harder to make the game overall harder. |
| Character select |  | Allows the user to choose a different character they control. These are unblocked using in game currency, playing levels and unlocking achievements. These characters are customisable and each has a rarity level depending on how hard they are to obtain. | For now, only one character will be able to played, however there may be the option in the future to be able to use in game currency to purchase new characters or be able to edit them |
| External links |  | The app offers links to external apps, this could be to share the game in other social medias, or go to the official channel and website of the game. | It is important to promote the app and give the users a way to stay updated about the current game and future games to be released. Links to websites such as instagram or twitter would be included. |
| Settings menu |  | Similarly to the in-game pause menu, it allows the user to edit settings of the game such as music volume. | This feature would be included as it is included in all games, it is important for a user to be able to edit the settings of the game and adjust them to their liking. |
| **Level selection menu** | | | |
| Level selection menu |  | Allows the user to choose what level they wish to play. For each level, the menu displays how far alone the user has managed to pass the level, if they have finished it and how many points they have scored/ stars collected. | The game will not have levels as it will start simple and get progressively harder. |
| Difficulty of level indicator |  | Shows the user how hard the level they are playing will be rated. | The game will not have levels as it will start simple and get progressively harder |

#### 

#### Researching Google Dinosaur

##### Description of Google dinosaur

It is a browser game that was developed by Google and placed as a secret within the Google Chrome browser. This game would be activated and available when there was no connection to the internet. Gameplay wise, the player controls a dinosaur through a side scrolling landscape, and must jump or duck through obstacles to not die and obtain a higher score.

| **Main features of Geometry Dash** | | | |
| --- | --- | --- | --- |
| **Features** | **Features included?** | **Purpose of feature** | **Why has it (not) been included?** |
| **In game** | | | |
| Player character |  | It allows the player to control a character within the game to sort through the obstacles. A sound is played each time the dinosaur jumps and each time it hops over an obstacle | Due to the nature of the game i am developing it is useful to have a character that must be controlled, a third person perspective also allows the player to have a more detailed sense of distance and space to be able to calculate where collisions may occur and how far the character can move and jump without colliding with any objects |
| Obstacles |  | Gives the game a purpose. Makes the player have to jump through obstacles and loose if the block hits an obstacle, making them reset the game | Obstacles are one of the main features of obstacle runner games, which is the proposed solution being made |
| Score displayer |  | Indicates to the player how many points the player has scored so far. Once a certain score is reached, a sound is played indicating to the player that milestone has been hit. | Shows the user how far they have progressed through the game and how many points they have picked up throughout the way. |
| Past high score display |  | Displays the previous highest score achieved in the game | It shows the user what their previous high score is, this encourages the player to improve their skills and keep playing to get a better high score than before, competing against their past self.  However this feature will not be added since records of the scores will not be kept and only displayed at the end, as the game is designed to be a fresh game each time it is initialised. |
| Decorative elements |  | It adds ambience to the whole environment and energy of the game, allowing the user to go deeper into the game experience. | Decorative elements are not particularly essential to a game, these may be added to the game to enable the user to have a better experience of the game. However, since the initial solution of the problem is highly abstracted initially to start developing the solution more easily, things such as decorative elements are not considered to be added until later on in the development of the solution |
| **Game over screen** | | | |
| Play again button |  | Indicates the player has died and prompts the user to play again and have another go. | Similarly to the past high score display, it encourages the user to play again |

#### 

#### Researching Vanguard (Atari 2600)

#### 



Character that the player controls

enemies

Obstacles

Obstacles

lives

Score display



Character that the player controls

enemies

Obstacles

Obstacles

lives

Score display

Projectiles character shoots

Pickable powerups

##### Description of Vanguard (Atari 2600)

Vanguard from Atari 2600[[6]](#footnote-6) is a scrolling shooter arcade video game that was developed by TOSE. It was initially released in 198. This game has a somewhat a plot where “The Gond has been terrorising nearby space colonies with its periodic raids of destruction”. This game includes different regions as levels which takes the player on a journey to defeat the Gond with their ship. Gameplay wise, the player controls a ship which automatically flies forward through the scrolling levels with a limited amount of fuel that decreases as the game goes on, however this fuel can be recharged by destroying enemies with your ship. The orientation of the map changes from horizontal to vertical to diagonal depending on the level, and there are power ups called ‘energy pods’ which make your ship immune to enemies and hits for a limited amount of time, allowing the player to destroy walls and enemies by running into them.

| **Vanguard (Atari 2600)** | | | |
| --- | --- | --- | --- |
| **Features** | **Features included?** | **Purpose of feature** | **Why has it (not) been included?** |
| **In game** | | | |
| Player character |  | It allows the player to control a character within the game to sort through the obstacles. A sound is played when either the ship crashes or the player loses a life | Due to the nature of the game i am developing it is useful to have a character that must be controlled, a third person perspective also allows the player to have a more detailed sense of distance and space to be able to calculate where collisions may occur and how far the character can move and jump without colliding with any objects |
| Enemies |  | Serve as obstacles, players must shoot projectiles to destroy them or dodge them. If the enemies touch the character, then the player will lose a life. | Enemies will be one of the main features of the game along with obstacles. As the player must fight with them while progressing through the game |
| Powerups |  | Allows the player to have special powers for a limited amount of time, or an upgrade until they are hit. Extra lives may also be collected so that players are not eliminated immediately after making a mistake | Powerups will not be included in the initial prototype of the game, however they could be implemented later on to add more interesting features. |
| Proyectiles / misiles |  | It is shot by the player to destroy obstacles and enemies. | This will be implemented in the game to add more mechanical skill to the game and strategy to get through stages. |
| Background music |  | Give more ambience to the game as well as letting the user know when to click as obstacles are placed in a manner that matches with the beat of the music. | The game will not be rhythm based, however background music will be added for ambience, enhancing the experience of the user as they could get more absorbed into the game. |
| Sound effects |  | Add ambience to the actions of the player, and helps indicate when an action has been performed, such as jumping or going through a portal, as well as dying. | Sound effects would be used in the game to make the game more entertaining to play and allow the player to feel more immersed into the experience. |
| Obstacles |  | It makes the game harder and stops the player from sticking to the ceiling or ground of the game where there are less enemies likely to spawn | It makes the game harder and has more mechanics, making it more dynamic and entertaining. |

## 

## Interview

### Have you played a side scrolled shooter game?

A survey was conducted with multiple different users so that the proposed solution could be drafted out from the wants and needs of the audience. The results ranged from users of the ages of 15 - 18, as it is the main target audience of the game.

### What set of controls would you prefer to play?

From the users interviewed, about 68% of the users claim that they have played a side scroller game, which shows that these types of games are quite popular amongst young audiences, and so it is possible that the potential solution can draw a large amount of users in to play it.

From the users interviewed, again, 68% said they preferred to play with WASD keys and F key to shoot. Therefore, the game will be designed to work with said keys, however another fairly large amount of users claimed that they preferred to play with the arrow keys and press Ctrl to shoot, so we will implement both controls to satisfy about 89% of those interviewed as the those who wished to use a mouse to control the game where a really small minority.

### Do you prefer faster or slower paced games?

From the users interviewed, 55.6% mentioned that they like faster paced games, while the remaining 44.4% claimed that they like to start slow and build up speed. Therefore we decided to start the game at a fair but not slow pace, to satisfy all users as they would soon be found to have a high speed game within seconds of starting the game.

### Do you like that games get harder as you progress?

From the users interviewed, about 89% percent of users said that they like to play games that get harder as they progress through the game. This reinforces the idea of having a game that starts slow and builds up speed as you play along, making it harder each time to stay alive in the game.

### Choose your favourite game aesthetic?

From the users interviewed, they were able to choose two aesthetics that were their favourite when playing a game, the top two were retro with a 100% pick rate, followed by indie with a 44% pick rate. Therefore, the game will be a retro style with indie aspects which conveys a darker tone than pastel colours while still following a bald pallet of colours which are easy on the eyes unlike primary colours.

### On average, how often do you spend time playing videogames a week?

From the users interviewed, from the majority groups, 22.2% (green slice) said that on average they play about 3-4 hours of video games per week, while the other 22.2% (purple slice) said that they played 20+ hours of video games per week. This shows us that people play video games fairly often, and so it is important to make a game that ensures entertainment for hours and is able to keep people engaged in it for a long time.

### What device do you play most games on?

From the users interviewed, 55.6% of users responded that they mostly play video games on their computer/laptop. This will be our main target audience since it is the majority of users, as well as being easier to design a computer game rather than a console game due to compatibility and testing capabilities.

### Which of the following do you think makes a game re-playable?

From the users interviewed, the majority of respondents replied that they feel that a game being competitive, having the ability to progress and power up, and the game being different every time you play are what makes a game replayable. These are features that will be highly considered when designing the potential solution to extend the game’s usable life.

### On a scale 1-10, how important do you think the following this is in games?

From the users interviewed, the highest rated overall as the most important to a game was the difficulty of the game, with most users scoring an 8 on the matter, on a scale of 1-10, 10 being the most important. This will be taken into consideration when designing the game as we want users to be able to enjoy the game to its full potential.

### Do you prefer randomly generated levels or pre-made levels?

From the users interviewed, 44.4% of users replied that they prefer a mix of pre-made levels and randomly generated levels, with 33.3% of respondents skewing towards randomly generated levels, making this about 55.5% of users wanting a rather randomly generated game.This will be taken highly into consideration when developing the game, as our goal is to provide a new experience every time the users opens up the game.

### Skill or progression games?

From the users interviewed, most users mentioned that they prefer a mix of progression and skill games. This will be taken into consideration when developing the game, to ensure the majority of the users will be drawn into playing the game.

## Specify the proposed solution

| **Features** | **Requirements** | | **Processes** | **Justification** | **Limitations** |
| --- | --- | --- | --- | --- | --- |
| Hardware | Software |
| Inputs | Keyboard  Mouse  Joystick  Microphone  Controller  Track ball  Touchpad  Touchscreen  Sensors | Device drivers  Coding programs  IDE  Developing programs  Photoshop  Operating system  Visual user interface | Inputs must be processed both by the device the game is being run on, as well as the game itself. This process must be carried out quite fast with minimum delay to enhance the gameplay.  The inputs will trigger key processes such as pressing WASD and the UP,DOWN,LEFT,RIGHT keys will move the player up,down, left and right accordingly.  By pressing the P key the game will be paused and pressing the escape button the game will be exited and closed.  Furthermore, by pressing either LCTRL or RCTRL buttons, missiles will be shot from the player’s sprite. | The user must have some way to be able to interact with the game, whether this be using a keyboard, mouse, controller or joystick to control the character. Other interesting game mechanics might require a microphone to direct the character, this could also be included as accessibility settings as some people may not be able to use a mouse/keyboard as input properly as they may have a disability that limits their movement. | Some limitations that those input devices may have as said, is that they may not be appropriate for use to people who may have a disability, and the alternatives not being quite reliable or comfortable as microphones require a quiet environment and can often be misunderstood by the computer, it also slows down the user’s reaction time in an action game. |
| Outputs | Screen  Speakers  Headphones  Vibrator motor  Touchscreen | Likewise, for outputs, they must be processed by the hardware and software being run on the device, as well as the game. This is because the game will have to output things such as sound or images, as well as constantly displaying the game.  Key outputs from the game will be things such as if a missile collides with an enemy, they both get deleted, and an explosion sound is played. This is similar to when the missiles are first fired, as a laser sound must be played. If the player collides with an enemy an explosion animation is played and explosion sound is played.  Score should also be displayed, being updated as it increases. Death, pause and start menus must also be displayed and the background must scroll faster as time goes on. | The player must be able to have some way to view the game to play it, this would be done using a visual interface, which would be displayed on a monitor. | Due to the nature of some output systems, some people may not be able to interact with it due to a disability, such as being blind which would prevent them from viewing the screen. |
| Storage | Solid state drive  Cloud storage  CD/ Blu-ray  RAM  ROM | Similar to the previous ones, storage must be processed by the correct drivers, as well as being processed by the game as information such as the code, entities and other information will have to be stored somewhere to be processed and output by the game. | To run the game effectively, the code of the game along with the entities used to create it must be stored somewhere for the user to access it and play as well as having the software obtained or bought from a source that has been storing it such as the cloud or a DVD.  Scores from the player must also be stored, along with their progress and the own device the user is playing on must have enough memory for the game to be loaded up to be processed, temporarily storing data in the device while it's being run. | Depending on the game, different storage sources may be used. Blu-ray provides a higher image quality than a DVD as well as storage to store large games. However this may be inconvenient and can easily break, as well as a lot of devices nowadays do not have a CD reader. Cloud storage is more widely used to download games, however a stable internet connection is required to download it, limiting people with no internet access. And once it has been downloaded it is more likely to be plagirised even if the code is compiled. |

### **Aesthetics**

Aesthetically wise, the game will use a retro game style palette using primary colours to denote characters and pixel art will be used to to display the game. Inputs might also include sound effects when the character performs specific actions such as flying, dying or destroying an object. This will be done to give the game a more retro vibe, as well as immersing the player deeper into the game rather than being a superficial experience.

The success criteria for the past features would be that the user is able to input the appropriate commands to the computer, and the desired outcome is achieved. This could be that the ‘W’ character allows the user to jump, and in the screen it can be immediately viewed when the character jumps up and down as soon as the key is pressed; this simple test shows that both input and output devices are working, along with their corresponding drivers as no information is lost along the way.

Another point of success criteria would be that up to five scores are stored in the database and correctly, with the right order; and displayed with the user's inputted game name, this would mean that the storage system is working correctly which also again involves all the

drivers and hardware that comes along with it.

## Future success criteria

| **Number** | **Requirements** | **Justification** |
| --- | --- | --- |
| 01 | Game can be closed | The game should be able to be halted and closed without the need of directly shutting down the console. |
| 02 | Scrolling background | The game is a scrolling game, therefore the background must scroll to give the impression the spaceship is travelling |
| 03 | Spaceship should be able to move up, down, left, right | The user must be able to have a character which they can view and control |
| 04 | Spaceship can shoot projectiles | The player must have some kind of way to defend themselves, therefore to fit the space theme, a projectile will be added to simulate the spaceship using weapons to make its way through the level |
| 05 | Enemy created | The game must have a purpose, and so the player must have some kind of enemy/ obstacle to play against |
| 06 | Enemies must disappear when shot projectile to. This will increase the player’s score. | The missiles are used to get rid of enemies, therefore they must be destroyed so that they are no longer an obstacle. The score must increase as they have done a task right |
| 07 | Player must die if they collide with an enemy | The game must have some type of challenge apart from dodging and killing enemies, and so the player must be killed if they crash with the enemy, therefore killing them |
| 08 | Player must have an explosion animation if they die | This will immerse the player into the game, as well as signalise they have lost the run. This also gives an end to the game rather than just ending the game immediately. |
| 09 | Death screen menu must be shown if the player dies. Pressing Esc must allow the player to go back to the start | This is also used to signal that the player has died and lost the run, they are prompted to go back to the start where they can start a new game rather than leaving the game entirely. |
| 10 | Start menu must be displayed at the start of the game. Game must not be run until the player presses a button | The player might want to leave the game open before actually playing, as well as it helps prepare the player to mentally prepare the game rather than going straight into the game without warning. |
| 11 | Pause menu must halt the game but not terminate it. Players must be able to unpause the game when they wish to, conserving the same game conditions as before they paused it. | The player might need or want to pause the game at any point without losing their progress, and so a pause function is necessary to halt the game without the player losing the momentum or score they have within the game. |
| 12 | Sound effects must be played when the following occurs: enemy dies, player dies, missile is shot. | Gives ambience to the game, immersing the player further into the experience, making them feel like they are inside the videogame rather than being an outside stand byer |
| 13 | Asteroid created | The game must have a purpose, and so the player must have some kind of enemy/ obstacle to play against |
| 14 | Asteroid must disappear when shot projectile to. This will increase the player’s score. | The missiles are used to get rid of enemies, therefore they must be destroyed so that they are no longer an obstacle. The score must increase as they have done a task right |
| 15 | Player must die if they collide with an enemy | The game must have some type of challenge apart from dodging and killing enemies, and so the player must be killed if they crash with the enemy, therefore killing them |
| 16 | High score display | The game must award points for the performance of the user to give them a sense of purpose in the game and give them a reference point to be able to compare to other users, making it more competitive. |

# 

# **Design**

## Decompose the problem

After reviewing Google Dinosaur, Geometry Dash and Vanguard; I have concluded that I wish to have the following in my solution:

Character

* I want to have an animation for every time it moves
* If the character moves, a jump sound effect will be played
* When the player dies a death animation will be played
* Character should be able to shoot projectiles
* If the player crashes against an obstacle, they die
* If the character is hit by an enemy, they die.
* The higher the score, the faster the character will go

Enemies

* Enemies are able to shoot projectiles at a certain speed
* If enemies are hit by a player projectile, they die
* If the enemies hit the player with a projectile, the player dies
* Enemies can not move
* The higher the score, the more enemies will appear
* Enemies to be randomly generated

Obstacles

* Will appear randomly on edges of screen
* Can not move
* May be destroyable by player
* If player hits it, then player dies

Score system

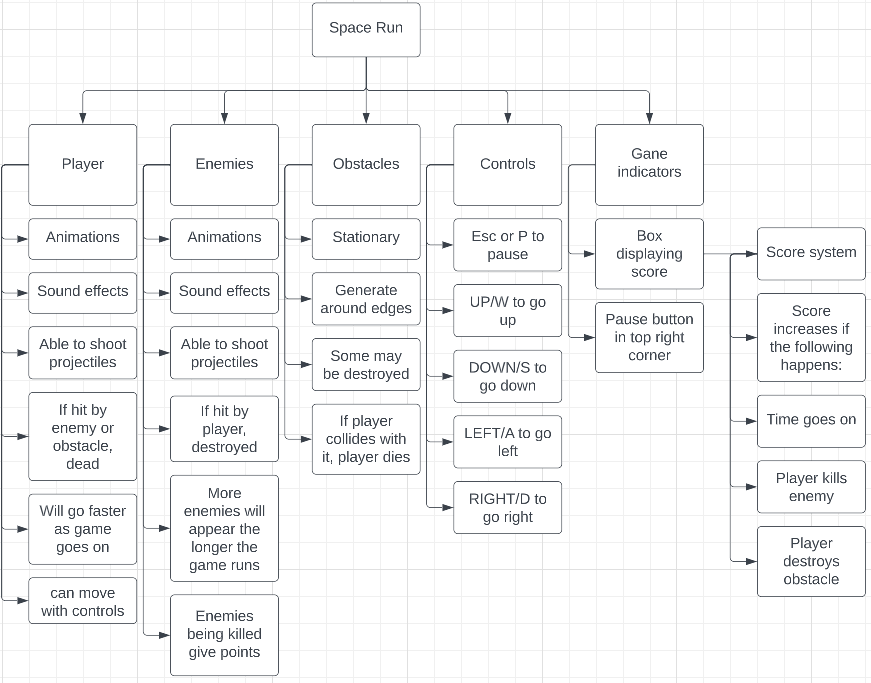
* Score should increase each time the player kills an enemy
* Score should increase as the player travels
* Score should be increased if the player destroys an obstacle
* Score should be stored once the player dies

Game indicators

* A box at the top showing the players score
* Pause button on a corner
* Highest score should be displayed next to the current score in a different colour

Game controls

* “Esc” key to pause the game
* “UP” arrow key to jump/navigate menu
* “DOWN” arrow key to squat/navigate menu
* “LEFT” arrow key to move left on the screen/navigate menu
* “RIGHT” arrow key to move right on the screen/navigate menu
* “RIGHT CTRL” key to shoot projectiles
* “ENTER” key to press buttons navigating through menus



## Describe the solution

I have decided to make the following design choices when designing the game.

**Start screen**

Design

Displaying the game title being fairly big and close to the middle allows the user to easily see it. The background image of the game will be used at the start screen to already set the ambience for the game.

At the bottom a text with instructions is given, again it is close to the middle and has a border around it to bring the user’s attention towards it. In this design it is indicated that the player must press the X key to play the game.

**In game display**

Design

The player sprite will be set in the middle this is because it is where the user’s eyes are typically focused when looking at a screen, this makes the user automatically take notice of where their character is. At the start of the game, no other entities will be spawned and so, the player’s sole focus will be on the character they are controlling.

**In game display - enemies, asteroids and projectiles**

Design

The missiles will be shot from the middle of the player’s current position, they will move towards the right side of the screen since that is where the player will be facing towards.

Both the enemy and asteroids will come from the right side of the screen going to the left, as that is the direction the player will be perceived to be travelling towards, and so it will leave the asteroids and enemies behind if they aren't hit or collided with.

**Death screen**

Design

I decided to lay out the ‘game over’ screen similar to the start screen to match the format. Again, I decided to place the ‘Game Over’ title in the middle to catch the player’s attention easily and the subtitle under it, instead of prompting the player to play, it will prompt the player to play again rather than fully exit the game, this will allow the player to easily start a new game and prevents the player from closing the game, ensuring the game’s durability.

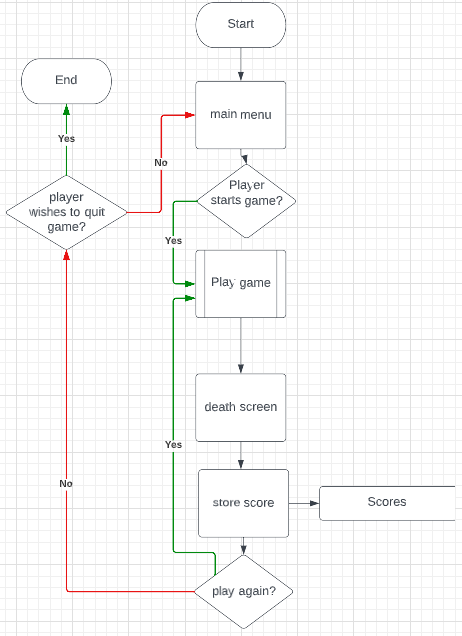
**Pause screen**

Design

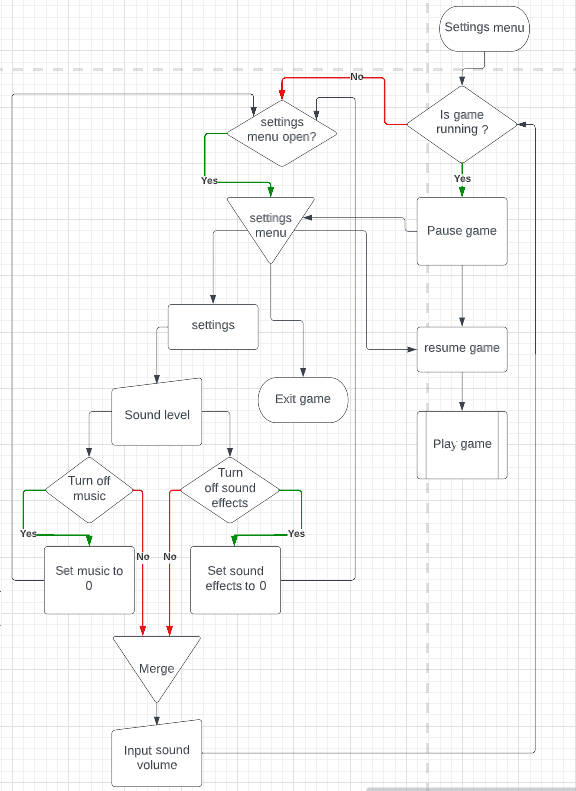
For the pause menu, I did a minimalistic design, just displaying a pause icon in the middle of the screen, again catching the player’s attention while keeping the halted game in the background visible so that when the player wishes to resume the game, they are able to view where they left the game at.

## Systems diagrams[[7]](#footnote-7)

### Game flowchart

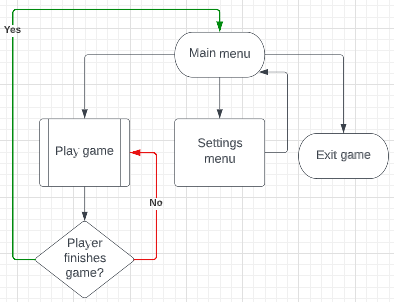


General system diagram



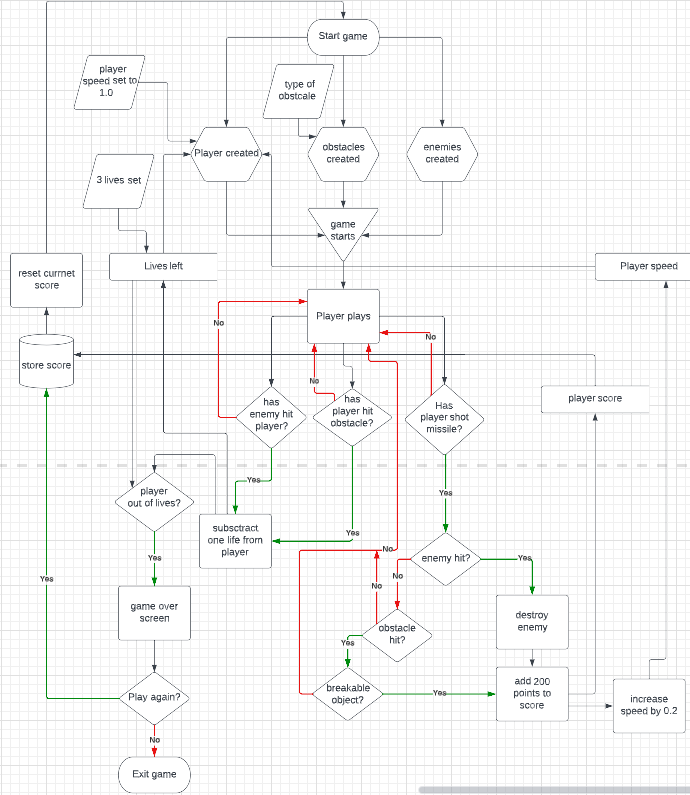
Settings menu flow diagram

## 



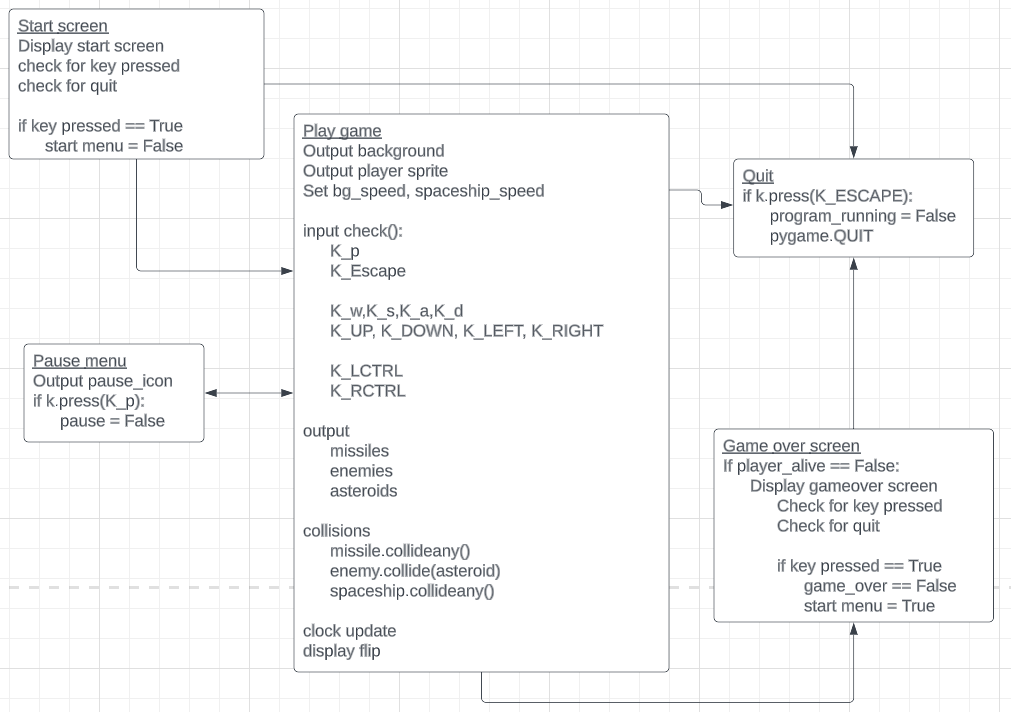
Menu data flow diagram

## 



Game data flow diagram

### Game state diagram



The state diagram shows the behaviour of the system and related fields and interactions while it is active. For example, in this state diagram it is described how the game will detect different actions and act accordingly to the input that has been given, as well as detecting any in-game interactions such as collisions and the change of variables throughout the game required for its functioning in each scenario.

### Class diagram

### 

A class diagram is a visual representation of the different classes within a program, each with their own attributes and methods that are coded within the class. Classes are created using object oriented programming and are used to create instances of the same objects as well as sub classes that share attributes with the parent class. The classes can also be programmed to interact with each other.

| **Assets and Variables** | | | |
| --- | --- | --- | --- |
| **Name** | **Asset/ Variable** | **Description** | **Justification** |
| Player sprite | Asset | The sprite the player will control, in the form of a spaceship | The player must view what they are controlling and where they are positioned so they can make appropriate choices within the game. |
| Missile sprite | Asset | A white missile that is shot from the centre of the player | The player is able to shoot therefore a projectile should be visible to know where it is going |
| Enemy sprite | Asset | An ufo that appears at the left side of the screen | For the game to have some kind of purpose, the player must have some kind of enemy or obstacle to beat. |
| Clock | Variable | Clock speed of the game and dictates how often the game updates. | Without this variable, the game will be frozen and unable to display any updates to the system. |
| Screen | Asset | Sets up the window screen where the game will be set up | The window is created so that images can be drawn on it, such as the characters of the game, menus and backgrounds |
| Bg (background) | Asset | The background image that is used to display the game | Without a background the default background would be white or black, a background image gives it depth and a good atmosphere |
| Start menu | Variable | Indicates whether the game menu loop has been broken so that the start menu image is not displayed and the game is able to run. | The user might want to leave the game open before starting it, therefore a start menu is required so that the game does not keep running indefinitely and ending until the user wishes to play. |
| Start menu surface | Asset | Menu given before starting the game, game will not start until the start menu is clicked off by the user | It makes the player feel welcome and it is a feature presented in all games keeping the format consistent.  The menu surface/ image is kept consistent with the styling of the game |
| Program running | Variable | Checks whether the program is running. If it is not then both the game and the whole program is exited so that it does not run in the background. | Used to quit the game entirely so it does not run in the background taking up space in the computer’s memory |
| Game running | Variable | Checks whether the game is still running. If it is not, then the game goes back to the start | This is used to be able to use the death screen as the game is halted and returned to the start screen once the player decides to click off the death screen. |
| Spaceship alive | Variable | Checks whether the spaceship is alive | If the spaceship is not alive then operations such as shooting missiles and controlling the player are halted. An explosion animation is played when the player dies and the death screen is displayed |
| Enemy count | Variable | Counts the enemies on the screen and how often they spawn | As the game goes on, more and more enemies will be spawned, making it harder to kill them all as well as dodging them |
| Spaceship explosion count | Variable | A timer to count down so that the exploding animation can be played | When the spaceship dies it explodes and so the spaceship sprite is updated with the different frames of the explosion |

## 

## Describe the approach to testing

The following testing methods will be used to test the software at different levels and stages:

* **Black box testing**
  + Black box testing tests functionality of an application/system with no prior knowledge of its internal workings, the tester only viewing the output generated from their input.
  + Only checks the correct outputs from inputs regardless of efficiency
* **White box**
  + White box testing, on the other hand, the tester has knowledge of the internal design of the application and analyses it during testing.
  + Tests all pathways through a program
* **Unit testing**
  + For unit testing, small parts of an application are individually tested for proper operation.
* **Module testing**
  + Individual units or components of the software are tested
* **Functionality testing**
  + Establishes whether each application feature works as per the software requirements.
  + Each function is compared to the corresponding requirement to ascertain whether output is consistent with the end user expectations.
* **End to end testing**
  + Tests the functionality and performance of an application under product-like circumstances and data to replicate live settings.
  + Simulates what a real user scenario looks like from start to finish.
* **User acceptance testing**
  + Software is tested with the intended audience or business representatives
  + There are different types of user acceptance testing, such as alpha and beta testing, contract acceptance, regulation acceptance and operational acceptance
* **Beta testing**
  + Performed by the user
  + Allows real users in a production environment to uncover any bugs or issues before a general release
* **Alpha testing**
  + Performed by the programmer
  + First end-to-end testing of a product to ensure it meets the business requirements and functions correctly

## 

## Prototyping and testing approach

### Prototypes plan

|  |  |  |
| --- | --- | --- |
| **Prototype number** | **Description** | **Success criteria** |
| 1 | The first prototype must be simple, a background must be set up and it must be able to infinitely scroll. Ideally a start menu is created. | * Scrolling background * Start menu available |
| 2 | Spaceship sprite drawn, collisions are tested.  Spaceship must also be able to move | * Spaceship sprite drawn * Spaceship can be controlled by player |
| 3 | Enemy ship is spawned, must travel from left to right  Death upon collision with the spaceship | * Enemy spawned * Death of spaceship upon collision |
| 4 | Start screen and game over screen available | * Start and death screens available and functioning |
| 5 | Asteroids added, must mimic the behaviour of asteroids  Spaceship death upon collision | * Asteroids added, must behave the same manner as enemies |
| 6 | Explosion animation upon death of spaceship  Missiles must be deleted upon collision with enemy, along with the enemy | * Explosion of spaceship upon death * Missiles die when they kill enemy |
| 7 | Asteroids and enemies are able to bounce off each other, as well as the top and bottom of the screen.  Pause menu functions | * Asteroids and enemies collision, must bounce off each other * Pause menu must be available and functioning * Any feedback from testing during the developing stages must be added. |

### 

### Overall testing required

| **Test number** | **Type of test** | **Test data or action** | **Reason for test** | **Expected result** |
| --- | --- | --- | --- | --- |
| 01 | Unit test | Background image for game  Program ran and debugged. | Ensure that the background is drawn on the screen | Load image which should fill the window as a background |
| 02 | Functionality test | Background must scroll sideways, increasing speed the longer it is open.  Program run | The game is a side scrolling shooting game, therefore the background must keep repeating indefinitely and faster until the player quits or dies, making the game harder | Background scrolling to from right to left, getting faster as time goes on |
| 03 | Unit test | Program is run | The spaceship must be drawn on the screen for the user to see on top of the background | Spaceship to appear in the middle of the screen |
| 04 | Module test | Program is run  Appropriate keys are pressed | The spaceship must be able to be controlled by the player using either the WASD keys or the arrow keys | Spaceship to move according to the keys pressed |
| 05 | Module test | Program is run | Check that missiles are shot | Missiles are shot from the middle of the screen |
| 06 | Module test | Program is run and tested | Spaceship fire only running when the adequate keys are pressed | Spaceship fire is played only while the keys are pressed |
| 07 | Unit test | Program is run then quit key is pressed | The game must be able to be quit using the ESC key | Game quits only when the escape key is pressed, not with any other key or moment |
| 08 | Unit test | Program is run and tested | Enemy ship is spawn | Enemy ship is drawn randomly on the left side of the screen |
| 09 | Module test | Program is run and tested  Spaceship is made to collide with enemy | Spaceship and enemy ship collision test | The ship will die when it collides with the enemy |
| 10 | Functionality test | Program is run and tested | Enemy list might overload if it is too big, taking up too much memory.  Therefore the enemy must be deleted once it is off the screen since the player will not be viewing them | Enemy must randomly spawn on the right side and despawned once off the left side of the screen |
| 11 | Module test | Program is run and tested | Both the missile and the enemy must be killed when the program is run. | On missile and enemy collision, enemy killed |
| 12 | End to end test | Program is run and tested | Start menu and game over menu work accordingly. Ship must not be visible in these menus | Ship must not be visible in these menus and only be visible when the right event takes place. |
| 13 | White box | Program was presented and run by the user | The user viewed the code, and decided to find any problems or loopholes | The user be able to find problems to be solved or none at all |
| 14 | Black box | Whole program was tested. User did not know any information about the solution and only pointed to the controls | Ensure that users are able to work the game without any bugs or actions that are not intended to occur in the game. | Everything should work as expected |
| 15 | Unit test | Program is run and tested | Asteroid is spawn | Enemy ship is drawn randomly on the left side of the screen |
| 16 | Unit test | Program is run and tested  Spaceship is made to collide with asteroid | Spaceship and asteroid collision test | The ship will die when it collides with the enemy |
| 17 | Unit test | Program is run and tested | Asteroid entity list might overload if it is too big, taking up too much memory.  Therefore the asteroid must be deleted once it is off the screen since the player will not be viewing them | Asteroid must randomly spawn on the right side and despawned once off the left side of the screen |
| 18 | Unit test | Program is run and tested | Both the missile and the asteroid must be killed when the program is run. | On missile and asteroid collision, asteroid and missile are killed |
| 19 | Unit test | Death animation must be played upon spaceship death | Gives the game a little more ambience and better aesthetic look, allows the player to be indicated the game is over | Death animation played if either asteroid or enemy collide with player |
| 20 | Functionality test | Program is run | Enemy and asteroid must bounce off each other if collided with, as the enemy must behave differently from the asteroid, as well as making the game harder and more interesting. | When a collision occurs between an asteroid and enemy they must bounce off each other. Asteroids must not leave the screen if collided with. |
| 21 | Module test | Program is run  Missiles are shot | Missiles must be limited as this will make the game harder, rather than being able to spawn infinite missiles throughout the screen deleting all obstacles. | Only one missile can be shot until the current one is gone |
| 22 | Functionality test | Program run  Pause button is pressed | The player might need or want to pause the game at any point without losing their progress, and so a pause function is necessary to halt the game without the player losing the momentum or score they have within the game. | Game must be halted on the current state it is in, without completely exiting the game. Player must be able to resume the game when they wish by pressing the same button again |

## 

# 

# **Developing the coded solutions**

For this project there are different software development

* **Waterfall model:**

The Waterfall model is the most[[8]](#footnote-8) widely used model as it is the oldest and easier to carry out. It is also known as the linear-sequential[[9]](#footnote-9) model, this is because once a stage of the ‘Waterfall’ process is done, you move on to the next stage and the next stage will always rely on the information or data provided by the earlier stage. The waterfall model is a sequential model, different development is broken down and divided into different stages which are carried out successively, one after the other, this model is suitable small projects where ‘deliverables[[10]](#footnote-10)’ are easy to define from the start; and teams can follow the sequential steps of the process and not move on to the next stage until the previous one has been completed.

* **Iterative model**

The dictionary definition of iteration[[11]](#footnote-11) is “the repetition of a process or utterance” and the repetition of a mathematical or computational procedure applied to the result of a previous application, typically as a means of obtaining successively closer approximations to the solution of a problem” (in a computing scenario).

As the definition describes, the Iterative model consists in repeating the same process over and over again, this makes this model able to be changed and reviewed almost continuously. Unlike the Waterfall model, you can review and edit the previous stages of the program system and instead of it being linear model (where you follow a series of steps from start to end), the Iterative model is more thought about as a cyclical process where the process is represented as a circle and is repeated over and over.

The iterative model is a ‘particular particular implementation[[12]](#footnote-12)’ where you start with a simplified implementation and then decisions taken are executed as the model process keeps repeating. And as the model process keeps repeating, the project keeps adding on implementations, making the project more elaborate and complex. This type of model is ideal for “agile[[13]](#footnote-13)” organisations, and for a project where that is not very clear how it will work and and the end result may vary from an initial analysis, this is because the iterative process allows the system to be added new things as they are thought about and it is easy to adapt to the client’s or the project’s new needs.

**Waterfall model vs Iterative model**

Both the Waterfall model and the Iterative model are very different, but that does not mean that one is worse than the other. They both have their similarities and differences, each with its own **advantages** and **disadvantages**.

Both models are simple to understand, this allows anyone to be able to quickly understand and carry out the model without making any mistakes in the process while developing the system using either model.

The Waterfall model follows a linear guide of work where you can not go and edit previous stages, whereas the Iterative model main use is repetition and going over the same steps over and over again. The Waterfall model is easier to understand and is very specific on deadlines as each stage has to meet certain deadlines in order to work properly.

Therefore for this project, I have made the decision to use the iterative model to develop my coded solution to the problem, being able to back and add things as required to the solution, while assessing and retesting every single aspect of the solution when new things are added.

## 

## Prototyping

|  |  |  |  |
| --- | --- | --- | --- |
| **Prototype number** | **Description** | **Success criteria** | **Accomplished?** |
| 1 | The first prototype must be simple, a background must be set up and it must be able to infinitely scroll. Ideally a start menu is created. | * Scrolling background * Start menu available | Yes  Yes  Prototype one: tests 01, 02 |
| 2 | Spaceship sprite drawn, collisions are tested.  Spaceship must also be able to move | * Spaceship sprite drawn * Spaceship can be controlled by player | Yes  Yes  Prototype two: tests 03,04 |
| 3 | Enemy ship is spawned, must travel from left to right  Death upon collision with the spaceship | * Enemy spawned * Missiles drawn * Death of spaceship upon collision | Yes  Yes  Yes  Prototype three: tests 05, 08, 09 |
| 4 | Start screen and game over screen available | * Start and death screens available and functioning | Yes  Prototype four: test 12 |
| 5 | Asteroids added, must mimic the behaviour of asteroids  Spaceship death upon collision | * Asteroids added, must behave the same manner as enemies | Yes  Prototype five: tests 15, 16 |
| 6 | Explosion animation upon death of spaceship  Missiles must be deleted upon collision with enemy, along with the enemy | * Explosion of spaceship upon death * Missiles die when they kill enemy | Yes  Yes  Prototype five: tests 18, 19 |
| 7 | Asteroids and enemies are able to bounce off each other, as well as the top and bottom of the screen.  Pause menu functions | * Asteroids and enemies collision, must bounce off each other * Pause menu must be available and functioning * Any feedback from testing during the developing stages must be added. | Yes  Yes  Yes  Prototype five: tests 20, 21, 22 |

(See overall testing and developing below for further reference)

## Overall testing and developing

| **Test number** | **Date of test** | **Type of test** | **Test data or action** | **Reason for test** | **Expected result** | **Actual result (+ solution if not expected result)** |
| --- | --- | --- | --- | --- | --- | --- |
| **Prototype 1** | | | | | | |
| 01 | 2 Mar 2023 | Unit test | Background image for game  Program ran and debugged. | Ensure that the background is drawn on the screen | Load image which should fill the window as a background | The image had a different RGB profile |
| 02 | 2 Mar 2023 | Functionality test | Background must scroll sideways, increasing speed the longer it is open.  Program run | The game is a side scrolling shooting game, therefore the background must keep repeating indefinitely and faster until the player quits or dies, making the game harder | Background scrolling to from right to left, getting faster as time goes on | Background scrolled appropriately. However, when updating the screen the background was dragged more than repeated. Adjustments were made on how the screen is updated and issue was solved |
| **Prototype 2** | | | | | | |
| 03 | 7 Mar 2023 | Unit test | Program is run | The spaceship must be drawn on the screen for the user to see on top of the background | Spaceship to appear in the middle of the screen | Result as expected |
| 04 | 17 Mar 2023 | Module test | Program is run  Appropriate keys are pressed | The sprite must be able to be controlled by the player using either the WASD or the arrow keys | Spaceship to move according to the keys pressed | Result as expected |
| **Prototype 3** | | | | | | |
| 05 | 18 Apr 2023 | Module test | Program is run | Check that missiles are shot | Missiles are shot from the middle of the screen | Missiles were shot slighting off the spaceship centre. X-y coordinates were adjusted |
| 06 | 20 Apr 2023 | Module test | Program is run and tested | Spaceship fire only running when the adequate keys are pressed | Spaceship fire is played only while the keys are pressed | Spaceship fire would turn on and off even if the spaceship was moving. This was solved by individually stating which keys must be pressed when the spaceship must move, so that the animation could be played. |
| 07 | 20 Apr 2023 | Unit test | Program is run then quit key is pressed | The game must be able to be quit using the ESC key | Game quits only when the escape key is pressed, not with any other key or moment | Initially the game would exit as soon as interaction was made, this was quickly solved by changing the set out of the code and the way the game would detect the key being pressed |
| 08 | 21 Apr 2023 | Unit test | Program is run and tested | Enemy ship is drawn | Enemy ship is drawn randomly on the left side of the screen | Image was too big covering the entire screen and the colourkey made parts of the enemy image transparent in some places.  The image was edited to make the black a lighter type of black and the enemy image was resized. |
| 09 | 21 Apr 2023 | Module test | Program is run and tested  Spaceship is made to collide with spaceship | Spaceship and enemy ship collision test | The ship will die when it collides with the enemy | The ship would die when colliding with the enemy however the hitbox was off and so, the ship would die as it would ‘collide’ with the enemy, however the ship would be substantially far away from the enemy.  Hitbox was re-centered and adjusted. |
| 10 | 23 Apr 2023 | Functionality test | Program is run and tested | Enemy list might overload if it is too big, taking up too much memory.  Therefore the enemy must be deleted once it is off the screen since the player will not be viewing them | Enemy must randomly spawn on the right side and despawned once off the left side of the screen | Result as expected |
| 11 | 24 Apr 2023 | Module test | Program is run and tested | Both the missile and the enemy must be killed when they collide. Program is run. | On missile and enemy collision, enemy killed | The enemy would be killed but not the missile on collision.  The loop used to kill the enemy was added to also kill the missile simultaneously |
| **Prototype 4** | | | | | | |
| 12 | 24 Apr 2023 | End to end test | Program is run and tested | Start menu and game over menu work accordingly. Ship must not be visible in these menus | Ship must not be visible in these menus and only be visible when the right event takes place. | Result as expected |
| 13 | 26 Apr 2023 | White box  User test | Program was presented and run by the user | The user views the code, and decides to find any problems or loopholes | The user is able to find problems or none at all | Hit box of spaceship was used to make the spaceship to stop moving further in that direction if it hit the side of the screen |
| 14 | 27 Apr 2023 | Black box | Whole program was tested. User did not know any information about the solution and only pointed to the controls | Ensure that users are able to work the game without any bugs or actions that are not intended to occur in the game. | Everything should work as expected | A few bugs were found, some of those were that enemies would stop spawning after the enemy count reset would drop to 0. This was solved by resetting the count once it reached 0 back to a higher number so spaceships could keep spawning.  The spaceship would be able to shoot after death as well as collide with the spawning enemies but invisible. This was fixed by killing the player entity after death, and changing some of the collision loops. |
| **Prototype 5** | | | | | | |
| 15 | 24 Apr 2023 | Unit test | Program is run and tested | Asteroid is spawn | Asteroid is drawn randomly on the left side of the screen | As expected  (Asteroid modelled after the enemy) |
| 16 | 27 Apr 2023 | Module test | Program is run and tested  Spaceship is made to collide with asteroid | Spaceship and asteroid collision test | The ship will die when it collides with the asteroid | As expected  (Asteroid modelled after the enemy) |
| 17 | 27 Apr 2023 | Unit test | Program is run and tested | Asteroid entity list might overload if it is too big, taking up too much memory.  Therefore the asteroid must be deleted once it is off the screen since the player will not be viewing them | Asteroid must randomly spawn on the right side and despawned once off the left side of the screen | As expected  (Asteroid modelled after the enemy) |
| **Prototype 6** | | | | | | |
| 18 | 27 Apr 2023 | Unit test | Program is run and tested | Both the missile and the enemy must be killed when the program is run. | On missile and asteroid collision, asteroid and missile are killed | As expected  (Asteroid modelled after the enemy) |
| 19 | 27 Apr 2023 | Unit test | Death animation must be played upon spaceship death | Gives the game a little more ambience and better aesthetic look, allows the player to be indicated the game is over | Death animation played if either asteroid or enemy collide with player | As expected |
| **Prototype 7** | | | | | | |
| 20 | 27 Apr 2023 | Functionality test | Program is run | Enemy and asteroid must bounce off each other if collided with, as the enemy must behave differently from the asteroid, as well as making the game harder and more interesting. | When a collision occurs between an asteroid and enemy they must bounce off each other. Asteroids must not leave the screen if collided with. | At first, only certain parts of the asteroid and the enemy colliding would make them bounce off each other.  This was fixed by making the asteroid into a sprite and making them collide at the centre of their hitboxes rather than the sides of them. |
| 21 | 27 Apr 2023 | Module test | Program is run  Missiles are shot | Missiles must be limited as this will make the game harder, rather than being able to spawn infinite missiles throughout the screen deleting all obstacles. | Only one missile can be shot until the current one is gone | Missiles would still fire continuously, this was solved by deleting the already existing missile if another one was spawned  This is an example of thinking concurrently as both the missile and obstacle must be deleted. |
| 22 | 27 Apr 2023 | Functionality test | Program run  Pause button is pressed | The player might need or want to pause the game at any point without losing their progress, and so a pause function is necessary to halt the game without the player losing the momentum or score they have within the game. | Game must be halted on the current state it is in, without completely exiting the game. Player must be able to resume the game when they wish by pressing the same button again | The game would completely freeze and not start  Then the game would pause permanently or the game would start doing logical operations without the game actually started  A while loop was placed to only function if the variable pause was false. And the game code was placed inside this loop so that it would not run unless stated it could, while still keeping the game current game variables in the state they were in, unchanged, effectively pausing the game. |

The following test images display the code’s state after it has been patched/fixed following the occurred problem. The final code will be all available under the “Codes and files” section.

Test 01

This initial prototype just loaded the background image, a red square was added to check how the screen updated.

Test02

On the second prototype, red square was also given a hitbox, this was to start testing the collisions from the spaceship sprite, as well as testing that the spaceship was drawn

Test 03

In this second prototype, the only function of the game was for the spaceship to be able to move up and down, with the background being updated appropriately and scroll faster and faster.

Test 04

The next stage of the second prototype was to be able to shoot the projectiles. This would again update the sprites and the background, ensuring that unlike the first prototype, the background and images did not lag.

Test 05

Added arrow keys

Test 06

The third prototype started to replace the rectangle with an enemy, this would spawn off the right and travel towards the left, as well as be able to collide with the enemy. Enemies were made to be able to spawn more as time went on.

Test 09

In this third prototype, when testing the collision between the spaceship and the enemy the hitbox was off, as the spaceship would die while being far away from the enemy. The hitbox of the spaceship was drawn, resulting in viewing that the hitbox was off, this was then adjusted by creating the ‘offset’ variable to readjust the hitbox.

Test 10

Added enemies

Test 11

This is part of the fourth prototype, the menus were added so that they could add a better interactivity with the game, in the game over screen asteroids and enemies can keep spawning to give the game more ambience rather than suddenly disappearing, while in the start screen no asteroids or enemies are generated yet.

Test 14

In this fifth prototype, the asteroids were added and modelled after the second’s prototype enemies, still being able to move from left to right and spawn more asteroids as the game goes on along with the enemies.

Test 16

In the sixth prototype, the explosion animation was added for the spaceship upon the spaceship's collision with an enemy/asteroid. This can be viewed in the video that will be recorded upon the final prototype.

Test 17

Test 18

For this sixth prototype, refinements were made to the code to allow for better interactions between the user and the game. The missiles were made to disappear when colliding with both the enemy and asteroid, as well as the asteroid disappearing on collision.

Test 19

The explosion count refers to the time passed after the player has been hit, this makes then triggers a chain reaction and depending on which stage it is at, it will update the spaceship sprite with a different death animation until it is finalised.

Test 20

Finally, for this final seventh prototype more adjustments were made, as the enemy was able to bounce off the asteroid and screens, as well as allowing the asteroid to bounce off the top and bottom of the screen.

Test 21

For the seventh prototype, the number of missiles was also limited after some user interviews on the prototypes, as the game was too easy if you were able to spam missiles, and so the missile number was limited to one.

Test 22

Finally, the pause menu was implemented to allow the user to pause

## Final aesthetic design

Start screen - after being created

The font of the title and letters are made to be cartoonish to make it more appealing to a younger audience.

I decided to make the font white as it would stand out more on a white background.#

I placed an uneven oval around the ‘Press Any key to start’ so it would stand out to the user, and to also make it seem like some sort of button even if it is not.

White also combines really well with the space theme as the other colours used in the game are fairly clear, apart from the black space background; as the planet surface is a bright blue.

In game display - after being created

I have decided to place the score at the top left of the screen so that the user can easily look at it as they play, and on the other side ‘Press P to pause’ as most games have a pause button on that side of the screen. This lettering is in the same font as the start screen to keep the format of the game consistent.

The ship will start in the middle, so it is easier to locate by the player at the start of the game as their eyes will typically be focused on the middle of the screen as the title was also in the middle.

The spaceship has the same colour palette as the background, but has some brighter colours than the background so it can stand out to the user, and it is easier to see. The sprites in the game are pixel art, with soft but dark colour pallets following the user interviews which suggested players prefer games with retro and indie aesthetics.

In game display (enemies, asteroids and projectiles) after being created

I have decided that, for the game, following the initial design and interviews; that the spaceship will still be placed in the middle of the screen at the start. The enemy ship still follows the indie and retro style, along with the missile also being minimalistic and simple, having a white colour which would greatly contrast with the black background so that they are easily visible.

Death Screen - after being created

The place where the player died is shown with the ship blown up, the game will end with a game over screen, again with the same format as the rest of the game keeping the interface consistent across the game and indications to go to the start screen so the player can decide to leave the game or play again.

Pause screen - after being created

As the design, the pause menu is minimalistic, allowing for the player to clearly see the game has been paused. The pause icon is white, to again contrast the black background, matching the colour pallet and being easily visible.

## Testing for evaluation

#### User acceptance testing 1

|  |  |
| --- | --- |
| Questions | Feedback (filled by user) |
| Were the game visuals easy to see? | Yes, the text, spaceship, enemies and asteroids contrast really well with the background. Makes it easy to see in the dark at night. |
| Do you think the game meets your criteria for a side scroller game? | Definitely yes, however it would be better if the game was able to have different modes, such as having a level where it can scroll sideways, up or downwards to make it more challenging. |
| Are you happy with how enemies/obstacles are generated? | No, some obstacles spawn too low on the screen for them to actually be challenging and the generation is too slow, as there is only up to 2 or 3 things on the screen at the time with most of them generated at the very top or bottom |
| Did you find the controls easy to use? | Yes |
| Are the colours used to make the game aesthetically pleasing? | Yes, very retro |
| Does the game have an appropriate level of difficulty? | Yes, I like the fact that the game starts slow, allowing you to pace yourself with the game |
| Were there any problems encountered while playing? | No |
| In your opinion, is the platform used to play the game appropriate? | Yes, playing on a computer/laptop is very comfortable |
| Would you consider playing the game again/downloading it in the future? | Definitely yes |
| Additional feedback | |
| Overall I think the game is quite good, although I would like for more features to be added, such as powerups and different game modes; as well as levels which can have a higher difficulty level to the infinite level in which you are able to progress. | |

#### User acceptance testing 2

|  |  |
| --- | --- |
| Questions | Feedback (filled by user) |
| Were the game visuals easy to see? | Yes, although I would like to have a colour blind mode since the colours on the enemy are quite dull for me. Otherwise the use of the colour white on black makes it easy to see for anyone despite being colour blind. |
| Do you think the game meets your criteria for a side scroller game? | Yes definitely. |
| Are you happy with how enemies/obstacles are generated? | I feel like it is good how they are generated, however I would like for enemies and obstacles to also spawn from different directions apart from the right side of the screen. |
| Did you find the controls easy to use? | Yes, I love that there are two different controls that you can use depending on your preference. |
| Are the colours used to make the game aesthetically pleasing? | Yes, but again, I wish there was a way to edit the colours or be able to make them more colour blind friendly. |
| Does the game have an appropriate level of difficulty? | Yes |
| Were there any problems encountered while playing? | No |
| In your opinion, is the platform used to play the game appropriate? | Yes but no, I would very much like for a mobile version to be made for this game since it makes it more casual and it would allow me to play it at any time. |
| Would you consider playing the game again/downloading it in the future? | Yes, however it would have to be a mobile version since I do not see myself sitting down on my computer playing this game, as there are more interesting games on my laptop. |
| Additional feedback | |
| I would like there to be sound effects and background music added since it would make the game feel more ‘real’ and just give a better retro arcade vibe overall. | |

#### User acceptance testing 3

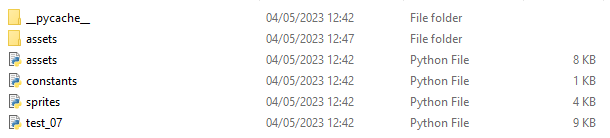
|  |  |
| --- | --- |
| Questions | Feedback (filled by user) |
| Were the game visuals easy to see? | Yes |
| Do you think the game meets your criteria for a side scroller game? | Yes it is done great |
| Are you happy with how enemies/obstacles are generated? | Yes, they are generated in a way that makes sense |
| Did you find the controls easy to use? | It is easy to use, not too complicated |
| Are the colours used to make the game aesthetically pleasing? | It makes me feel like I'm back to the old days, I really like it. Especially since retro style is so trendy right now, and so aesthetically being really good. |
| Does the game have an appropriate level of difficulty? | Of course, if you play for a while it gets harder but it is not impossible so the level of difficulty is done great |
| Were there any problems encountered while playing? | I did not experience any problems while playing |
| In your opinion, is the platform used to play the game appropriate? | I do not see anything wrong with it. |
| Would you consider playing the game again/downloading it in the future? | Yes, but I would like to see updates from time to time. With updates i would definitely stay longer to play |
| Additional feedback | |
| In my opinion, the game is done great but needs updates from time to time, to keep the customer interested. It could involve awards such as: colour changing of the background, of the spaceship and of the obstacles. It could also involve shape changing rewards of obstacles and spaceships.  I also think that levels should be added since there are not any, I would like to see my progress through the game to have a sense of improvement, or at least be able to level up. | |

### Improvements for usability

Based on the user feedback above, it has been concluded that the game is quite successful. However, some points have been made which will be considered following any updates to the game. This would be the addition of sound effects, power ups that can be picked up in game and the addition of levels. Coins might also be added, so that players are able to purchase skins for all spaceships, asteroids and enemies; as well as add a colour blind mode which would include colour adjustments for different types of colour blindness.

## Code and files

### Game files



The game has been split into different files, so that the game as a whole is split up which makes it easier to find different parts of the code in the event that they need to be changed. This makes it easier to navigate through the code and reduces the amount of global variables required within the code, as different parts of the code can be recalled and imported into the main file that puts everything together.

* Assets: this file sets up the dimensions, loads and resizes the sprites and surfaces so that they can be imported into the main game and used.
* Constants: contains the main constants to be used in the game, this allows for quick change of variables without the need of individually changing the variables in each part of the game where the constants have been used. This may include the size of the screen, speed of the game and other things such as the movement speed of the spaceship.
* Sprites: takes the files from assets and turns the images into sprites, creates different classes for each entity in the game and interactions/controls to be used within the game.
* Test\_07: this file contains the actual code for the game and imports all of the files mentioned above. It mainly sets the controls for the spaceship and contains different loops for the menus and the game as a whole to run and collisions between sprites.

#### File inside ‘assets’

This folder contains all the different graphics that are imported into the game, which are first set up by the assets and sprites code, giving them each property and turning them into surfaces/sprites accordingly.

#### Files inside ‘graphics’

Images used in the game, the backgrounds are set to black as the colourkey to make the background transparent is set to black.

### Assets code

This code above displays the spaceship sprites, this includes the sprite’s width and height. The sprites surface is set up so that later on an image can be loaded up to fit the surface the first two spaceship surfaces display the spaceship while it is alive, with the spaceship motor on (spaceship\_1) and the spaceship with the motor on while its moving for an animated effect (spaceship\_2), these are shown below.

The following variables in the code are the explosion animations when the spaceship dies.

Much like the spaceship, the missile sprite’s height and width is set and the right image is selected from the sprite sheet to select the missile that is facing to the right. Missile image is shown above.

Again, like the missile sprite, the enemy sprite is set up and the surface is created with their heights and widths. Image of the enemy is also shown above next to the code.

Again, like the previous sprites, the asteroid sprite is set up and the surface is created with its set heights and widths. Image of the enemy is also shown above next to the code.

These variables represent the start, game over and pause menus. They are made into surfaces and are scaled to fit and be positioned in the middle of the screen.

Below the menu images are displayed (the ones on a blue background are transparent):

This is where the images for the sprite sheet are split into equal parts and the images are selected and each is assigned to their respective image.

The following variables in the code are the explosion animations when the spaceship dies.

The colour keys are set to black, which essentially makes the black background of the sprite sheet transparent when it is drawn into the game.

This is the code for the sprite graphic for the missile, as before, this is the loading of the spritesheet, and the selection of the image to be used, along with the colourkey to make the black background transparent.

This is the code for the sprite graphic for the enemy, as before, this is the loading of the spritesheet, and the selection of the image to be used, along with the colourkey to make the black background transparent.

Likewise, this is the code for the sprite graphic for the asteroid, as before, this is the loading of the spritesheet, and the selection of the image to be used, along with the colourkey to make the black background transparent.

And again, these are the codes for the sprite graphic for the start menu and game over menus, as before, the spritesheet is loaded, and the selection of the image to be used, along with the colourkey to make the black background transparent although the game over does not need to be colour keyed since it is already transparent.

As before, this is the code for the sprite graphic for the pause menu. The spritesheet is loaded, along with the colourkey to make the black background transparent and the pause icon is resized to appropriately fit the screen.

### Constants code

These are constants that may need to be changed often, this includes the screen width,height, the speed of the spaceship and the enemy and asteroid count resets for spawning. These constants can easily be changed without the need to change the main code, having to individually change every single variable’s value while also reducing the amount of global variables used.

### Sprites code

The game imports pygame, constants file and assets file which are used in this part of the code.

The spaceship’s hitbox adjustments are also listed.

The class for the spaceship is created with its respective attributes and methods.

These are the methods for the spaceship, which allow it to move, have a hitbox and shoot projectiles.

This is the class for the asteroid, with the image, hitbox and methods which allow it to move.

Same goes for the enemy sprite, as it has the same methods and classes as the asteroid.

This is the missile class, again with its attributes and methods which allow it to move and spawn

The class groups are then placed into a simple variable which allows for the entity to be called and spawn when required.

### Game code (test\_07)

This is the start of the initial code for the main game. All the libraries and files needed are imported at the top, the pygame screen is set up, with the clock. Background image is loaded up and displayed, along with the asset graphics, and spaceship is created and added to the screen.

The program is then initialised, with the start menu created, and the game keeps checking if the game has been quit so that the game does not run in the background.The start menu is then initialised and checked if it has been quit yet so the actual game can be started.

A new game is started, the variables are reset to default in the case that a new game is being created after the player’s death. This makes the spaceship spawn in the middle of the screen, the speed of the game is set to 0, hitboxes are set up and the spaceship is set to alive, with the explosion count to 0 since it is alive. The asteroid counts and enemy counts are also reset so they can start spawning slowly.

Set up the game running so that it can be terminated upon death so that the program will return to the start screen loop. The game also sets up the pause menu so that it can be activated at any time while in game.

While pause is not active the game is run, which allows the game to freeze any updating variables while the game is running without fully terminating the game, this includes the background movement, spaceship movement and both asteroid and enemy spawning, as well as the shooting of projectiles.

This part of the code is in charge of spawning and moving asteroids and deleting them if they go off the screen.

The same goes for enemies, this part of the code ensures that they keep spawning and moving, and are deleted if they go off the screen.

This part of the code detects the collisions between the enemies and asteroids so that they can bounce off each other if they collide.

This checks if the spaceship is alive, the first part of the code allows for the spaceship to shoot missiles, and caps the missiles shot by one so any subsequent missiles shot are deleted unless they collide with an enemy first or go off the screen.

This part of the code checks whether the missile has collided with any obstacles, this then deletes both the obstacle hit and the missile themselves.

Then the code sets up the images to be displayed when moving the asteroid as all but the key going to the left must have an animation.

This plays the death animation for the spaceship upon collision, which results in toggling the boolean variable for ‘spaceship alive’ into false so that the game over menu can be displayed.

This allows for the game to go back to the start screen after the run is over so that the cycle can start again if wished so by the user.

# **Evaluation**

## Success criteria review

| **Number** | **Requirements** | **Justification** |  | **Reference** |
| --- | --- | --- | --- | --- |
| 01 | Game can be closed | The game should be able to be halted and closed without the need of directly shutting down the console. |  | After test01 |
| 02 | Scrolling background | The game is a scrolling game, therefore the background must scroll to give the impression the spaceship is travelling |  | After test 02 |
| 03 | Spaceship should be able to move up, down, left, right | The user must be able to have a character which they can view and control |  | Test 03 and Test 04 |
| 04 | Spaceship can shoot projectiles | The player must have some kind of way to defend themselves, therefore to fit the space theme, a projectile will be added to simulate the spaceship using weapons to make its way through the level |  | After test 05 |
| 05 | Enemy created | The game must have a purpose, and so the player must have some kind of enemy/ obstacle to play against |  | After test 08 |
| 06 | Enemies must disappear when shot projectile to. This will increase the player’s score. | The missiles are used to get rid of enemies, therefore they must be destroyed so that they are no longer an obstacle. The score must increase as they have done a task right |  | After test 11 |
| 07 | Player must die if they collide with an enemy | The game must have some type of challenge apart from dodging and killing enemies, and so the player must be killed if they crash with the enemy, therefore killing them |  | After test 16 |
| 08 | Player must have an explosion animation if they die | This will immerse the player into the game, as well as signalise they have lost the run. This also gives an end to the game rather than just ending the game immediately. |  | After test 16 |
| 09 | Death screen menu must be shown if the player dies. Pressing Esc must allow the player to go back to the start | This is also used to signal that the player has died and lost the run, they are prompted to go back to the start where they can start a new game rather than leaving the game entirely. |  | After test 12 |
| 10 | Start menu must be displayed at the start of the game. Game must not be run until the player presses a button | The player might want to leave the game open before actually playing, as well as it helps prepare the player to mentally prepare the game rather than going straight into the game without warning. |  | After test 12 |
| 11 | Pause menu must halt the game but not terminate it. Players must be able to unpause the game when they wish to, conserving the same game conditions as before they paused it. | The player might need or want to pause the game at any point without losing their progress, and so a pause function is necessary to halt the game without the player losing the momentum or score they have within the game. |  | After test 21 |
| 12 | Sound effects must be played when the following occurs: enemy dies, player dies, missile is shot. | Gives ambience to the game, immersing the player further into the experience, making them feel like they are inside the videogame rather than being an outside stand byer |  | Due to time constraints and compatibility errors, it was not possible to implement sound effects in the game. However this is a feature that can easily be implemented into the game in future updates. |
| 13 | Asteroid created | The game must have a purpose, and so the player must have some kind of enemy/ obstacle to play against |  | After test 17 |
| 14 | Asteroid must disappear when shot projectile to. This will increase the player’s score. | The missiles are used to get rid of enemies, therefore they must be destroyed so that they are no longer an obstacle. The score must increase as they have done a task right |  | After test 18 |
| 15 | Player must die if they collide with an enemy | The game must have some type of challenge apart from dodging and killing enemies, and so the player must be killed if they crash with the enemy, therefore killing them |  | After test 16 |
| 16 | High score display | The game must award points for the performance of the user to give them a sense of purpose in the game and give them a reference point to be able to compare to other users, making it more competitive. |  | Due to time constraints and compatibility errors, it was not possible to implement a high score display or storage system in the game. However this is a feature that can easily be implemented into the game in future updates.  However in some prototypes of the game, it was shown internally the scoring system that would print the score when it was updated, either when killing an enemy/asteroid or dodging them. However, due to the time constraints it was not possible to create an UI that would display them in a consistent form with the rest of the game or be able to be stored in an external file. |

## White box interview (test 13)

I carried out some white box testing, testing each individual part of the program and its efficiency. Initially I decided to test the main core features of the game, such as the start, pause and death screens. Everything worked as intended, including the user returning to the start screen after death, not being able to access the pause during the start and death screens; as well as adequate collisions and interactions between different entities and sprites within the game.

Then I proceed to test the efficiency of the code, by attempting to lag out the game, freeze it and find glitches. Some actions performed was the attempt to overload the missile list by spamming missiles constantly, this would eventually freeze the game once there were enough entities within the game. This was fixed by limiting the amount of missiles shot, this would completely reduce the entity cramming in the game as well as make the game harder as previous testing proved that being able to indefinitely spam missiles made the game too easy.

Another test I carried out was to spam the pause button to see if any glitches would appear or the game would freeze due to the constant input of “pause”; unlike the previous test, this did not result in any problems and was working and designed.

## 

## Black box interview (test 14)

The game was handed to a user who was instructed on the game controls. The main focus of a black box test is to observe in detail how users interact with the game without any previous knowledge of the internal functioning of the game, and aims to focus on the pure outputs of the game based on the inputs. The user played the game normally and everything seemed to work as intended, however towards the end, the user got bored after death and decided to spam buttons, this resulted in the user being able to still interact with the actual game while on the death screen, resulting in being able to still collide with enemies and shoot missiles despite the actual spaceship sprite not being visible. This was noted down and later fixed by rearranging the following code inside the ‘spaceship alive’ loop.

The user also mentioned that by being able to spam the missiles it was too easy, therefore the missile limit was reduced to one, being unable to spawn new missiles unless the current one was destroyed, attempting to create a new missile resulted in the previous one being deleted.

This test was screen recorded as it recorded, as well as the result after the issues were fixed.

## Usability review

The game is designed to be quite simple, with two different sets of controls to satisfy the user’s different control wishes, as mentioned above in the user interview that about half of users wish to use the WASD keys to control the game, and the other half to use the arrow keys. Each key does the intuitive function that is expected based on the positioning of the key, as W/UP will make the spaceship go up since they are the ones placed pointing upwards.

The spaceship, asteroids, missiles and enemies use clearer/brighter colours than the background so that they can easily stand out to the user while playing, this is also applicable to the different menus, having a big contrast with the background so that they are easy to read. The background also uses mainly a dark colour, making it softer on the eyes if the user wishes to play at night, while still being clearly visible during daytime as the different sprites will still stand out despite the background being dark.

Some improvements in the future for the future usability of the game would be to allow players to adjust the colours of the game, as well as create a colourblind friendly environment to those who wish to have it. However, since the interface for this retro-arcade game is so simple as inspired by the simplicity of the interface of the original arcade games from the 80’s there is not much to add to the interface to make it more user friendly since there are so few elements that affect it; and although most players now a days can understand the words ‘start’ and ‘game over’ it would be possible in the future to be able to translate the game’s interface into another language if required.

## Overall review

Overall the success criteria review is quite good since all but two tasks were accomplished during the time given for the development of this solution. However, to compensate for the lack of sound effects, the game does have some new features that were implemented on the go, such as enemies being able to bounce off and change the direction of both the spaceship and the asteroids if they collide.

In the future, for this game, to extend its durability it will be possible to add more features in future game updates, such as sound effects and the highscore table, as well as the bringing the possibility of making it an online game where you are not only able to view your local high score but also a worldwide high score table. But also features such as adding collectable items, such as coins or some kind of points which could be used to be able to purchase aesthetic items such as skins for missiles, enemies, asteroids and the spaceship. Power ups within the game may also be considered to be added, as to make the game more dynamic and entertaining for the player. All of this is possible if given more time to keep developing the game.

Maintainability wise, the code is written and ordered carefully, allowing for anyone who can access the game to easily find where code has been placed and its functions. Appropriately named variables have been assigned to both variables and functions, e.g. spaceship is the spaceship and enemy is the enemy. Additionally, by having separate files that link to each other, it reduces the amount of code that there is in each file, allowing for easier browsing of variables and settings, while also reducing the amount of global variables needed per file. This allows for easy maintenance, even if it is not by the creator themselves.

However, since pygame was used to code the solution and there are multiple files linked to each other, it can be hard to make it so that it is able to be transferred into a different language, and by adding more features it is possible that both the quality and performance of the game downgrades since python is not an extremely fast or efficient language, and by adding more features the game may start to crash by having too much information to process at the same time.

All of this game has been made by applying the aforementioned computational thinking techniques, as the problem has been **decomposed** into small tasks and parts, which were then carried out in each prototype successfully through **trial and error**, by testing out different ways to code the solution. Each test, success criteria and prototypes were planned out carefully, showing how it is important to **think ahead** and planning out the solution carefully.

Additionally, due to the nature of the game, **thinking logically/procedurally** was an absolute must when coding the developed solution, as code works through logic and a large part of the game runs on boolean logic since that is how the menus are constructed and controlled to work, as variables must be toggled through true and false for the game’s functioning, such as spaceship being alive, game over screen running, pause running and start screen running.

Finally, thinking concurrently was to be also considered when developing the coded solutions as all possible interactions between the entities inside the game must be considered and accounted for. This can include the collisions between the friendly entities, such as the missiles and spaceship, between the unfriendly entities, such as the asteroids and enemies.

## Future potential features

Due to the nature of this game, it would be possible to add more features in the future, to keep it updated and keep the current audience engaged while attracting new users simultaneously.

| **Number** | **Feature** | **Description** |
| --- | --- | --- |
| 01 | Power ups | Power Ups could be added in the future to make the game more interesting. This can be either to make the spaceship travel faster/slower. Make it shoot twice the missiles or have a protective shield for a given amount of time. |
| 02 | Collectable objects | Things such as in-game currency, e.g. coins, or rare collectables such as figurines and achievements, e.g. destroy 100 meteors in 10 seconds. |
| 03 | Additional aesthetic elements | Along with collectable objects such as coins, the user would be able to choose to purchase different spaceship, missile, asteroid, enemy appearances as well as different menus and colour pallets. |
| 04 | Background music/ sound effects | This was an intended feature for the first creation of the game but sadly could not be added due to time constraints and file incompatibilities. Hopefully in the future this can be added to assimilate more an original arcade game. |
| 05 | Scoring | Score display was also another feature that was to be originally added, however hopefully it can also be implemented in a future update of the game. |
| 06 | Online competition | The possibility along with scoring would be to enable a world wide leaderboard which would allow players to view the best players across the world |
| 07 | Two player mode | A dual player mode on the same device could also be implemented, such as fireboy and water girl where the arrow keys are used for one spaceship and the WASD controls used for the other. This could easily be accomplished through an update that would separate the two possible controls of the spaceship into two different sets of controls. |
| 08 | Levels separate from endless mode | As given by user testing feedback, levels are a thing which are wanted by many players, this was also previously backed up by the questionnaire that was carried out before the development of the game so it is a feature that could be added in the future having more time to keep developing the game. |
| 09 | Purchasable upgrades | Along with the collectable objects, boosts could also be potentially purchased along with the different aesthetic elements, as well as the possibility to create different ships that can be purchased with different abilities. |
| 10 | Additional obstacles/ enemies | More enemies could be added, with different interactions and moving patterns than the current asteroid and enemy, more enemy varieties could be added and as geometry dash modes as inspiration, different sections of the game could be changed to be more challenging as the player passes through a phase/portal. |
| 11 | Tutorial mode | We are aware that in the current game right now there are no indications of the actual controls for the game other than the start screen and game over screen, in the future we would like to add a small tutorial or screen that would indicate the player on how to operate the spaceship upon first opening the game. |

# 

# **Project appendixes**

Apps

<https://www.python.org/>

<https://www.pygame.org/news>

<https://code.visualstudio.com/>

<https://apps.microsoft.com/store/detail/paint/9PCFS5B6T72H>

<https://lucid.app/>

Sprites

<https://www.kindpng.com/imgv/iwbwoih_spaceship-sprite-sheet-png-transparent-png/>

<http://pixelartmaker.com/art/078cb44dc3cf9f1>

<https://www.pngfind.com/mpng/ibJwRhh_asteroid-hard-minecraft-pixel-art-hd-png-download/>

<https://animal-groups-roleplay.fandom.com/wiki/User_blog:Conservation/8-Bit>

Operating systems

<https://learn.microsoft.com/en-us/lifecycle/products/windows-7>

<https://www.microsoft.com/software-download/windows11>

<https://www.linux.org/>

1. <https://www.thetech.org/sites/default/files/techtip_computationalthinking_v3.pdf> [↑](#footnote-ref-1)
2. <https://www.bbc.co.uk/bitesize/guides/z4rbcj6/revision/3> [↑](#footnote-ref-2)
3. <https://languages.oup.com/google-dictionary-en/> [↑](#footnote-ref-3)
4. <https://www.merriam-webster.com/dictionary/trial%20and%20error> [↑](#footnote-ref-4)
5. At the same time; simultaneously [↑](#footnote-ref-5)
6. <https://en.wikipedia.org/wiki/Vanguard_(video_game)> [↑](#footnote-ref-6)
7. A system diagram is a visual model of the system, its components, and their interactions. [↑](#footnote-ref-7)
8. [https://www.softwaretestinghelp.com/what-is-sdlc-Waterfall-model/](https://www.softwaretestinghelp.com/what-is-sdlc-waterfall-model/) (date accessed 21/09/21) [↑](#footnote-ref-8)
9. [https://www.tutorialspoint.com/sdlc/sdlc\_Waterfall\_model.htm](https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm) (date accessed 21/09/21) [↑](#footnote-ref-9)
10. <https://www.lucidchart.com/blog/pros-and-cons-of-waterfall-methodology> (date accessed 12/11/21) [↑](#footnote-ref-10)
11. <https://languages.oup.com/google-dictionary-en/> (date accessed: 7/10/2021) [↑](#footnote-ref-11)
12. <https://airbrake.io/blog/sdlc/iterative-model> (date accessed: 12/11/2021) [↑](#footnote-ref-12)
13. <https://www.professionalqa.com/iterative-model> (date accessed: 10/12/2021) [↑](#footnote-ref-13)