OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name : Richard Yaya-Abatan

Candidate Number: 8164

Chobham Academy: 10487

Title of Project: Chickens Can’t Fly (CCF)

Title of project

Table of Contents

[A. Analysis 3](#_Toc98963829)

[Research 5](#_Toc98963830)

[Featuresof solution: 10](#_Toc98963831)

[Limitations**:** 10](#_Toc98963832)

[Success criteria**:** 12](#_Toc98963833)

[B. Design 13](#_Toc98963834)

[Systems diagram 13](#_Toc98963835)

[Usability features 15](#_Toc98963836)

[CCF Game Classes 19](#_Toc98963837)

[Game sections flowcharts 20](#_Toc98963838)

[Game functions flowcharts 25](#_Toc98963839)

[Main variables and data structures 27](#_Toc98963840)

[Test Table 28](#_Toc98963841)

[C. Developing the coded solution (“The development story”) 30](#_Toc98963842)

[Development of Game Screens 30](#_Toc98963843)

Game Pseudocode…………………………………………………………………………………………………………………………………………….32

[Code deconstruction 36](#_Toc98963844)

[D. Evaluation 49](#_Toc98963845)

Full Program Evaluation…………………………………………………………………………………………………………………………………….51

Input Validation………………………………………………………………………………………………………………………………………………58

[Project Appendices 59](#_Toc98963846)

# A. Analysis

**An outline of the problem:**

My project is a game inspired by the classic game of flappy bird, in which a character glides through the air, whilst avoiding pillars that function as obstacles. The game requires a single player to interact with the character by using the left click on the mouse to change its altitude, which would determine if the character can avoid the obstacle or not.

The game requires the character to have a hitbox that must not touch the pillars when the game is being played and if this occurs the player loses. Additionally, if the player does not maintain the altitude of the bird and hits the borders of the map, they also lose. It runs on infinitely and every obstacle passed is a point which is recorded on the score board. The game becomes progressively difficult, with pillars spawning in more frequently.

However, differently from flappy bird my game will involve the score being based on survival time rather than obstacle passed. This will make the game more intense as it increases the incentive to stay alive. As the game progresses, the frequency of the pipes will increase, this will be determined by the player’s score. This eliminates the feeling of repetition that users may have felt in the original flappy bird.

**Stakeholders:**

The game can be played on any computer devices that supports python, due to the simple controls that’s utilised. Limiting the game to one player removes any need for age restrictions as there is no commutation between players which may lead to the use of unpleasant language. The need for only one control means that it can easily be adapted for different devices in the future through the creation of useable buttons or tapping on the screen which would serve the same function as the controls on computer devices.

My stakeholders are teenagers, young adults, and children. The game is suitable for children because it is easy to use and does not contain any explicit content. Additionally, its childish interface with the use of animated sprites makes it appealing to children because it would feel familiar to them, it teaches them the concept of setting goals as they aim to beat their personal record on each round of the game they play.

It is also suitable for teenagers as it is an exciting game that rebirths an already well-known and popular game. The game is action packed and therefore provides the thrill teenagers often look for in games and general activities. It can easily become a regular activity because of the simplicity of the game yet

For young adults it is a nostalgic experience, and they might take interest in it to relive their childhood memories, they will also already be familiar with the game’s concept and controls so for most adults, this removes the skill gap that may exist.

An example of my target audience is a friend of mine who suggested the idea for a remix on the popular flappy bird. He is a video game enthusiast, who is also a computer scientist and programmer. He has experience in the arcade video game genre, so my project will not be alien to him. My target audience is a suitable one because he is a representative of my stakeholders, fitting into one of the three categories as a teenager. He can critique my project, allowing me to understand the areas that require improvements, which can then be worked on, allowing me to produce a more refined final project.

**How the problem can be solved by computational method:**

I can solve this problem through computational methods by first breaking down the problem into simple and approachable tasks that are easier to program, the end-product which is the game will be the most refined version I have produced, with each new variation building on the previous by fixing issues, adding new functions/features, and improving overall gameplay quality. Below is an explanation of how I intend to use computational thinking to meet my success criteria components and produce a tangible product:

**Thinking abstractly:**

This is the stage in which I decide what is important to the development of my program and what features are unnecessary/can be removed:

* + Removing unnecessary sound effects
  + Removing the background image
  + Removing the leader board
  + Add start menu for the game
  + Adding obstacles and creating the game borders
  + Add a score counter

These key features will make up my success criteria which will be used to determine the essential components of my game. By meeting my success criteria through the implementation of the listed features, I can ensure that the problem is solved computationally.

**Thinking ahead:**

This stage is about laying out what I plan to do next:

* + Using pygame to develop the game
  + Designing the menu
  + Developing what the character will look like
  + Finding out how to connect the user input to the character.

By using this computational method, I can predict the inputs and outputs my programs will consist of and how I plan on handling them, for example deciding which keys will serve as valid inputs in my game such as handling w button inputs as up instructions for my game sprite. This will allow me to work towards meeting my success criteria to come, more effectively because I will have accounted for factors that may limit the usability of my program.

**Thinking Procedurally:**

The structure I will use is the agile model, this consists of multiple subprograms that will be built on to produce the final program, this structure allows me to tackle errors I may come across easily and identify them quickly. Each iteration of my game which in practice is each session of program development will expand on previous sessions, I will start each “iteration” with an establishment of requirements and then proceed to follow the agile model cycle. This will allow me to focus more on meeting my success criteria and producing a functional product at the end of each iteration, rather than building the program in the linear structure of the waterfall model which may lead to problems such as bug fixtures because debugging would require analysis of the full program every time, I run into a issue. This model is also the most practical when working on a project with a time limit which I currently must because it allows the greatest output in the shortest time.

However, this model is most effective when working as a member of a team, but I am a sole developer so the agility of development is limited due to the lack of resources available to me, so features of the model such as feedback reflection cannot be fully utilised.

**Thinking logically:**

The logical aspect of the game includes the loops that are running and the conditions that are constantly being checked as the game is running.

For example, the program checks if the control button has pressed and if so, increase the height of the character. If an obstacle is hit the game ends and a game over image is displayed. If the user clicks the setting button the setting option, such as character customisation and sound control are displayed. The map the user plays on is also a loop which is critical to the user enjoyment of the game.

**Thinking concurrently:**

This allows me to understand what processes will be conducted simultaneously within my program. For example, the game must update the map, obstacles, and score all at once. Whilst this is happening the game will also be constantly checking for collisions between the game’s sprite and obstacles. The method allows me to determine which functions will have relationships in my program and plan out how I will make the connections between them.

**Conclusion:**

In conclusion these computational methods enable me to give my program structure and visualise how to solve the problem, it allows me to account for factors that may limit the functionality and success of my program and eliminate unnecessary features, it also provides a plan for how to successfully complete the project.

## Research

**Interview:**

Questions:

1. Visual/audio design:

* Would you favour a changing background?

If so, what do you think would be the most appropriate settings?

* For action games do you prefer upbeat or sombre music?
* Would you like the ability the change the character’s appearance?

1. Gameplay:

* Would you prefer for the obstacle to become more difficult as the game progresses?
* Would you like a level based or score-based game? – This would provide a twist on the concept of the game
* Do you think a game that does not end is more fun that an infinite one? – Flappy bird, the game that this is modelled after is known for being endless

1. Scoring:

* Should there be rewards linked to their score?
* Should there be a limit to the score achievable?
* Earlier I had asked if it should be level based or score based, based on your previous answer what is your favoured scoring system

Interview Script:

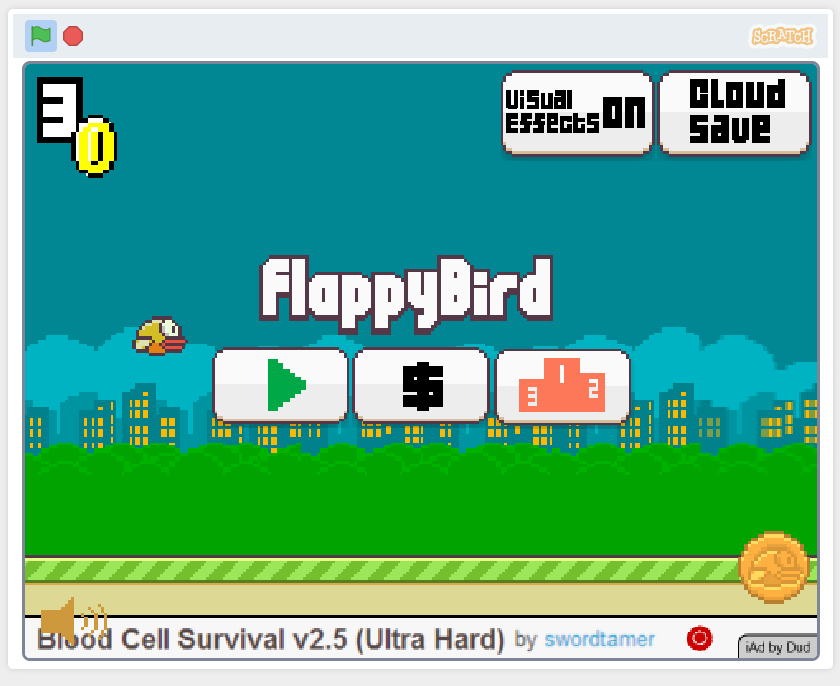
* 1. Would you favour a changing background? Yes, very much so, if so, what do you think would be the most appropriate setting? A farm that progresses to the city
  2. For action games do you prefer upbeat or sombre music? Upbeat music because it would add to the tense atmosphere of the game making the player more on edge whilst playing
  3. Would you like the ability to change the character’s appearance? Yes, I would like the ability to personalise my game character with customisations such as colour change or outfit change, the ability to change the type of character such as the species of bird would also be interesting
  4. Would you prefer for the obstacle to become more difficult as the game progresses? Yes, I would like obstacles to become more unpredictable as the game progresses, following on theme of flappy bird, unexpected obstacles such as other birds or objects
  5. Would you like a level based or score-based game? A level-based game would be more fun because it makes the game more challenging to play and provides the player with goals to achieve
  6. Do you think a game that does not end is more fun that an infinite one? If the game is level-based an end would be preferred because it provides a sense of achievement to the player, however if difficulty is score based then an infinite game would be more logical
  7. Should there be rewards linked to their score? Yes, there should because games with rewards retain players for longer as it is an incentive for playing
  8. Should there be a limit to the score achievable? If the game is score based, then yes there should be a limit to the score achievable because I as a player may get bored if it runs on infinitely.
  9. Earlier I had asked if it should be level based or score based, based on your previous answer what is your favoured scoring system? My favoured scoring system is one that is based on the events that happens during the game such as distance travelled, items collected, and obstacles avoided.

**Analysis:** The interview was a success because I was able to collect information on the types of adjustments that makes a game retain more users and attract inexperienced players. I was also able to find out the preferences of a potential user in terms of game ability and customisations.

**Review of game:**

Game Url: <https://sites.google.com/site/unblockedgameswtf/flappy-bird>

This game is a copy of the original flappy bird since the original was taken down by the creator for bad media attention.



The total number of golden coins collected is shown on the home screen.

A start, shop and high score button is shown on the centre of the home screen of the game

An animated “flappy bird” that if flying through the map is shown on the home screen.

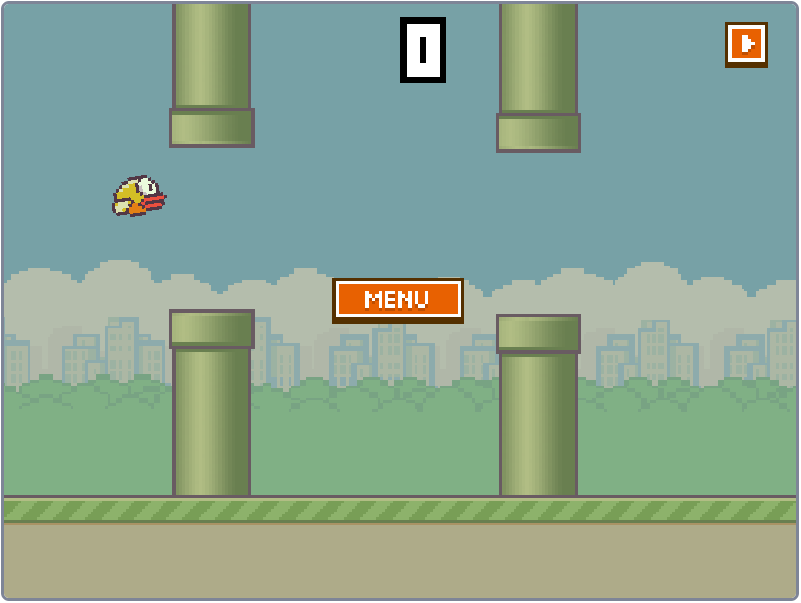
A button to toggle visual effects on and off can also be seen on the home screen.

The user is also given the option to save their game data to the “cloud”, which just saves the game data onto the pc the user is on.



Upon start the player is told how to play the game with a little hint.

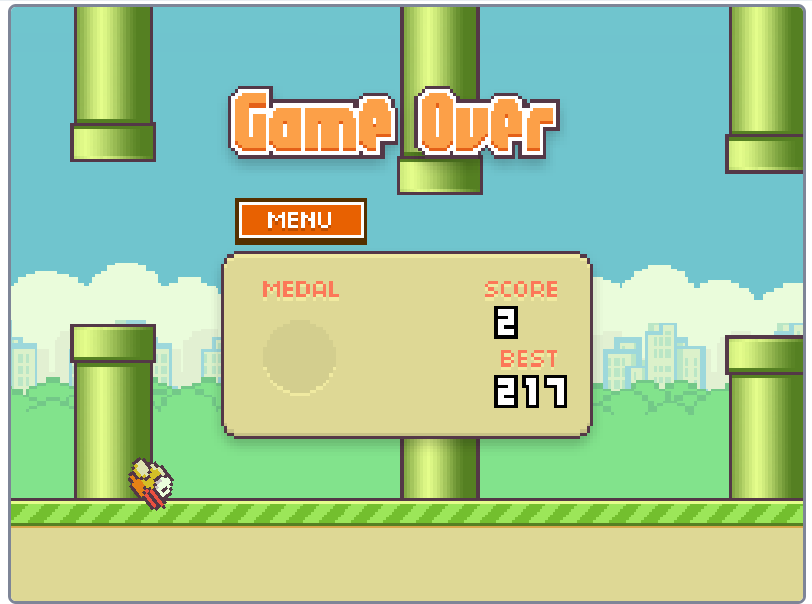
A big green text is also presented to the player to alert them the game is about to start



Current score is displayed at the top of the pause menu

The player can pause the game

When the player hits an obstacle or hits the ground, a Game Over menu is displayed



The total score is also shown on this menu along with the best score ever achieved

The total medals collected is also shown

Table

Description automatically generated

A button to return to the main menu.

The shop also contains items that gives the user advantages. The coin multiplier increases the value of each coin collected.

The shop contains all the items purchasable with the coins collected.

Text, qr code

Description automatically generated

The game records and displays the top two highest score ever achieved with the username of the players that got them.

## Featuresof solution:

|  |  |
| --- | --- |
| **Menu** | This shows the player the different sections of the game e.g., settings. It allows ease of navigation for the user |
| **Mouse control** | The main game controls are mouse based (click to make the character move.) |
| **Sound effects** | This will make the game more interactive as it provides responses for user actions |
| **Changing backgrounds** | This creates a sense of progression within the game as backgrounds would change as the game progresses |
| **Special/HOT Keys** | These improves convenience for the user, makes it easier to pause/un-pause and an alternate between w and the right click for up |
| **High score table** | Allows comparison of score between players and creates competition |
| **Game over screen** | Tells the player when the game is over |
| **Game Settings** | Allows the user to set their preferences for visuals effects and audio |
| **Random Obstacle generation** | This increases the intensity of the game |

## Limitations**:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time Limitations   |  |  |  | | --- | --- | --- | | Requirement | Justification | Limitation | | Changing background | It makes the game progressive and gives it a storytelling aspect. | Changing backgrounds would require development/designing of diverse backgrounds. | | High score table | Creates a competitive environment. | There is a time limitation on this because to create a high score table from different users, I will have to get multiple people to play the game which I do not have time for. | | Sound effects | This would make the game more reactive and would alert the player to objects in the game. | The project has a time limit placed on it so I may not have enough time to develop the sound effects. | | Character customisation | This would make the game more interesting for the user because it allows them to personalize their character | However, because of the time limit I may not have enough time to develop contrasting character skins. Additionally, I am not currently fluent with graphic development so I may struggle. | | Game shop | The user can use the coins collected from playing the game to purchase skin, this is an incentive for the player because they would want to play for longer to unlock more items/skins. | The time limitation means that I may not be able to incorporate this feature into the game as I will have to prioritise the completion of other tasks to meet the deadline. | |
| Hardware Limitations   |  |  |  | | --- | --- | --- | | Requirement | Justification | Limitation | | A mouse | The user will be able to use the mouse to control the character by clicking to make it “flap.” | The game will not be playable if the player does not have a mouse | | A computer | A computer is required to run the game. | Computers are more expensive than devices such as mobile phones and not all computer devices may be able to run it. | | A graphics card | The game will have animations that may not be detailed without a graphics card. | Visuals may be blurred or lack colour depth if graphic card not powerful enough. | | Keyboard | Allows the user to make use of hotkeys and special keys | These keys will not be available to the user if they do not have a keyboard and will therefore have to access them manually. | |
| Software Limitations   |  |  |  | | --- | --- | --- | | Requirement | Justification | Limitation | | Windows, macOS, Linux | These are operating platforms that will allow the user to run a python-based game. | This restricts the player base to only users on these operating systems. The game could be adapted for other OS, but I have yet to learn the skills for this and do not have enough time to develop this. | | Visual effects | This will make the game more immersive and fun because it adds depth to the game. | I might not be able to fully develop these as I do not have a graphic design background. | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Requirements**   |  |  | | --- | --- | | Hardware | Justification | | **A computer device** | The game is only computer-based meaning only works on computers | | **Standard Keyboard** | This must be English and contain the letters W, P, S and Esc. These are the specials keys the game requires. | | **A mouse** | The game’s main input is a right click on a mouse. |  |  |  | | --- | --- | | Software | Justification | | **Python Idle** | This will let the player execute the game because it is python based and this is needed to run python code. | | **Pygame** | A cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. | | **Windows OS** | Supports Python based applications. | | **macOS** | Supports Python based applications. | | **Linux OS** | Supports Python based applications. | |

## Success criteria**:**

High priority =

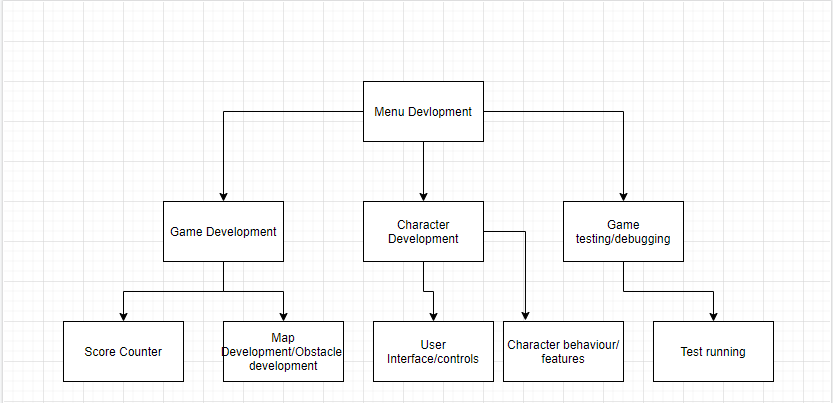
Low priority =

* Menu (SC1): This feature is essential to the game because it allows the user to easily navigate through the game and it acts as a hub for the key features of the game. Such as game settings, shop, and high score table
* Mouse control (SC2): The main game control is mouse based and the game may be more difficult to play if the mouse cannot be used and it.
* Object/Sprite creation (SC3): The creation of objects within the game such as obstacles and the bird itself will allow me to successfully create the game because these objects have perimeters which must not touch e.g., the bird object touching the obstacle object. The actual game is a combination of different objects, so it is important this feature is added.
* Score counter (SC4): It will allow the user to keep count of their current score which will represent the user’s progression/goals.
* Creating borders (SC5): This creates a game zone for the player that cannot be exited and leaving this game zone would result in the players death. Boundaries are created around objects/sprites that aren’t meant to interact when the game is played. With this I can check for collisions.
* Backgrounds (SC6): The backgrounds will add a storyline aspect to the game and acts as a progress indicator for the player as the background may change when the difficulty of the game increases.
* Keyboard controls (SC7): The ‘w’ key will function as an alternative for the right click and will provide the player with the option to choose between the controls, they are more comfortable with. The other controls such as the ‘m’ key to allow the player to easily mute game audio.
* Obstacle generation (SC8): The spawning of obstacles whilst the game is running create confusion for the player making increasing the game difficulty.
* Tutorial (SC9): Because the game is targeted at a spectrum of audiences a tutorial will be important to understanding the game
* Game coins (SC10): This creates a reward and collectible system within the game because the coin can be used to purchase different skins which will be part of the players collection, at the same this acts as an incentive for playing the game.
* Game shop (SC11): The game shop will contain the customisations available to the player. This will make the game more fun because it allows for user personalisation.
* Game settings (SC12): This is essential to the game as users may have adjustments that have to make before they are able to play the game.
* High score table (SC13): This will make the game competitive as players are able to compare results and set personal goals
* Sound effects (SC14): Alerts the player that their click was registered and makes the character more interactive.

# B. Design

## Systems diagram

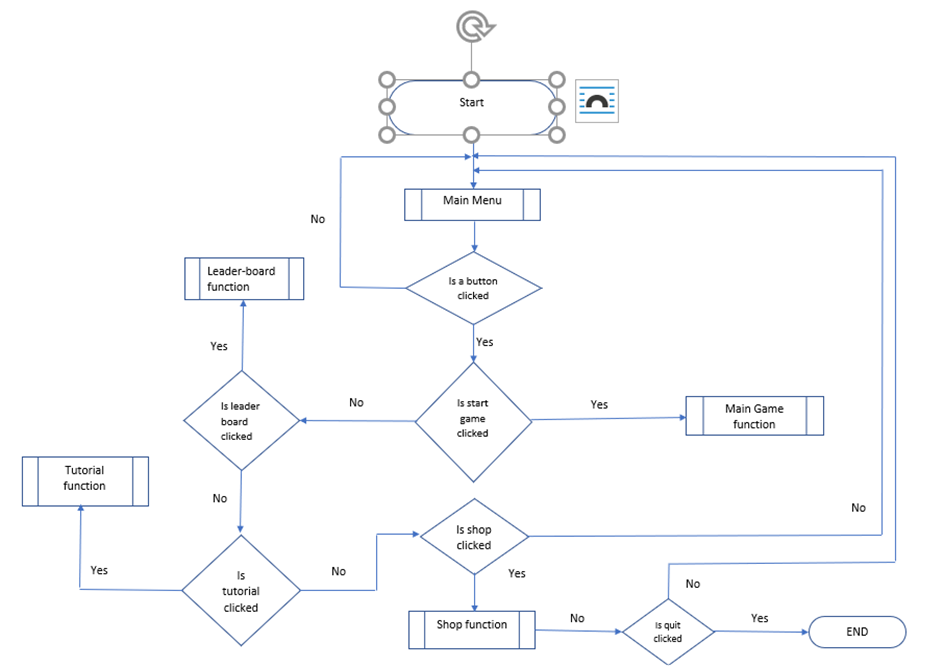
The systems diagram allows me to model what would otherwise be a complex system of interconnected functions, with the use of a systems diagram I am able to map out the relationship between these functions and understand how they will work together. By establishing relationships, I can structure my development around them, creating solutions in order of said relationships.



**Why I used the systems diagram:**

The systems diagram also acts as a broken-down structure of the project. I can then implement this model into the agile model which will allow me to build my game in progressive prototypes that means at the least I will have a product to present to the examiners. This is the best development model to use in my software model because other models may lead to me not being able to produce a program if I run out of time such as the waterfall model which builds a program by sections.

**Game flowchart:**

****

**Menu development:** Menu development is the starting screen of the game; the user is presented with the option to start the game, customize their character, or access any additional settings that may be available to them.

**Game development:** This includes the maps the character/ user will play on and sees the creation of obstacles on the map which will increase in difficult the longer the player survives, and the period the player survives is based on the score counter which is also developed at this stage. Every time the score reaches a certain number the map will increase in difficulty and the frequency of obstacles will increase.

**Character Development:** At this stage I will develop the features of the character, such as the appearance and customizable. This stage will also include the development of character behaviors such as ending the game if the character touches an obstacle or decreasing the character altitude by 5% of the monitor size every time the user does not interact. To prevent the game ending I will add a button which will allow the user to interact with the character by increasing the character altitude by 5% of the monitor size every time the button is pressed.

**Game Testing:** This is the final stage of my game development where I assess for any bugs or errors that may cause the game not to function as intended. I must evaluate the integrity of my program, measuring it against my success criteria which determines whether my program was as stated in my plans.

## Usability features

Diagram

Description automatically generated

Allows the player to check the highest scores achieved by different players.

Starts a new game.

Displays the title of the game.

A settings button directs the user to where they can toggle their game settings.

Allows the player to view the item store/ customisations available to the player.

It explains the game making it easy for inexperienced players to understand how to play.

A picture containing text, chicken, bird

Description automatically generated

Allows the player to toggle the visual effects of the game on/off.

Allows the player to toggle the sound effects of the game on/off.

Returns the player to the main menu of the game.

Displays the game credits/ creators of the game which is just me (Richard Yaya-Abatan).

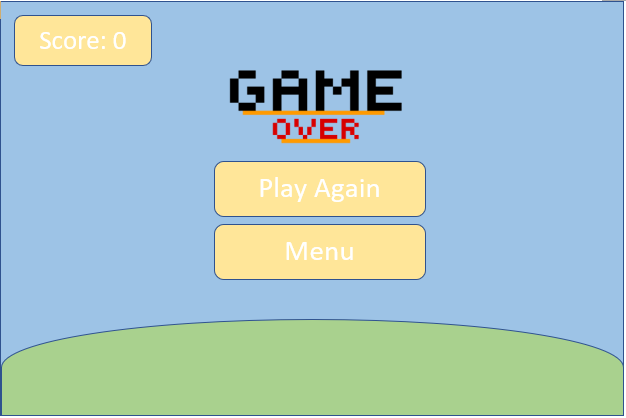
Diagram

Description automatically generated

Ends the current game the player is in and returns them to the main menu.

Takes the player to the game settings whilst keeping the game on paused.

Returns the player to where they were before they paused the game.



Returns the player to the main menu of the game.

Displays the total game score at the time of death.

Restarts the game for the player.

Alerts the player that the game is over they have died.

Diagram

Description automatically generated

Returns the player to the main menu of the game.

Displays the top scores achieved by the game’s players.

A picture containing diagram

Description automatically generated

The pillars/obstacles spawn from portals in the ground and the sky.

Displays the current game score.

Allows the player to pause midgame.

Diagram

Description automatically generated

This informs the user on how the game is played.

## CCF Game Classes

Game surfaces

Window size

backgrounds

game\_floor

game\_bird/sprite

ccf\_pillar

Game Background

Main Background

Menu Background

Game Obstacle sprite

Tutorial Background

Game floor

Import menu, main, shop, tutorial background

Move\_floor()

Obstacles

Pillars/Fences

Game border

draw\_pipes()

move\_pipe()

Objects on screen

Game Bird

Game Backgrounds

Game Obstacle

Build\_floor()

Game Pillar/Fence

Generate Fence

Place fence on screen

Random.choice(fence height)

Set Fence (x,y)

create\_pipe()

Score box

Game score

High score

Display\_score()

High\_score()

Write Highscore to file

Sprite

Bird sprite

Bird movement

Bird collision

Check\_collisions()

()

Bird Movement

Gravity

Bird x,y

Bird\_movement

Set Bird (x,y)

Like the systems diagram I built previously; my class diagram is a blueprint of my program. Through the construction of the classes diagram I can associate relationships between functions and model the objects my program will consist of. This will be useful for structuring my program and incorporating the agile model development model I plan on using to build my program.

## flowcharts

**Main menu flowchart**:

Diagram

Description automatically generated

This will be used for completing my first success criteria – the menu function. I will use the flowchart as the layout for my menu function. Here I have made sure to include the potential structure of this function, explaining how each component of the function is connected by a series of checks carried out by the program through the use of if statements.

**Game flowchart (Part 1):**

Diagram

Description automatically generated

This flowchart explains how the actual game works when the user is playing. It depicts the concurrent process that occur when the user is playing, but due to the linear structure of a flowchart, I have also had to adapt the process to a linear structure. The flowchart ties into the success criteria of objects creation as it explains how the user’s action via key inputs, play on the bird object I create.

Diagram

Description automatically generated

**(Part 2)**

Part 2 builds on the first part, showing the order of which objects are created in a linear form, outside of the flowchart the processes carried out within this flowchart would represent one cycle.

**Game settings flowchart:**

Diagram

Description automatically generated

Although, game setting isn’t a high priority success criterion so it may not be incorporated into my game due to the time limitation. I can use this flowchart to develop a game setting in the future by simply using the flowchart as a plan.

**Game tutorial flowchart:**

Diagram

Description automatically generated

The tutorial flowchart is simple because my tutorial will be simple since the game is easy to play. During development I plan on using a image for the tutorial screen, this means the tutorial screen will be a still image that explains how the game is played. It checks for simple button checks for returning to the main menu. Although it isn’t a key success criterion, the simplicity means I can quickly incorporate it into my program.

## Game functions flowcharts

**Check collisions function:**

Diagram

Description automatically generated

This flowchart is a visual representation of my border creation success criteria. It explains how I check for collisions between objects through the use “rects” placed around each object which means I can check for collisions by confirming matching x and y coordinates of the “rects”.

**Game high score flowchart:**

Diagram

Description automatically generated

The flowchart shows the process of which a new high score is determined, the score obtained by a user is compared against the current high score, if it is discovered that this score is greater than the current high score it becomes the new high score. The user’s score and high score-if a new high score is set, will be stored in a flat file database, this data will be used to assist my post development plans of creating a leader board of scores.

**Trace table**

Here I trace the outcome of a few iterations of my high score function, assuming that the previous high score is 9 and the user loses when they reached a score of 10. The trace table displays the process the high score function follows in determining a new high score through comparison of a value it already has against the value the user determines by how long they survive.

|  |  |  |
| --- | --- | --- |
| Game score | High score | Output |
| 0 | 9 | 10 |
| 1 | 10 |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |

## Main variables and data structures

|  |  |  |
| --- | --- | --- |
| **Variable name** | **Data type** | **Variable purpose** |
| Game\_score | integer | It records the current score of the game, initial game score is zero as the game would starts over every round. |
| High\_score | integer | It records the game high score based on whether the game score is greater than the current high score, initial high score is nine to make the game competitive. |
| Fence\_x\_position | integer | This holds the x position of the fence object; it is required because for the object to move across the screen the x position must change. |
| Ground\_x\_position | integer | This holds the x position of the ground object; it is required because for the object to move across the screen the x position must change. |
| clicked | Boolean | Allows for checking when an event happens such as the user clicking a button. |
| Game\_font | string | It sets the font that is used in the game for writing text. |
| Main\_background | string | This will allow me to import the background that will be used from the game folder and put it on the screen. |
| Game\_sprite | string | This will allow me to import the image that will be used as the game sprite from the game folder and put it on the screen. |
| Fence\_heights | array | This is the list of heights that will be used by a random generator to alternate the height of the obstacle-(fence) in the game. |
| Screen\_size | string | Initialises the size of the game window. |
| clock | integer | The clock variable is used to initialize the frame rate of the game (fps) using the system clock. |
| Fence\_sprite | string | Imports the image that will be used as the fence obstacle from the game folder and put it on the screen. |
| Menu\_button | string | This creates the menu button and attaches what will be the menu function to the button. |
| Tutorial\_button | string | This creates the tutorial button and attaches what will be the tutorial function to the button. |
| Settings\_button | string | This creates the settings button and attaches what will be the settings function to the button. |
| Mousepos(x, y) | integer | Finds the x and y position of the mouse, this way the mouse position on screen can be checked for things such as collisions. |
| Game\_state | string | Allows for checking when an event happens such as the user clicking a button. |
| Gravity | Float | The gravity value is the rate at which the y position of a sprite reduces, making the sprite appear to fall. |
| Bird\_movement | integer | This the rate at which the game sprite moves when the space bar or w button is pressed. |

## Test Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test subject** | **Success Criteria** | **Normal data test** | **Erroneous data test** | **Expected outcome** |
| Game menu | Menu allows the user to easily navigate through the game and it acts as a hub for the key features of the game. | Pressing the button works and runs the menu function.  The menu works and displays the game options such as tutorial and settings and these are also functional. | Checking that pressing outside the button has no unintended effects.  The menu button only runs the menu function. | Pressing the menu button would result in the menu function being ran. |
| Score counter | It will allow the user to keep count of their current score | Counter goes up by one when the game is being played and the player is alive. | Running the game and checking the counter does not go up by any other value except one. | The score is incremented every time a check occurs and there are not any collisions. |
| Game borders | A game zone for the player that cannot be touched, touching this game zone would result in the players death. | Hitting the game roof and floor with the game sprite to check if the game ends. | Hitting other objects to see if they have the same effect as hitting the game borders. | The game borders are the same as the window border, when the game is running if this encounters the sprite the game ends. |
| Keyboard controls | The keyboard control “w” will serve the same purpose as the right click which is used for actual gameplay, clicking the bird | Checking the “w” button works and makes the chicken “fly” when clicked | Pressing other button to check if they unintentionally carry out the same function as the “w” button | The “w” and “spacebar” button can both be used to play the game. They change the sprite’s altitude when clicked. |
| Mouse control | mouse left click collision with objects will lead to responses; The main game control is mouse based. | Mouse clicks are recognized by the program and consequent events occur for significant clicks. | Checking that only the left click has responses. The right click will not be assigned any roles so it should not have consequent responses. | Mouse clicks are commands that initialize whatever function on which they are clicking. |
| Tutorial | The game is targeted at a spectrum of audiences a tutorial will be important to understanding the game | The button works and runs the tutorial function  The return to menu button on the returns to the Main menu | Check if the tutorial button is running any other functions. | The tutorial can fluently translate how the game works to a player with no experience that relates to the game. |
| Game settings | Users may have adjustments that have to make before they are able to play the game. | Game settings buttons are responsive and carry out an action when toggled. | Verifying the buttons do not make more modifications than the user requested. | The buttons in the settings make modifications to the game to suit the user’s preferences. |
| Backgrounds | The backgrounds will add a storyline aspect to the game. | The backgrounds have a meaning/storyline to it, there are multiple backgrounds that are functional. | wrong image is displayed, images do not change | Multiple functional backgrounds that change depending on the game screen, has some level of depth to it and adds a storyline aspect to the game |

**Post-development Data:**

The user’s score will be saved to a flat file database in the form of a text file. The score will be placed in ascending order with the credentials of each player attached to their score. This will form a leaderboard which I can transfer into my game. The leaderboard can then be viewed on the leaderboard page I plan to create, where the user can check their position on what could be either regional or/and global leaderboards. This would make the game more competitive because interaction between players in different regions is possible by competing for the highest score, it also allows players to compare their skills against that of the people around them.

# C. Developing the coded solution (“The development story”)

## Development of Game Screens

I developed my game screens using word’s shapes library, it was filled with all the shapes suited for the graphics I needed for my game as it was not that complicated.

=Errors/testing = Links to success criteria

Graphical user interface, diagram

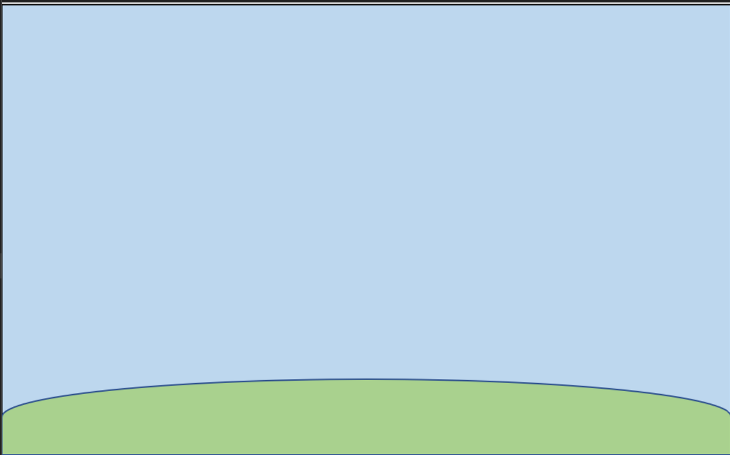
Description automatically generated

The background itself was made by stretching a rectangle and changing its fill colour. This accomplished my success criteria of a game tutorial.

The bird was a 3d model from word’s 3d model library

The obstacles were created by combining different sized rectangles from word’s shapes library

The clouds in the background were drawn on a website called pixilart, the image was then imported to my word document.



This is one of two backgrounds, it was made using an ellipse and rectangle, the ellipse is partially cut of to represent the curvature of the earth. The rectangle covers the entire screen and is filled with one colour. I had previously stated in my success criteria that I wanted my game to have multiple backgrounds that would create a storyline effect. By having two different backgrounds, I can alternate them to show progression thereby building a storyline effect.

A picture containing text, bird

Description automatically generated

Here is the second background, it is for the menu, and it is the same as the first except it has grass growing on it to represent time passing when the game is not running. It was also made in word.

This assists the completion of my menu function, the first task of my success criteria SC1.

The grass was drawn on using word’s draw feature.

The chicken is a 3d model from word’s 3d model library

## Game pseudocode

Set score to 0

Set high score to 9

Set clicked to false

Set soil x position to zero

Set delay counter to zero

Set game font to Constantia

Import backgrounds, character sprite, obstacle sprites

Initialise system

Function build floor

Display floor sprite at screen position (x position, 560)

Display floor sprite at screen position (x position + 900, 560)

Function create pipe

Pillar\_height equal to random selection of pillarheights

Base\_pipe equal to bottom of screen

Apex\_pipe equal to top of screen

Return Base\_pipe and Apex\_pipe

Function move\_pipe (pipes)

When game is running move pipe x position to the left eight pixels each cycle

Function draw pipes(pipes)

If the bottom of the pipe is greater than or equal to six hundred

Display pipe

If not rotate the pipe

Display pipe

Function check collisions(pipes)

If bird collides with pipe

Return false

If bird touches top of screen or bottom of screen

Return false

If not

Return true

Function tutorial screen

Import tutorial background

Set background to tutorial background

Display tutorial background

Create menu button

If menu button is clicked return to main menu

Set clicked to false

If user quits game

Stop system initialisation

Display menu button

Function main menu

Display main menu background

Create start button

Create tutorial button

If start button is clicked

Run game loop function

If tutorial button is clicked

Run tutorial screen function

Display start button

Display tutorial button

Function high score(score, high score)

If score is greater than high score

Set high score to score

Return high score

Function display score(score, high score)

If player alive

Display score

if player is dead

Display score

Display high score

Function game loop

Set gravity to 0.5

Set game running to true

Set bird movement to zero

Set score timer to zero

Set delay counter to zero

If user quits game

Stop system initialisation

If w button is clicked

Set bird movement to -8

If space button is clicked

Set bird movement to -8

If menu button is clicked return to main menu

Display main menu background

if game running is true

Set game movement += gravity

Display bird

Draw pipes

If delay counter >= 70

If check collisions equal true and score timer equal 120

Score += 1

Score timer += 0

else score timer += 1

display score(score)

else delay counter += 1

soil x position -= 1

Else high score equal high score(high score, score)

Display score(score)

If menu button is clicked return to main menu

Draw menu button

build floor

if soil x position <= -900

soil x position = 0

Call main menu function

This will form a complete solution because the removal of language specific syntax and structures means that I can easily adopt the pseudocode to the programming language of my choice with little work required. This reduces the workload of my project and makes sure all areas of my success criteria are met before beginning the coded solution.

## Code deconstruction

The library I chose for the development of my game was the pygame library, I thought it was very well suited for the task at hand because the pygame library allows you to manage graphics, collisions, and player input. The aspects of the library were all key components to the game I had in mind, essentially building blocks for what would become the game. Additionally, the library has a large user base, this meant that I would easily be able to find help from this community for any problems I came across.

import pygame, sys,random

from time import \*

from pygame.constants import MOUSEBUTTONDOWN

\*This gives me access to the pygame, sys and random module by importing it into my game environment, this means that I can then use the components of the libraries without any issues.

The creation of an environment means I can contextualise the theme behind my game allowing players to easily understand the concept of my game, this plays into the tutorial success criteria as a well-developed environment will help player better understand the topic of my program and even how to use it without specific instructions.

**Variable definition**

game\_score = 0

gamehigh\_score = 9

delay\_counter = 0

clicked = False

\*Here I set the variables that are used across my program and require assignment before being called.

These variables will be used in the score function I will later create, this will allow me to store current scores as the game is running and also compare the high score to the preset high score enabling me to meet the key success criteria of a score counting system.

**Screen initialisation**

pygame.init()

screen= pygame.display.set\_mode((900,600))

clock = pygame.time.Clock()

Text

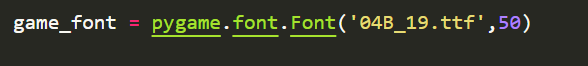
Description automatically generated\*pygame.init() initialises all the pygame module I imported from the library. At this stage I also set the size of what will be the game window and I also create a variable aptly named “clock”; this just calls the clock module which will allow me to use the system clock.

I encounter my first development error here; this was a simple logic error and was cause by a spelling mistake. I wrote pygame.innit() rather than pygame.init(). This meant that pygame was not initialised and therefore I could not use the pygame modules.

**Assigning game font**

game\_font = pygame.font.SysFont('Constantia',40)

Text

Description automatically generated\*I assigned the font I will be using for text in my game, this font was called from the system library of fonts. Initially I had imported a font I downloaded from the internet to use: But I had an issue where even though the file that contained the fonts I wanted to use, and my game file had the same directory the font file could not be found.

To fix this I resulted to using

a font that was readily available “Constantia” and set the font size to “40”

**Importing backgrounds**

main\_background = pygame.image.load('pyproj/gamefile/mainbg.png').convert\_alpha()

background = pygame.transform.scale2x(main\_background)

menu\_background=pygame.image.load('pyproj/gamefile/secondbg.png').convert\_alpha()

mainmenubg = pygame.transform.scale2x(menu\_background)

\*Here I import the images I will use as my backgrounds, of which there are two. One for when the game is running and the other for the main menu. The “pygame.image.load” function imports this image from the folder it is saved in so it can be used in my program and the “pygame.transform.scale2x” function resizes the images to fit the size of my game window. The varying backgrounds will add the storyline aspect I was keen to add to my game as a feature of my success criteria.

When doing this I ran into the issue of the images failing to export, this was due to the fact that the image files had separate storage locations to the game file.

Text

Description automatically generated

**Importing game soil/floor**

game\_floor = pygame.image.load('pyproj/gamefile/soil.png').convert\_alpha()

game\_soil = pygame.transform.scale(game\_floor,(600,20))

game\_soil = pygame.transform.scale2x(game\_soil)

soil\_xpos = 0

\*Imports the image that is used as the game floor and resize it to scale with the game backgrounds. Unlike the previous images I imported this requires a location as it will be at the bottom of the screen. Here I also assign the variable “soil\_xpos” the value “0”, this is used to make the game floor appear to move by changing the x position of it in a loop. The image file used as the game soil builds on the game’s backgrounds helping to emphasise the storyline aspect listed in the success criteria. The initial image used as the game soil wasn’t compatible with the game background, the colour scheme of both image files were too similar which meant that it was difficult to differentiate between the floor and the background, due to the game soil being a restrictor of player access as the player must not touch it whilst playing, it is important to ensure that users can tell the difference.

Graphical user interface

Description automatically generated with low confidenceInitial game soil:

Final Game soil:

A screenshot of a computer

Description automatically generated with medium confidence

**Importing Game Sprite/Character**

game\_bird= pygame.image.load('pyproj/gamefile/flyingbird.png').convert\_alpha()

game\_bird = pygame.transform.scale(game\_bird, (120,70))

bird\_rect = game\_bird.get\_rect(*center* = (100,300))

\*Imports the image that will be the game sprite which is an example of a game object I previously idealised, scales the image to the size I have set “120” width and “70” height then creates a rectangle around the image which will allow me to check for collisions. The image is then positioned on screen with center alignment.

**Importing Game obstactle/fence**

ccf\_pillar = pygame.image.load('pyproj/gamefile/grassyfence.png').convert\_alpha()

ccf\_pillar = pygame.transform.scale(ccf\_pillar, (70,350))

pillar\_list = []

PILLARSPAWNER = pygame.USEREVENT

pygame.time.set\_timer(PILLARSPAWNER,1000)

pillarheights = [250,350,450]

\*Imports the image that will be used as the game obstacle and transforms the image to the size set.

An empty list called “pillar\_list” is created, this will be used to store the pillar height picked by a random generator from the list of “pillarheights” which are just predefined heights I have selected that will give the pillars variation and make the game more intriguing.

“PILLARSPAWNER” checks if this pygame event occurs and “pygame.time.set\_timer” sets a timer of 2 seconds is set so that every 2 seconds the event “PILLARSPAWNER” happens.

**Build floor function**

*def* build\_floor():

    screen.blit(game\_soil,(soil\_xpos,560))

    screen.blit(game\_soil,(soil\_xpos+900,560))

\*Here I am putting the game floor surface onto the screen with “screen.blit”, two of these images are used with different x positions, this allows me to create the illusion of an endless floor as without this regardless of how much I stretch the image on the x axis, moving it along the screen on the x will lead to the image disappearing when the game is being played. This way I can loop two images of the floor that would make it appear endless.

This meets my success criteria of an active background because this function builds on the idea of my background having a storyline effect behind it due to the motion created from the games floor moving along as the game is running signalling progression.

**Create pipes function**

*def* create\_pipe():

    pillar\_height = random.choice(pillarheights)

    base\_pipe = ccf\_pillar.get\_rect(*midtop*=(1000,pillar\_height))

    apex\_pipe = ccf\_pillar.get\_rect(*midbottom*=(1000,pillar\_height-200))

    return base\_pipe, apex\_pipe

\*To meet my success criteria of obstacles that have respected areas, this function creates the game obstacle “pillars” with random heights generated from the list of predefined heights “pillarheights”. Each time this function is called, duplicate “pillars” are drawn with one inverted to function as the top “pillar” and with a difference in height of “pillar\_height-200” to leave a gap between for the sprite to travel through. Although this does not completely meet my success criteria of random obstacle generation, I have still managed to incorporate randomisation into the code.

**Move pipes function**

*def* move\_pipe(*pipes*):

    for pipe in *pipes*:

        pipe.centerx -= 8

    return *pipes*

\*This function changes the game pillar obstacle’s x position, each iteration of the loop the game pillar image is moved eight pixels to the left. Thus, creating the illusion of the game sprite approaching the pillars when the game is running. This also plays into the success criteria of my background having a storyline because it creates the illusion of motion helping to build the effect of the game being active.

**Draw pipes function**

*def* draw\_pipes(*pipes*):

    for pipe in *pipes*:

        if pipe.bottom >= 600:

            screen.blit(ccf\_pillar,pipe)

        else:

            rotatepipe = pygame.transform.flip(ccf\_pillar,False,True)

            screen.blit(rotatepipe,pipe)

\*The game pipes are created based on the “pillar\_height” if the pillar height is “>=600” that is if this is the bottom pillar then the pillar is displayed on the screen surface. If the “pillar\_height” is less than 600 however, the pillar image is flip vertically in “pygame.transform.flip” to make the top pillar which is then displayed on the screen surface.

**Check collisions function**

*def* check\_collisions(*pipes*):

    for pipe in *pipes*:

        if bird\_rect.colliderect(pipe):

            return False

    if bird\_rect.top<= -100 or bird\_rect.bottom>= 560:

        return False

    return True

\*In this function I am checking for collisions between the bird sprite’s rectangle and the game pillar’s rectangle, if this occurs then false is returned. I am also checking if the game sprite hits the game borders (game ceiling and game floor) if so, false is also returned, if neither of this happens then true is returned because it creates a game zone that the player cannot exit as mentioned in my list of requirements to making the game a success. This allows me to check the game state and determine whether the player has died or is still alive, the function returning true means that no collisions have occurred and vice versa.

**Game loop function**

*def* game\_loop():

    global soil\_xpos,pillar\_list,game\_score,gamehigh\_score,clicked

    delay\_counter = 0

    gravity = 0.5

    bird\_movement = 0

    game\_running = True

    score\_timer = 0

    while True:

        clicked = False

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_w:

                    bird\_movement = 0

                    bird\_movement -= 8

                elif event.key == pygame.K\_SPACE:

                    bird\_movement =0

                    bird\_movement -= 8

            if event.type == PILLARSPAWNER:

                pillar\_list.extend(create\_pipe())

            if event.type == MOUSEBUTTONDOWN and menubutton.collidepoint((mx,

my)):

                main\_menu()

            if event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    clicked = True

    #end of for loop

        screen.blit(main\_background,(0,0))

        if game\_running:

            bird\_movement += gravity

            bird\_rect.centery += bird\_movement

            screen.blit(game\_bird,bird\_rect)

            pillar\_list = move\_pipe(pillar\_list)

            draw\_pipes(pillar\_list)

            if delay\_counter>=70:

                if score\_timer == (60) and check\_collisions(pillar\_list)==True:

                    game\_score +=1

                    score\_timer = 0

                else:

                    score\_timer += 1

                    display\_score('main\_game',game\_score)

            else:

                delay\_counter+=1

soil\_xpos-=1

        else:

            gamehigh\_score = high\_score(gamehigh\_score,game\_score)

            display\_score('game\_over',game\_score)

            mx, my = pygame.mouse.get\_pos()

            menubutton = pygame.Rect(300,400,200,75)#Tutorial

            if menubutton.collidepoint((mx, my)):

                if clicked:

                    main\_menu()

            pygame.draw.rect(screen, (255,217,102), menubutton)

        build\_floor()

        if soil\_xpos <= -900:

            soil\_xpos =0

        game\_running= check\_collisions(pillar\_list)

        pygame.display.update()

        clock.tick(60)

A screenshot of a computer

Description automatically generated with medium confidence\*This is my main game loop, I decided to make this a function so I can easily call on it when making other sections of my code. I made the variables global because it meant I would not have to pass it as a parameter which I had initially done but it caused errors, I think this may have been because I was passing too many variables.

This led to problems such as the game floor being stationery when the game was running, this removed the aspect of all the objects being in motion when the game is running, I then defined all my local variables.

Within the game loop clicked is redefined as “False” to make sure no other definitions is referenced. I use “for event in pygame.event.get() ” to check if any of the listed events happen, these are if the user “QUIT” the pygame module is exited “pygame.quit()” then the python system is exited “sys.exit()”; it then checks if the event is “pygame.K\_w” or “pygame.K\_SPACE”, this is checking if the “w” or “space” button is pressed and if so the game sprite “game\_bird” is moved up by (“bird\_movement-=8”) 8 pixels instead of down because the pygame module inverts the y axis values with “0” being the top of the screen, I did this because I wanted to provide alternative buttons for the user so they can make use of the keys they are more comfortable with as mentioned in my success criteria; the event “PILLARSPAWNER” checks if it is referenced which it was previously, to spawn every 2 seconds; the final check is to confirm if the menu button is clicked.

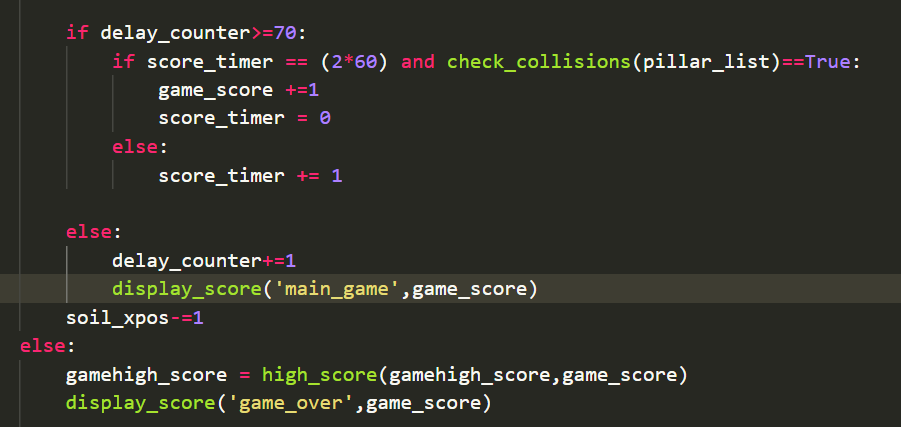
Whilst the game is running the bird sprite’s position is changed by the value of gravity every time the screen is updated, this makes the sprite appear to be falling as if affected by gravity. This sprite is then drawn onto the screen surface with “screen.blit” along with the pillar obstacle being initialised by being drawn and moved with its respective functions.

Due to the delay before all the objects are put on the screen, I made a delay counter for my score system that waits 1.16 seconds or 70 fps before beginning the score counting. The score system checks every 2 seconds (“if score\_timer ==60”) if there are any object collisions (“check\_collisions(pillar\_list)==True”) if not then the game score is incremented by 1 (“game\_score+=1”) and displayed, if there are any object collisions both the game score and high score are displayed. However, the score display system had an issue where after 2 seconds of the game running the score would disappear off the screen:

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

This was caused by my accidental placement of the code to display the score within the external “if” loop: rather than in the nested “if” loop:

Text

Description automatically generated

When the player dies, that is when a collision occurs, I get the mouse cursor’s x(“mx”) and y(“my”) position so to check for collisions with the menu button I created using “menubutton.collidepoint”. When this happens the “main\_menu” function is executed consequently. I get an event’s occurence with “pygame.event.get” which constantly checks if the two elements collide and simultaneously if the mouse button is down while the “game\_loop” function is running, “clicked” which was initially set to “False” changes to “True” to confirm the collision occurred. The menu button is drawn onto the screen using “pygame.draw.rect” and the screen is constantly updated-(“pygame.display.update”) at 60 fps(“clock.tick(60)”) to keep the window up until the function ends.

At the end of this function the game floor is initialised with its function (“build\_floor”) being called and an if loop to check if the image is completely out of the screen (“if soil\_xpos<=-900…soil\_xpos=0”) before being reset.

**High score function**

*def* high\_score(*game\_score*,*gamehigh\_score*):

    if *game\_score*> *gamehigh\_score*:

*gamehigh\_score* = *game\_score*

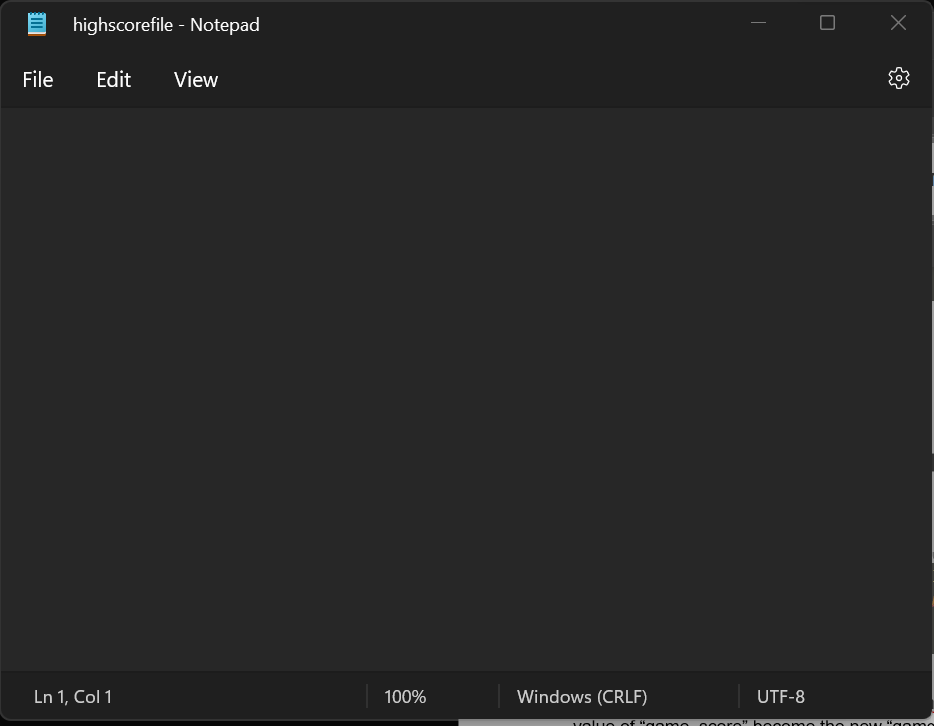
        scoretable = open("highscore\_file.txt", 'w')

        scoretable.write(""+str(*gamehigh\_score*))

        scoretable.close()

    return *gamehigh\_score*

\*The function passes the predefined variables “game\_score” and “gamehigh\_score” which have initial values 0 and 9, respectively. When “game\_score” is greater than “gamehigh\_score” the value of “game\_score” become the new “gamehigh\_score”. I then attempted to write this high score value to an empty text file, with this I could then create a high score table. However, when a new game high score was determined, “scoretable” which is the name of the text file remained empty which means “.write” was not working. Like I mentioned earlier this will make the game more competitive as players can set goals and create competition amongst each other.



**Display score function**

*def* display\_score(*game\_active*,*game\_score*):

    if *game\_active* == "main\_game":

        score\_surface = game\_font.render(str(int(*game\_score*)),True,(255,255,255))

        score\_rect = score\_surface.get\_rect(*center* = (700,100))

        screen.blit(score\_surface,score\_rect)

    if *game\_active* == "game\_over":

        score\_surface = game\_font.render(*f*'Score: {int(*game\_score*)}' ,True,(255,255,255))

        score\_rect = score\_surface.get\_rect(*center* = (700,100))

        screen.blit(score\_surface,score\_rect)

        highscore\_surface = game\_font.render(*f*'High Score: {int(gamehigh\_score)}',True,(255,255,255))

        highscore\_rect = highscore\_surface.get\_rect(*center* = (700,400))

        screen.blit(highscore\_surface,highscore\_rect)

\*In this function I create the game state “game\_active”, with this I can have multiple game states which are just the current screens the player is on (alive or dead). When the game state is “main\_game”, the “game\_score” is “blit” onto the screen. When the game state is “game\_over” that is when the player is dead, the “game\_score” and “gamehigh\_score” are both displayed on the screen.

Text

Description automatically generatedWhen developing this function, I ran into the problem of trying to render the high score onto the screen.

This was caused by the program refusing to accept the game high score as an integer and instead requesting me to convert it to a string. This problem surprisingly fixed itself after I deleted the line of code and rewrote it. This will allow the user to keep count of their current score which is a representation of their progress as stated in my success criteria.

**Main menu function**

*def* main\_menu():

    global clicked

    while True:

        screen.blit(menu\_background,(0,0))

        write\_text(screen,300,100,'Main Menu')

        mx, my = pygame.mouse.get\_pos()

        startbutton = pygame.Rect(300,300,200,75)

        tutorialbutton = pygame.Rect(300,400,200,75)#Tutorial

        if startbutton.collidepoint((mx, my)):

            if clicked:

                game\_loop()#(should be the game function being called here)

        if tutorialbutton.collidepoint((mx, my)):

            if clicked:

                tutorial\_screen() #(should be the tutorial function being called here)

        pygame.draw.rect(screen, (255,217,102), startbutton)

        pygame.draw.rect(screen, (255,217,102), tutorialbutton)

        clicked= False

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == MOUSEBUTTONDOWN and startbutton.collidepoint((mx, my)):

                game\_loop()

            if event.type == MOUSEBUTTONDOWN and tutorialbutton.collidepoint((mx, my)):

                tutorial\_screen()

            if event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    clicked = True

        pygame.display.update()

        clock.tick(60)

\*The variable clicked is passed into this function as a global variable so it can be used across the function without the constant need for referencing. The menu background is put on the screen surface then I render the text “Main Menu” onto this surface using a “draw\_text” function I created.

I get the mouse pointer’s x(“mx”) and y(“my”) position which allows me to check for collisions with the “startbutton” I created using “startbutton.collidepoint”. When this happens the “game\_loop” function is executed consequently, these steps are repeated for the “tutorialbutton” I created. The two buttons are then drawn onto the screen with centre alignment using “pygame.draw.rect” and a beige colour to match the game design. I get an event’s occurrence with “pygame.event.get” which constantly checks if collision between two elements happen and if the mouse button is also down when this collision occurs, if so, depending on circumstances one of many functions may be ran e.g., “game\_loop”. “clicked” which was initially set to “False” changes to “True” to confirm the collision occurred.

A logic error occurred when checking for collisions because I didn’t make sure to check for both the “MOUSEBUTTONDOWN” and “menubutton.collidepoint(mx,my)” event happening simultaneously in my attempt to give the game mouse controls like I mentioned in my success criteria, this was because at first I didn’t realise that “.collidepoint” only checks for matching coordinates rather than clicks occurring. This meant that nothing would happen when I clicked on one of the buttons:

if event.type == MOUSEBUTTONDOWN:

game\_loop()

As stated in my success criteria the menu should allow the user to easily navigate through the game and I have made sure this was fully incorporated when developing my menu of the game, the buttons transfer the user to the various parts of the game.

**Tutorial function**

*def* tutorial\_screen():

    global clicked

    tutorial\_background = pygame.image.load('pyproj/gamefile/tutorialbg.png').convert\_alpha()

    while True:

        screen.blit(tutorial\_background,(0,0))

        mx, my = pygame.mouse.get\_pos()

        menubutton = pygame.Rect(720,50,150,75)

        if menubutton.collidepoint((mx, my)):

            if clicked:

                main\_menu()

        clicked = False

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == MOUSEBUTTONDOWN and menubutton.collidepoint((mx, my)):

                main\_menu()

            if event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    clicked = True

        pygame.draw.rect(screen, (255,217,102), menubutton)

        pygame.display.update()

        clock.tick(60)

\*A key part to any successful game is a tutorial as listed in my success criteria. Even though I didn’t list the tutorial as a key success criterion, I still created the tutorial because I thought it was essential to the playability of the game and eliminating any skill gaps that may exist the game is targeted at a spectrum of audiences therefore a tutorial will be important to understanding the game.

The image that is used as the tutorial screen is imported because it was simpler to create the tutorial in word then import it as an image and fill the screen with it instead of using pygame modules to draw a screen. I used word’s draw and shapes feature to make this screen.

Once this is done, I get the mouse pointer’s x(“mx”) and y(“my”) position so to check for collisions with the menu button I created using “menubutton.collidepoint”. When this happens the “main\_menu” function is executed consequently. I get an event’s occurrence with “pygame.event.get” which constantly checks if the collision between the two elements happens and when the mouse button is down while the “tutorial\_screen” function is running, “clicked” which was initially set to “False” changes to “True” to confirm the collision occurred. The menu button is drawn onto the screen using “pygame.draw.rect” and the screen is constantly updated-(“pygame.display.update”) at 60 fps(“clock.tick(60)”) to keep the window up until the function ends.

**Write text function**

*def* write\_text(*screen*,*x*,*y*,*Word*):

    import pygame

    pygame.init()

    font = pygame.font.SysFont('Constantia',40)

    text = font.render(*Word*, True,(255,255,255))

    #text.set\_alpha(100) sets the opacity of text

*screen*.blit(text,(*x*,*y*))

\*I made this function to allow me to easily write text onto the surfaces of my game, the surface-(“screen”) which the text will be put on is passed as a parameter along with the “x” and “y” position of the text and what the “Word” will be. The pygame library is imported to be used then initialised. The font the function uses is set to “Constantia” and the variable “text” renders the text with the colour assigned. The text is then put onto the screen surface with “blit”.

# D. Evaluation

=Errors/testing = Links to success criteria

My initial systems diagram was inaccurate because it was broad, I resolved this through the flowcharts I created, the function of each section was better defined compared to the systems diagram. They were more accurate because I was able to tailor each flowchart to a success criteria objective where relevant, this meant that I could then incorporate the features of each flowchart into my algorithm. I did have to make some adaptations to my code however, because of the limitations I encountered, such as time, which was my biggest restrictor of game development. I developed my flowcharts based on the success criteria for each section of my program, such as ensuring there is a high score function within my program as stated within my success criteria by first constructing this function as a flow chart which I can then easily translate to my programming language of choice. In conclusion the flowcharts I created were especially useful in my development because they functioned as a layout of each section of the game, and they were also guidelines to ensuring I met my success criteria.

My lack of experience with graphic designing meant that I could not use Photoshop to design the graphics of my game, such as the backgrounds and the game sprites. I instead managed to use the Word draw feature and its objects library to create my game backgrounds and the initial pillar obstacle used in the early stage of my game development. Through repeated testing of my game, I realised that the sprites created on word became unrefined when imported onto my code editor, the resolution of the sprites were reduced automatically which I think may have been to accommodate for faster performance when running my game. Where possible I instead opted to used images from the internet as sprites, for example my game character and the fence obstacle. I also discovered that the frame rate I had set my game to sometimes caused issues during runtime, it led to my game freezing, lagging, and sometimes crashing. The framerate issue was caused by the fps I had set being too unnecessarily high for my processor to manage and the order of the iterative loops I had used in my game. Through a couple tweaks to the order of the loops within my program and a reduction of the frames per second I was able to get the game to a comfortable state with rarely any visual issues being present when running.

**Interview with user evaluation:**

* How familiar were the keyboard controls and how quickly were you able to adapt to them? I was able to quickly adapt to the game controls because the game made use of the vanilla pc controls which w, a ,s d and the space bar. This meant that there was not a skill curb when it came to learning how to play the game and I very quickly became comfortable with the gameplay, there was also alternatives provided for each key function, so I was able to choose between the ones I was more familiar with.
* How interactive was the game background? Although I would have preferred the background to alternate, the motion created from the background and the on-screen objects moving with along with the character created an immersive feel, forming a storyline around the game. It also functioned as a progress marker because I used the looping background as a determinant of how far I travelled.
* Did the game’s use of a score and high score make it more competitive? I became very fixated on the score counter when playing the game because I was determined to beat my previous score on each iteration and beat my high score or set a new high score. Therefore, it became incredibly competitive for me because I could track my progress and set new goals.
* How useful was the tutorial in helping to learn the game? The game was already easy to use because I was already familiar with prior renditions of flappy bird. But if I were completely new to the game, I believe the tutorial would have helped to understand how the game is played, a live action demonstration rather than a fixed image may have been better at explaining though because of the spectrum of audiences the game is targeted at.

By conducting the interview with the user, I was able to examine how well my success criteria objectives came across to an actual user. It also helped me determined the effectiveness of features that weren’t necessary, but I still chose to add such as the tutorial screen. From my evaluation, the user found the tutorial screen easy to understand and follow which means that it serves the purpose of adding it.

**Evaluation of Testing**

I recognised that the success of my testing is limited since I only conducted white box testing- me attempting to assess the limits of my game. If I managed to use black box testing my game would be more refined because a third party can test more limits that I as a sole developer may not be able to recognise, additionally black box testing removes the prejudice aspect of personal testing and provides a third person perspective on my code, further illuminating weakness within my program. Based on the testing I conducted; I have begun work on the bugs I encountered. Examples of these bugs include my game requiring the application to be restarted every time a player dies in order to reset the game loop because a bug means that the game loop is stuck when the player dies. I cannot show evidence of this because it is not a still event.

I managed to successfully connect the distinct parts of my game with different buttons that created ease of access for users of my program. The hub of navigation and game controls is my game menu which I idealised in my success criteria. The menu was an essential part of my program because it would allow a user to easily navigate through the game and it is the centre of all my game features.

**Limitations (solutions to limitations):**

* My biggest limiter when developing my game was the time limit, I was under, this meant that I had to make a couple sacrifices when deciding which features to include or remove from my game, I solved this by using computational thinking, by decomposing the initial problem into sub-tasks I was able to make use of abstraction in deciding which components were key to my game being a success. Due to this I was comfortable with developing my game under a time limit since I had sorted the sub-task into priority levels in my success criteria. However, even with this strategy I ended up short on time and had to exclude some features from my program to ensure its refinement to the highest standard.
* A second limitation I faced was my lack of skill in graphic designing, to create a game as a sole developer some knowledge must be present on graphic designing to address the visuals the game would have. To solve this, I used my already present Microsoft Word skills and managed to use its tool library to formulate different objects, backgrounds, and sprites all to be used within my game.
* The third limitation I faced during development was the insufficiency of my knowledge in the pygame modules. To increase my fluency in the library I made use of online forums such as stack overflow to ensure I could meet my success criteria and most importantly develop my program using the language I am most comfortable with.

## Success criteria Evaluation

Overall, my success criteria were functional because I was able to use it to the best of my ability accounting for the limitations that were out of my control. Based on this, the success criteria I created allowed me to refine my ideas and then polish the structure of the project. As a result of my success criteria, I recognised the features that were essential to my project and halt on the development of those that were not essential. This allowed me to account for the limitations I have and collaborate with them, reducing the impact they would have initially had. The success criteria allowed me to incorporate the computational thinking methods of abstraction and decomposition into my project enabling me to reinforce my knowledge in these.

***Linked Success Criteria and Program Evaluation***

Having completed the program, upon running tests my program was able to meet the success criteria I listed to a prominent level, especially accounting for the limitations I was faced with.

**SC1:**

The main menu is fully functional with dedicated buttons that allow swift navigation through to each section of my game as required in my success criteria.

**SC2:**

Text

Description automatically generatedWhen on the game menu the system constantly checks for mouse inputs:

This meets the success criteria of mouse controls because mouse clicks are the most used control in my program, as they are responsible for all button interactions.

**SC3:**

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceI was able to meet my success criteria of objects creations thanks to the randomised obstacle generation I used, the program is able to generate “fences” with randomised heights:

**SC4:**

As seen in the screenshot above I was also able to meet my success criteria of the implementation of a score counter as the games score counter increments every 120 frames after checking for collision or if the user is still alive.

**SC5:**

The criteria of the creation of borders are met through the rectangles I placed around each on screen object.

These “rects” are used in python to create a boundary around each object which is a literal rectangle and when rectangles interact by having shared coordinates (x,y), collisions can be detected.

**SC6:**

I was also able to meet my success criteria of a changing background as my main menu and the game screen have diverse backgrounds. The idea around my main menu background is the stillness, lack of activity allows the grass to grow, it then transitions to a clear field when the game loop is running because constant activity such as walking on the fields prevents the grass from growing by killing it. This meets my success criteria aim of creating a storyline around my backgrounds.

**SC7:**

I implemented two keyboard controls into my game, which is the w and space button. They serve as alternative for each other, and their purpose is to control the game sprite. However, I wasn’t able to implement multiple keys such as ‘m’ for muting the game because I didn’t add audio files to my program because I unfortunately ran out of time, so this success criteria wasn’t met to my fullest ability.

**SC8:**

My last key success criteria, which is object generation was also met due to the fact that I was able to set the objects to generate at set interval.

**UNMET/MET Low Priority SC Eval:**

During the creation of my success criteria, I factored in the time limitation I was presented with and organised my success criterium on importance and how likely I could complete the task during the time limit I had. As a result, I was able to ensure that I was on task with the key features my program required.

* **L(SC9):** Success criteria 9 is the creation of a tutorial screen, I managed this by making a screen on windows word using the tools, I then imported this into my program. The game is simple to use to use so crafting the background was easy hence why I decided to add this feature to my program.

. Diagram

Description automatically generated

* **L(SC10/11):** This is the creation of collectible coins and a game shop; I chose not to begin the making of this feature because I knew I would be unable to complete it within the time frame. However, this is a feature I plan on adding within future updates of my program because it will keep users interested and make use of data collected from the game by collecting and storing the number of coins each player has and the items, they have purchased from the game store.
* **L(SC12):** The rationale behind the removal of a game setting is because of the lack of

customisable setting such as game sounds, a result of my time limitation meaning that I couldn’t create or acquire the resources needed to produce these features within but they can easily be made in the post development stage.

**Future Developments:**

In the future I would like to incorporate new features into my game such as the addition of new playable characters, more backgrounds and/or different maps. I would also focus on the development of features I was unable to create due to the time limit I was under, features such as a game shop and collectable coins which would serve as the currency to be used within the game shop for purchases. On the contrary based on the prioritisation of success criteria, I can conclude that my game was a success because I was able to meet the priority tasks and develop the priority features that were essential to my game by abstraction.

I plan to develop levels to maintain player retention, by constantly putting out new levels and challenges, users will remain interested in the game. This can be done through the incorporation of my success criteria objective of a game shop and collectible coins as it provides more purpose for the collectibles and further improves user retention and long-term user engagement.

The leader board table will also be created because I will need to collect a substantial amount of data from different users in order to compile a comprehensive database of different user’s scores prior to making the leader board. This will allow the leader board to be as competitive as possible because a larger sample size is usually more accurate at determining a range of values such as a high score.

**Bug Fixes:**

There were parts of my game that lacked refinement such as the minor bugs that can easily be fixed with more time spent on development.

An example of this was the problem with adding text on buttons.

The issue was that there is a layering system on pygame with text and the button function which meant that buttons created using the button function always had a higher priority that any other objects on screen which included text. I attempted to bypass this by using my write text function to display text on the screen and by repositioning the text, I also tried to reorder the functions by writing the text I wanted to put on the button before the button function is defined but all attempts failed. This resulted in my buttons not having text on top to indicate their functions. I carried out online research on the pygame button function and couldn’t find a solution because he more commonly used method of developing functions doesn’t involve using the button function rather using rectangles and checking for collisions; I plan on fixing this by possibly converting my buttons to the more commonly used method of rectangles which will allow me to overlay text.

A second bug that required fixing game loop being stuck after a player dies, this meant that upon death a player would be unable to start a new game without closing the program and reopening it, this was caused by the game loop not being reset every time a death is determined. In an attempt to fix this, I created the variable “active ” which determined the condition of which the game loop runs, the game loop would run if “active” is true and should theoretically reset if “active” is set to false, I “active” to false when the player dies but that still didn’t fix the bug, this bug will be addressed in future bug fixes.

**Maintenance**

I plan on frequent updates to my game through bug fixtures, addition of graphic effects, visual improvements, and the fixture of bypasses to the games system that may exist. These problems may lead to the players finding cheats that would allow them to bypass limits or find other ways of solving the game which would make the game less fun because it removes the difficulty of the game. Improvement to the game mechanics will also be included with latest updates in order to improve the fluency of the game.

User validation will also be introduced, upon starting the program the user will be asked to enter a username and password, this helps with the program security by removing unauthorized access to user data and to the game itself. Additionally, this helps with the post development data because at this stage I can determine the user and store their data such as the highest score they achieve on the leaderboard database.

## Input Validation

|  |  |  |
| --- | --- | --- |
| **Key Input** | **Outcome** | **Solution** |
| Esc | Has no effect on game when it is running | No Solution Needed as it is not a recognised input. |
| Right Click within game loop | Text  Description automatically generated | Since the problem is that I referenced the variable menubutton before it was defined, I redefined at this location. |
| All keys | No unexpected outcome from unassigned keys | No solution needed ; unassigned keys aren’t recognised as inputs. |
| Mouse Click | Only valid mouse clicks are recognised. Mouse click only has an outcome when it is on a button. | No solution needed, mouse click function is as expected. |

These were the only keys that required validation because all other keys had no impact on the program when inputted without referencing. The Esc key is tested because it often overrides referencing.

# Project AppendiCes

Complete code:

import pygame,sys,random

from time import \*

from pygame.constants import MOUSEBUTTONDOWN

game\_score = 0

gamehigh\_score = 9

delay\_counter = 0

clicked = False

pygame.init()

screen= pygame.display.set\_mode((900,600))

clock = pygame.time.Clock()

game\_font = pygame.font.SysFont('Constantia',40)

#Game surfaces

main\_background = pygame.image.load('pyproj/gamefile/mainbg.png').convert\_alpha()

background = pygame.transform.scale2x(main\_background)

menu\_background = pygame.image.load('pyproj/gamefile/secondbg.png').convert\_alpha()

mainmenubg = pygame.transform.scale2x(menu\_background)

game\_floor = pygame.image.load('pyproj/gamefile/soil.png').convert\_alpha()

game\_soil = pygame.transform.scale(game\_floor,(600,20))

game\_soil = pygame.transform.scale2x(game\_soil)

soil\_xpos = 0

game\_bird= pygame.image.load('pyproj/gamefile/flyingbird.png').convert\_alpha()

game\_bird = pygame.transform.scale(game\_bird, (120,70))

bird\_rect = game\_bird.get\_rect(*center* = (100,300))

ccf\_pillar = pygame.image.load('pyproj/gamefile/grassyfence.png').convert\_alpha()

ccf\_pillar = pygame.transform.scale(ccf\_pillar, (70,350))

pillar\_list = []

PILLARSPAWNER = pygame.USEREVENT

pygame.time.set\_timer(PILLARSPAWNER,1000)

pillarheights = [250,350,450]

*def* build\_floor():

    screen.blit(game\_soil,(soil\_xpos,560))

    screen.blit(game\_soil,(soil\_xpos+900,560))

*def* create\_pipe():

    pillar\_height = random.choice(pillarheights)

    base\_pipe = ccf\_pillar.get\_rect(*midtop*=(1000,pillar\_height))

    apex\_pipe = ccf\_pillar.get\_rect(*midbottom*=(1000,pillar\_height-200))

    return base\_pipe, apex\_pipe

*def* move\_pipe(*pipes*):

    for pipe in *pipes*:

        pipe.centerx -= 8

    return *pipes*

*def* draw\_pipes(*pipes*):

    for pipe in *pipes*:

        if pipe.bottom >= 600:

            screen.blit(ccf\_pillar,pipe)

        else:

            rotatepipe = pygame.transform.flip(ccf\_pillar,False,True)

            screen.blit(rotatepipe,pipe)

*def* check\_collisions(*pipes*):

    for pipe in *pipes*:

        if bird\_rect.colliderect(pipe):

            return False

    if bird\_rect.top<= -100 or bird\_rect.bottom>= 560:

        return False

    return True

*def* write\_text(*screen*,*x*,*y*,*Word*):

    import pygame

    pygame.init()

    font = pygame.font.SysFont('Constantia',40)

    text = font.render(*Word*, True,(255,255,255))

    #text.set\_alpha(100) sets the opacity of text

*screen*.blit(text,(*x*,*y*))

*def* tutorial\_screen():

    global clicked

    tutorial\_background = pygame.image.load('pyproj/gamefile/tutorialbg.png').convert\_alpha()

    while True:

        screen.blit(tutorial\_background,(0,0))

        mx, my = pygame.mouse.get\_pos()

        menubutton = pygame.Rect(720,50,150,75)

        if menubutton.collidepoint((mx, my)):

            if clicked:

                main\_menu()

        clicked = False

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == MOUSEBUTTONDOWN and menubutton.collidepoint((mx, my)):

                main\_menu()

            if event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    clicked = True

        pygame.draw.rect(screen, (255,217,102), menubutton)

        pygame.display.update() #keeps the window on the screen infinately

        clock.tick(60)

*def* main\_menu():

    global clicked

    while True:

        screen.blit(menu\_background,(0,0))

        write\_text(screen,400,200,'Main Menu')

        mx, my = pygame.mouse.get\_pos()

        startbutton = pygame.Rect(400,300,200,75)

        #start button

        tutorialbutton = pygame.Rect(400,400,200,75)#Tutorial

        if startbutton.collidepoint((mx, my)):

            if clicked:

                game\_loop()#(should be the game function being called here)

        if tutorialbutton.collidepoint((mx, my)):

            if clicked:

                tutorial\_screen() #(should be the tutorial function being called here)

        pygame.draw.rect(screen, (255,217,102), startbutton)

        pygame.draw.rect(screen, (255,217,102), tutorialbutton)

        clicked= False

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == MOUSEBUTTONDOWN and startbutton.collidepoint((mx, my)):

                game\_loop()

            if event.type == MOUSEBUTTONDOWN and tutorialbutton.collidepoint((mx, my)):

                tutorial\_screen()

            if event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    clicked = True

        pygame.display.update() #keeps the window on the screen infinately

        clock.tick(60)

*def* high\_score(*game\_score*,*gamehigh\_score*):

    if *game\_score*> *gamehigh\_score*:

*gamehigh\_score* = *game\_score*

        scoretable = open("highscore\_file.txt", 'w')

        scoretable.write(""+str(*gamehigh\_score*))

        scoretable.close()

    return *gamehigh\_score*

*def* display\_score(*game\_active*,*game\_score*):

    if *game\_active* == "main\_game":

        score\_surface = game\_font.render(str(int(*game\_score*)),True,(255,255,255))

        score\_rect = score\_surface.get\_rect(*center* = (700,100))

        screen.blit(score\_surface,score\_rect)

    if *game\_active* == "game\_over":

        score\_surface = game\_font.render(*f*'Score: {int(*game\_score*)}' ,True,(255,255,255))

        score\_rect = score\_surface.get\_rect(*center* = (700,100))

        screen.blit(score\_surface,score\_rect)

        highscore\_surface = game\_font.render(*f*'High Score: {int(gamehigh\_score)}',True,(255,255,255))

        highscore\_rect = highscore\_surface.get\_rect(*center* = (700,400))

        screen.blit(highscore\_surface,highscore\_rect)

write\_text(screen,320,320,"You died")

#Main game loop

*def* game\_loop():

    global soil\_xpos,pillar\_list,game\_score,gamehigh\_score,clicked

    delay\_counter = 0

    gravity = 0.5

    bird\_movement = 0

    game\_running = True

    score\_timer = 0

    while True:

        clicked = False

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_w:

                    bird\_movement = 0

                    bird\_movement -= 8

                elif event.key == pygame.K\_SPACE:

                    bird\_movement =0

                    bird\_movement -= 8

            if event.type == PILLARSPAWNER:

                pillar\_list.extend(create\_pipe())

            if event.type == MOUSEBUTTONDOWN and menubutton.collidepoint((mx, my)):

                main\_menu()

            if event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    clicked = True

    #end of for loop

        screen.blit(main\_background,(0,0))

        if game\_running:

            bird\_movement += gravity

            bird\_rect.centery += bird\_movement

            screen.blit(game\_bird,bird\_rect)

            pillar\_list = move\_pipe(pillar\_list)

            draw\_pipes(pillar\_list)

            if delay\_counter>=70:

                if score\_timer == (60) and check\_collisions(pillar\_list)==True:

                    game\_score +=1

                    score\_timer = 0

                else:

                    score\_timer += 1

                    display\_score('main\_game',game\_score)

            else:

                delay\_counter+=1

            soil\_xpos-=1

        else:

            gamehigh\_score = high\_score(gamehigh\_score,game\_score)

            display\_score('game\_over',game\_score)

            mx, my = pygame.mouse.get\_pos()

            menubutton = pygame.Rect(300,400,200,75)

            if menubutton.collidepoint((mx, my)):

                if clicked:

                    main\_menu()

            pygame.draw.rect(screen, (255,217,102), menubutton)

        #soil\_xpos-=1

        build\_floor()

        if soil\_xpos <= -900:

            soil\_xpos =0

        game\_running= check\_collisions(pillar\_list)

        pygame.display.update() #keeps the window on the screen infinately

        clock.tick(60)

main\_menu()