PyGame ‘Flappy’ Doc

Contents

1. Project Aim

The aim of this project is to introduce myself to the basics of game development by producing a game that is an imitation of 'Flappy Bird' using the Python 'PyGame' library.

2. Project Objectives

- To further develop python programming skills
- To introduce game development programming basics
- To introduce game development practices (design, development, etc)
- To further improve software development process & project management skills
- To produce a working MVP of a game like Flappy Bird

3. Budget

Project budget costs will be measured in terms of scheduled effort across five days.

Task	Schedule Allocation
Study PyGame official documentation	Day 1 - 2
Practice PyGame basics, follow educational tutorials	Day 2
Implement Initial PyGame Setup	Day 3
Source game images/sprites, sound effects, & music	Day 3
Implement game	Day 4 & 5
Testing	Day 5

4. Resource Identification

Resource	Description
PC	The machine to be used during development of project.
VS Code	IDE selected for developing the source code of project.
Python	The Programming language to be implemented for development of project files.
PyGame (Python)	The library to be installed for development of the project within Python.
Git / GitHub	The version control software tool selected to maintain software files.
Virtual Environment (venv)	The virtual environment selected to isolate project files, avoiding unnecessary dependencies being produced.

5. Documentation Review & Initial Setup

5.1 PyGame Documentation Basics

PyGame is not a game engine, it is a set of Python libraries. PyGame is a library composed of Python constructs, which include several different modules. These modules provide abstract access to specific hardware on your system, and uniform methods to work that hardware. Some of the major PyGame modules; display, joystick, event, image, key, music.

Every game uses a Game Loop to control gameplay. In a simple, traditional game loop, it would process user input, update the game state, and generate graphic / audio outputs. Every cycle of the game loop is called a frame. The number of frames per second can be assigned, this is known as the frame rate. Games today make use of multithreaded programming and multithreaded game loops to take advantage of multiple cores.

The first thing the game loop does is process user input, and the PyGame *event* system provides a method of user input. All user input results in an event being generated. Outputs are drawn to the screen and the screen is updated (flipped).

5.2 Setup

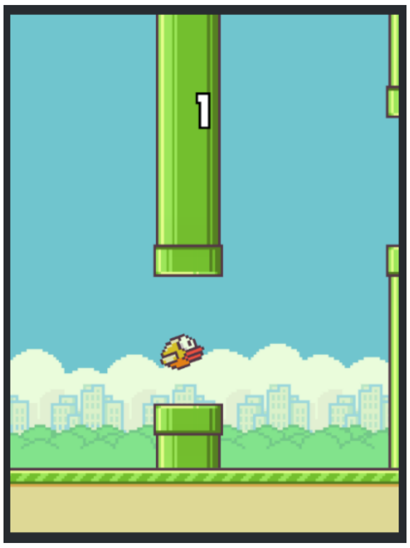
Created a GitHub account to upload a repository for the maintenance of an online backup and for version control of the project. Bash had been previously installed for the use of Git. Repository on GitHub is named 'PyGame'.

A virtual environment was created to keep project package dependencies and installations isolated. Venv module was used as the virtual environment tool. Venv directory created and named 'venv', as is convention. PyGame package installed to 'venv' folder using pip package manager.

The Python Interpreter set to use venv system path, to give access to downloaded PyGame module. An issue was encountered where interpreter was not able to access python packages correctly. Running 'pip list' command in prompt showed access to all python packages downloaded, so venv didn't appear to be correctly setup, or was assumed to have an incorrect pathway, or PATH had not been set in system environment variables. The venv folder PATH was added, yet the issue remained. Further research of the issue flagged a possible problem caused due to moving the initial project folder and 'venv' folder from original folders they were created during the organising of the project files and folders. The initial venv & 'venv' folder were deactivated and deleted, with process repeated to give clean install of the virtual environment. A pip list command was run and confirmed that new venv had been created with no packages installed. Installation of PyGame to the venv then showed that the venv interpreter and game.py file was now correctly locating and importing the PyGame module.

5.3 Game Design Goals

This is my first experience with game development. The game will attempt to replicate aspects of 'Flappy Bird'. I play tested the web version of Flappy Bird to get an understanding of the game's design and its mechanics. The produced game design goals are outlined below.


1 Flappy Bird game during play test

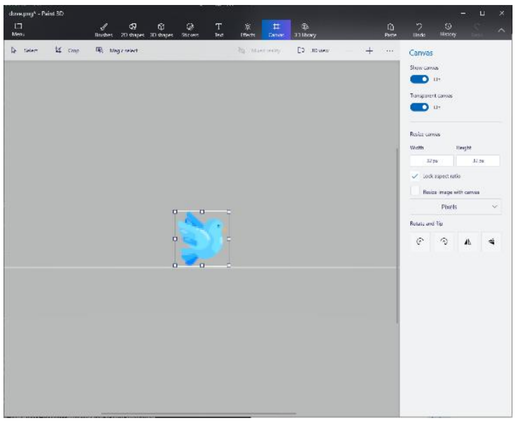
- Player input to move the flappy bird sprite
    - Movement based on clicks to make bird 'jump'
    - Initial implementation could have player character move directionally with arrow key inputs
        - Player must not move off screen
- Avoid incoming/moving objects
- Collision with objects must end game

The goals identified are suitable for to produce a minimum viable product. Other goals for possible further implementation if the scheduled time allows include:

- Increment score as the player successfully passes obstacle
- Track & display player score
- Provide user with score displayed upon game ending
- Offer replay option
- Multiple lives
- Start button / menu

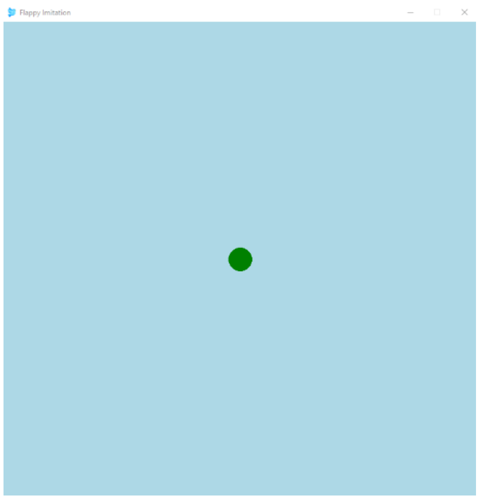
‘Dove’ Icon to be used as character image sourced from flaticon.com. Attribution given to ‘Freepik’ for free icon (https://www.flaticon.com/free-icon/dove_3241838#). 32-bit .png version of icon was used for project. The icon was edited in Paint3D software to suit needs. The creation of self-made Pixel and 2D art to be studied at later date, this project is for focusing on Python / PyGame programming and development aspects of game design/development.

Asset folder created to act as repo for storing graphic and audio files.


2 'Dove' icon by 'Freepik' being edited in Paint3D

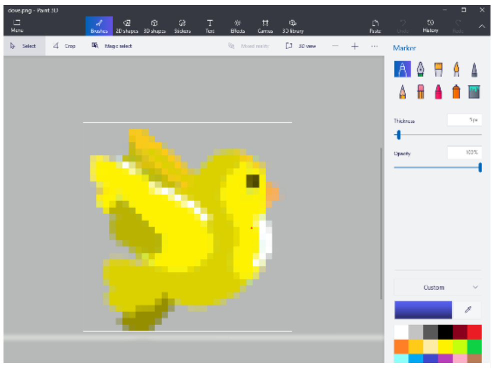
Initial implementation to show setup of game window, screen, icon, title, background colour, and circle to represent player starting position.

Selected blue coloured ‘Dove’ icon to be changed due to colour design conflicts with ‘sky blue’ background.



3 Initial Game Environment Setup

‘Dove’ icon was altered and recoloured using with Paint3D. Yellow colour selection chosen as it is bright colour easily seen for player to track on screen and being quite a complementary colour to blue when reviewing the colour wheel.


4 'Dove' icon after recolouring

With the virtual environment activated, the basic shape of the application established, the Git repository created on GitHub with an initial commit, and player sprite ready the setup of the project has been completed. The project can now continue into an initial functional prototype to begin the development the game.

A ‘pip freeze’ command was used to produce the installed packages and their versions which would act as requirements to run the project files, this produced just one result:

- pygame==2.1.2

6. Initial Functional Prototype

6.1 Initial Prototype Requirements

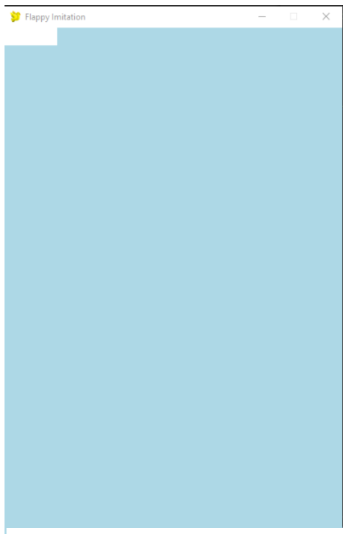
- Identifiable player character
- User input-based movement
- On-screen obstacles

6.2 Initial Prototype Game Loop

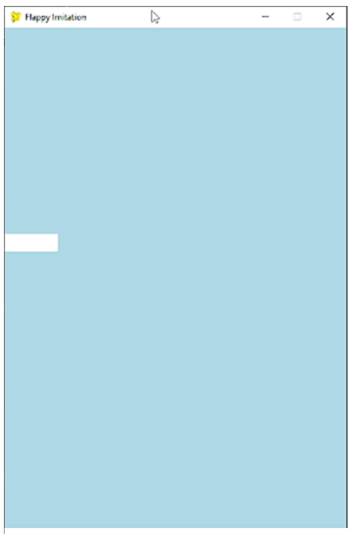
```python
# Game Loop
running = True      # Bool control variable to keep game loop running
while running:      # Starts the event handler


        # Loop through every active event
        for event in pygame.event.get():
            # Did user click the window close button? If so, stop loop
            if event.type == QUIT:
                running = False
            # Did user press a key?
            elif event.type == KEYDOWN:
                # Did user press Escape key? If so, stop the loop
                if event.key == K_ESCAPE:
                    running = False
```

The initial game loop created the game screen / window, set to 720 x 480 pixels. The game loop consisted of two events, which were both for quitting the game. Either by closing the window with the close button or by key pressing the escape button. Code was created to ensure if player sprite attempted to move beyond set screen dimensions they would be set back within the screen.

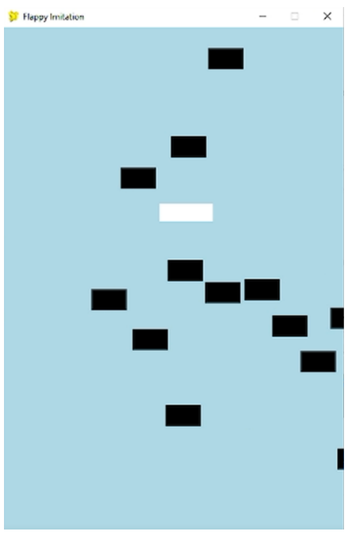
The game window was given a light blue colouring, RGB values 173, 216, 230, to replicate the sky in game. The game title 'Flappy Imitation' was assigned, seen along the top of the window, along with a yellow bird icon. The initial player object was created as a rect object filled white. Its default position was top left of the screen (coordinates 0,0). Player and Obstacle classes were created, with rects assigned to represent each. User input in the form of arrow key or WASD presses were implemented.



5. Player 'character' created



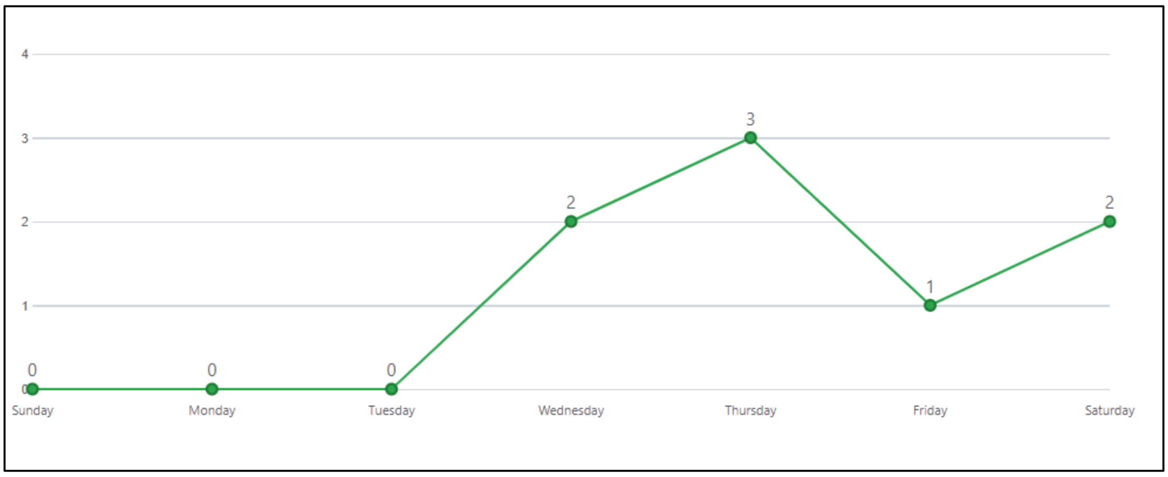
6. Player given starting position



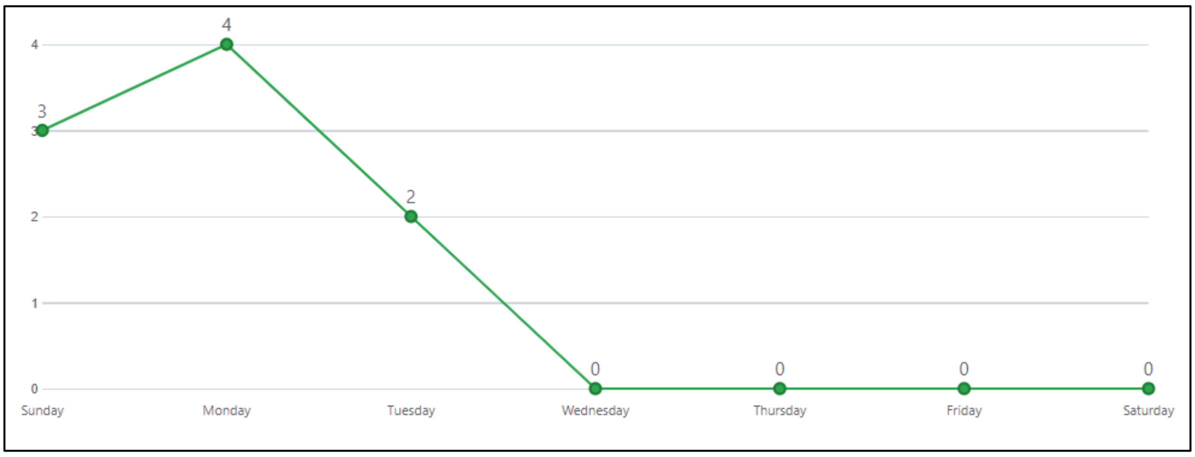
7. User input-based movement & 'Obstacles' added

7. Implementation

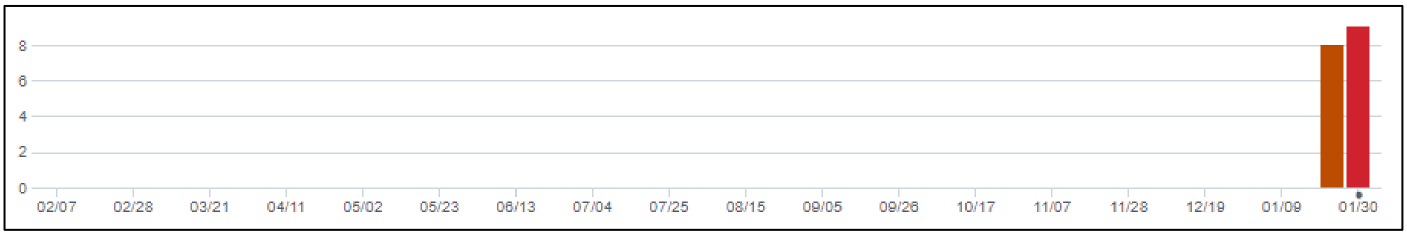
7.1 Version Control



Git commit hostry for week of Jan 23rd - 8 commits



Git commit hostry for week of Jan 30th - 9 commits



Git commit hostry by year

Git used within Bash terminal to create version control on GitHub.

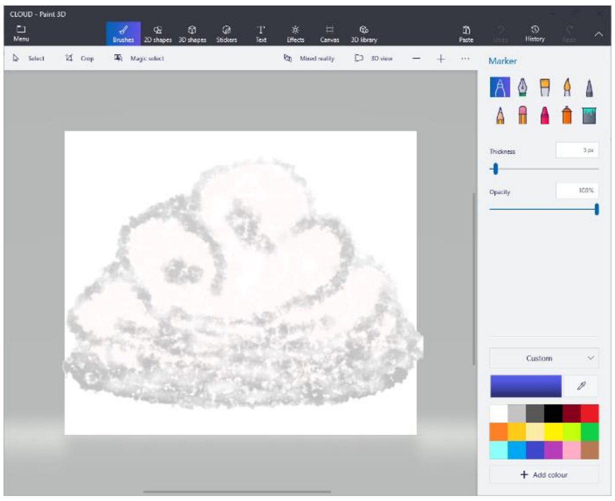
7.2 System Implementation

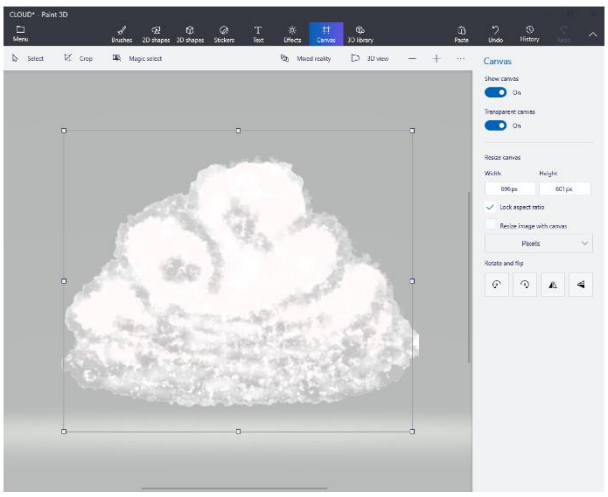
The player sprite was loaded in and implemented into the code base, replacing the white rectangle as the player 'character'.

When testing movement, the game speed seemed wrong, when assigning the 'speed' the differences between 1, 2, or 3 seemed inconsistent. Upon research of the movement speed issue, it was discovered that the actual 'game speed', referring to frame rate, was the issue. Upon setting up clock.tick(30) to set the game event loop to run at 30 frames per second, the movement speed of the game was immediately smoothed out. Testing and adjusting the movement speed of on-screen sprites & rects was resolved. An FPS display was created to observe the frame rate as I tested adjusting it between 30 FPS and 60 FPS, ultimately settling for 60 FPS.

The oncoming black rectangle were replaced by green rectangles which reach up and down the screen which appear in sequence. These obstacles are to replicate the pipes in the original Flappy bird. Understanding the positioning of the obstacles was a challenge. It took testing and changing rect object dimensions to get a better understanding and ultimately finding the appropriate coordinates. Having the rects reach towards the centre of the screen at random heights was also a challenge of understanding coordinates and settings. Testing through inputting different values was the approach which provided best insight and resolution.

The cloud images were made in Paint3D by me, it seemed a simple enough task to quickly draw one up rather than search for assets online. This was also a chance to get very basic idea of operating an art pipeline. When importing the cloud.png into the game, it was shown that the .png had not been set to transparent, and the pixels size of 32 was too small. With the assets brought back into Paint3D it was enlarged and the background was set to transparent before exporting. With the adjustments made it was reloaded into the game and the issues were resolved.



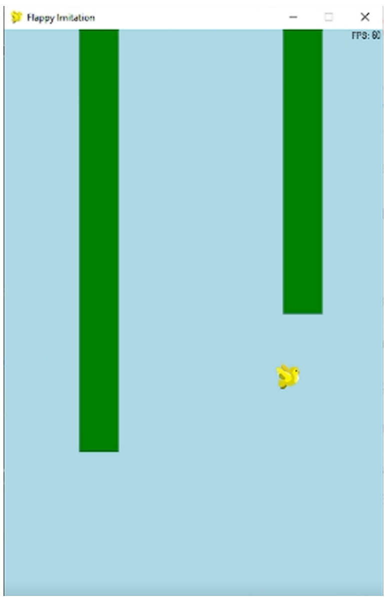


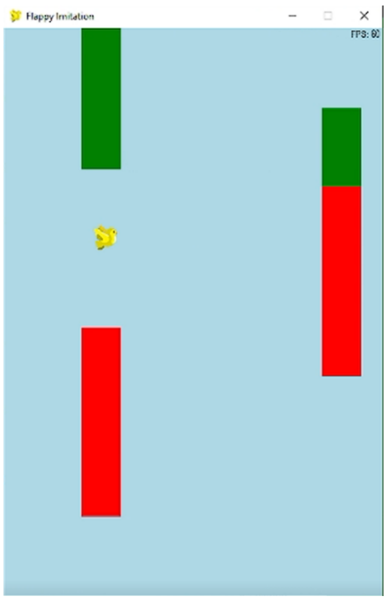
Cloud speed was refined and set to be slower than the player and moving obstacles, which made for a much better feel to how the clouds moved across the screen. The order in which the player, obstacles and clouds were drawn to the screen was also adjusted to place the Cloud sprites the be drawn in the background, with the obstacles and player sprites in the foreground.

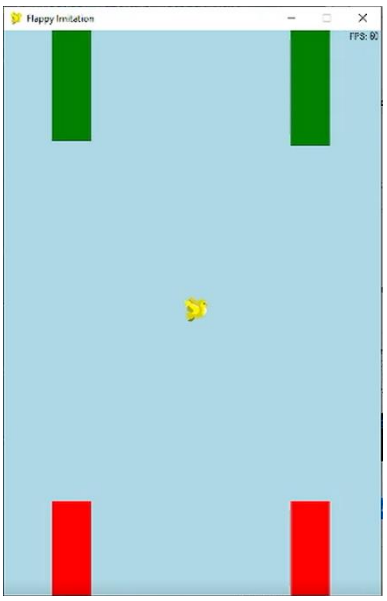
For further development I could implement occasional cloud sprites which vary in image and size. Having occasional clouds that can obscure the obstacles and player would provide more variety and add to the challenge of gameplay.

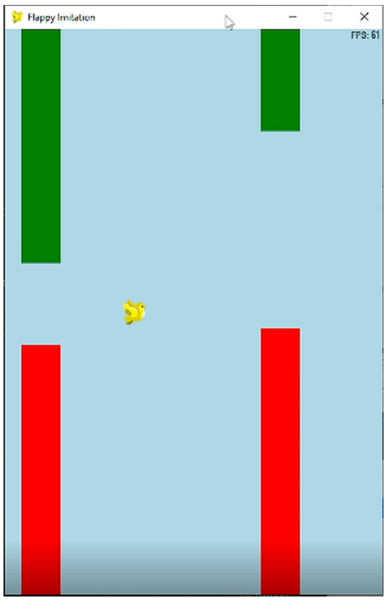
I was unable to figure out within my time limit how to correctly implement the sound effects intended to play as the player sprite moved upward or downward. A wing flapping sound effect for upward and a bird call for downward. They were to occur on key press, however due to the movement system, the player holds down the keys, which caused the sound effects to play continuously. The sounds were removed from the game as I was already 'overbudget' when it came to the scheduled effort in days.

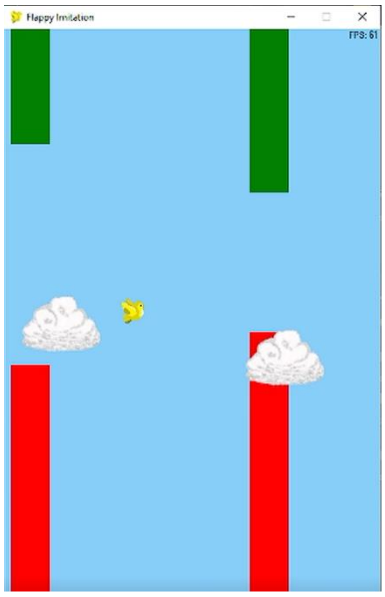
Development Story Board

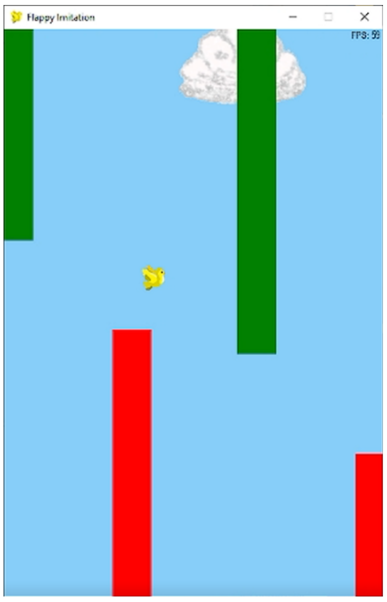


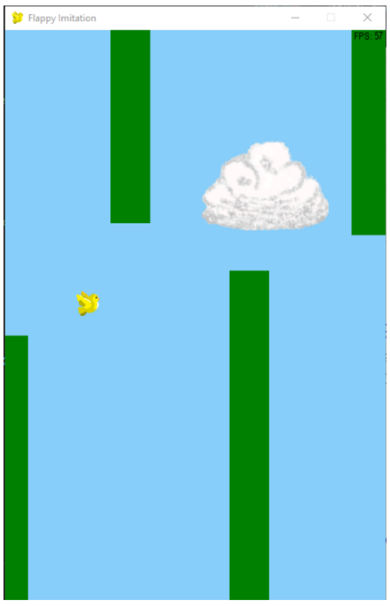












8. Verification

Test cases carried out:

Test ID	Test Case	Actual Result
1	When the game begins the game music is to play.	Pass
2	The music is to loop during play.	Pass
3	When player presses an arrow key or WASD key the associated directional movement is to be executed on screen.	Pass
4	When player sprite reaches edge of screen the sprite does not move off screen.	Pass
5	The associated sound effects are to play when the player sprite moves up or down.	Fail
6	When player sprite collides with obstacle sprite the 'game over' is to sound plays.	Pass
7	After the 'game over' sound plays the game ends.	Pass
8	The game closes when the escape key or window close buttons are pressed.	Pass

9. Conclusion & Reflection

9.1 Critical Appraisal of Project

Overall, I'm happy with the MVP achieved within the time dedicated to the project and I had fun. With zero experience of game development and the PyGame module I feel I learned a lot. I could return to the project in the future for important aspects such as user score and a start menu, but I feel the game challenge is easily understood and can be enjoyed. Ultimately, I must leave the project where it is as I need to move onto other material.

9.2 Reflection on the Budget, Development Approach, Implementation, & Execution of Project

I underestimated budget cost in terms time spent of reading documentation and learning concepts, implementing concepts, and doing so with genuine learning.

Task switching was an issue, where a problem would arise, and I would search for solutions which required implementation of other aspects of the game. Or becoming distracted with things I found interesting. Having clear, outlined tasks for allotted time slots could be one solution to this in the future.

A lack of software methodology applied to development while approaching the project was obvious. Being a small project, I didn't consider taking a methodology into account. In the future all projects no matter how small will be assigned some form of development methodology in attempt to keet delivery of a MVP within schedule.

Using a methodology that implements Trello to create task cards, such as Kanban, would be a way to define each task or aspect of the project needed to complete. Having the progress of tasks and overall project laid out visually should help with any blind spots.

Smaller batch deliveries would be implemented with a methodology, with documentation on changes more frequent and accurate. Git commits should be more frequent within these smaller development periods, helping with eliminating omissions in patch notes and better backup/rollback opportunities.

With these Trello cards created for each task sprints could be established made up of selected group of tasks. This should help reduce task switching and outlining the desired deliverables for each step in the development process.

Thanks for reading.