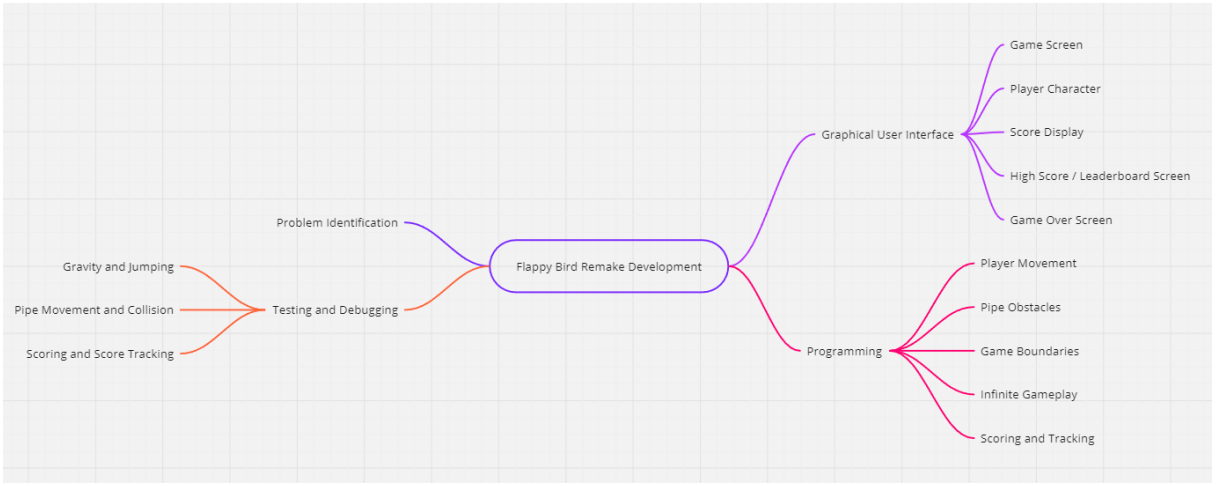
Criterion A – Problem Specification

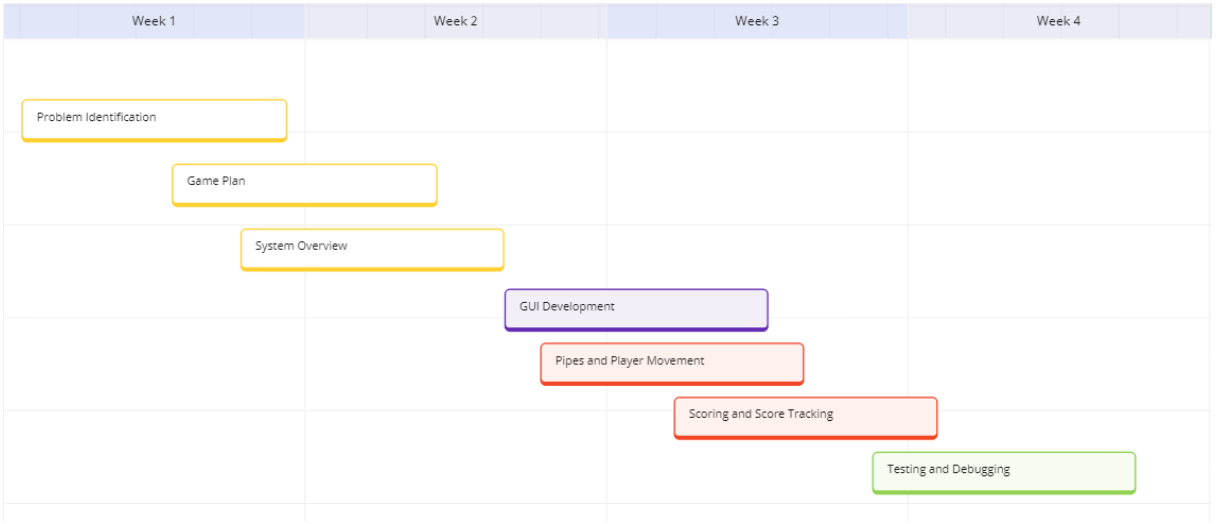
For my Computer Science Internal Assessment I decided that I want to replicate a game I used to play years ago called Flappy Bird. I decided to do this to test my programming skills and because I think that it would be a fun personal project that I am interested in. I decided to pick Python for the development of this game, because it features the PyGame library that will make it simple and quick for me to create game objects and GUIs, and because Python is a language that I was taught in school, therefore, it is a language that I am versatile in using.

In order for my game to succeed, it should meet some basic success criteria:

- The game will feature a player that is constantly pulled by gravity, but can jump mid-air to keep themselves in the game.
- The game will feature pipes with holes that the player must pass through.
- If the player hits a pipe, he loses the game.
- If the player goes above the game map or falls below it, he loses the game.
- The game goes infinitely until the player loses.
- For every pipe passed without hitting, the player earns a score.
- The game keeps track and saves past scores in a separate file.
- The game can display the high scores achieved, with their time of achievement.
- The pipes keep speeding up the longer the player plays.

Criterion B – Planning



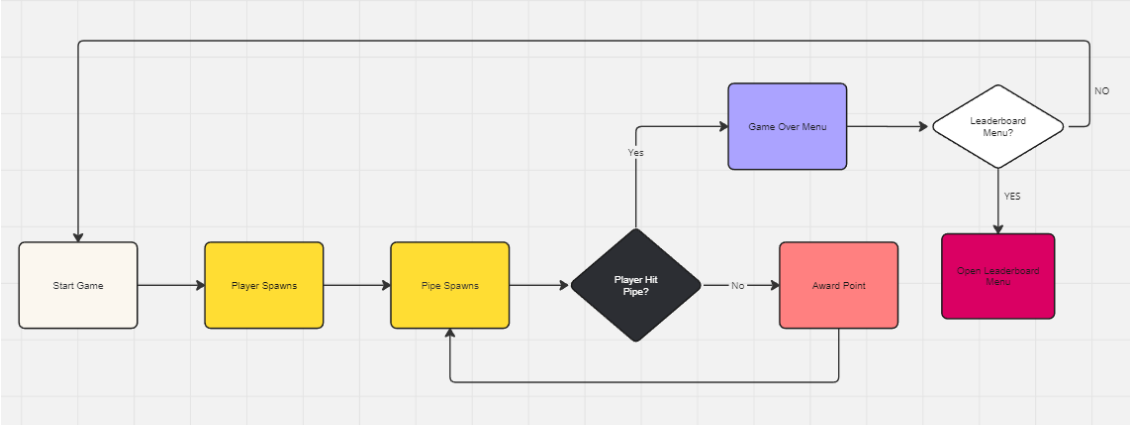


Criterion C – System Overview

To test my game once it is done, I will use the following testing strategy:

1. Open the game.
2. Pass through a few pipes without dying. (my score should increase and save)
3. Hit a pipe on purpose. (I should get a game over screen)
4. Restart the game.
5. Don't jump on purpose and fall off the map. (I should get a game over screen)
6. Check the leaderboard menu. (my two scores should be saved and ordered)
7. Close the game window and open it again.
8. Play the game until the speedup of pipes is visible.
9. Check the leaderboards. (all scores should be visible even though the game was closed)

The following game loop, made using the Miro online service, represents my system model:



Main Game Loop Algorithm

```
function main:
    initialize bird_rect with position and size
    initialize bird_velocity to 0
    initialize pipes as empty list
    initialize pipe_timer to 0
    initialize score to 0
    set running to true
    set game_over to false
    set pipe_speed to initial value
    initialize passed_pipes as empty set

    while running:
        for each event in event_queue:
            if event is QUIT:
                set running to false
            if event is KEYDOWN:
                if key is SPACE and game_over is false:
                    set bird_velocity to FLAP_STRENGTH
                if key is R and game_over is true:
                    call main
                    return

        if game_over is false:
            add GRAVITY to bird_velocity
            update bird_rect.y with bird_velocity

            increment pipe_timer
            if pipe_timer > threshold:
                reset pipe_timer
                generate random pipe_height
```

```
            add new pipes to pipes list

        for each pipe in pipes:
            move pipe with pipe_speed

        for each pipe in pipes:
            if bird_rect collides with pipe:
                set game_over to true
        if bird_rect.y is out of screen bounds:
            set game_over to true

        remove pipes out of screen from pipes list

        for each pipe in pipes:
            if bird passed pipe and pipe is not marked as passed:
                increment score by 0.5
                mark pipe as passed

        clear screen
        draw bird
        draw all pipes
        draw score
        update display

        limit frame rate

        increment pipe_speed gradually
    if game_over is true:
        save score
        show game over screen with score
        while true:
```

```
            for each event in event_queue:
                if event is QUIT:
                    quit game
                    return
                if event is MOUSEBUTTONDOWN:
                    if restart button is clicked:
                        call main
                        return
                    if leaderboard button is clicked:
                        show leaderboard
                        while true:
                            for each event in event_queue:
                                if event is QUIT:
                                    quit game
                                    return
                                if event is MOUSEBUTTONDOWN:
                                    if back button is clicked:
                                        show game over screen again
                                        break
                            if event_handled:
                                break
                        return
            update display
            limit frame rate
    quit game
```

Save Player Scores to External File Algorithm

function save_score(score):
  if LEADERBOARD_FILE exists:
  open LEADERBOARD_FILE in read mode

```
    load leaderboard from file
  else:
    initialize empty leaderboard
  get current time in specified format
  append {"score": score, "date": current_time} to leaderboard
  sort leaderboard by score in descending order and keep top 10
  open LEADERBOARD_FILE in write mode
  save leaderboard to file
```

Pipe Class with Movement Algorithm

```
class Pipe:
  constructor(x, y, passed=False):
    create rectangle with position (x, y) and size (PIPE_WIDTH, PIPE_HEIGHT)
    set passed to passed

  function draw(surface):
    draw rectangle on surface with color GREEN

  function move(speed):
    decrease rect.x by speed
```

Criterion D – Development

Technique 1: Object-Oriented Programming

*Code sample in Appendix, page 2 and page 3.

I used object-oriented programming to develop unique pipes for my game, in order to set up a Pipe class that can be used to create unique pipe objects with different coordinates and sizes, and so that I could put them in sets to check if they have been passed or not. Objects of the Pipe class are initialized if the game is ongoing (Appendix, page 6) and appended into the set. This makes the game create unique and fun pipes that can be passed. My testing strategy effectively tests this technique during the testing of the game itself and passing the pipes and scoring, as the pipes are objects of the Pipe class and contain data defined by the class, utilized during testing.

Technique 2: File reading and writing

*Code sample in Appendix, page 4.

In order to save my scores between restarts of the game instance, I write the amount of points to a LEADERBOARD_FILE file in the directory in which the game runs. I use the ‘with’ statemet to ensure that the resources used will be released when all processes end, and I load the file using the JSON library for Python. Then, I load the current DateTime and format it, and I append it as a JSON object to the leaderboard variable, and then sort it by the amount of points using a lambda statement, and after that, I dump the contents into the file. This effectively makes my game able to save points between game instances and makes sure that the players won’t lose their progress, so it is a simple yet useful technique, that is well-tested and evaluated by my testing strategy, since parts 7 and 9 of the testing strategy are used to test this technique specifically.

Technique 3: Graphical User Interface (GUI)

*Code samples on nearly all pages, but mostly on pages 2 and 3.

A GUI allows my game to be playable and makes the users able to manipulate the player character. To create a GUI, I used the PyGame library in collaboration with different classes and objects. For example, to create buttons, I created a Button class, that contains necessary fields such as text, coordinates on the screen, size, color, etc. Assigning values to these fields by creating objects and passing them to PyGame makes me able to create as many buttons as I need without the need for repeating code. The testing strategy tests this technique effectively throughout the entire testing process.

Criterion E – Evaluation

Since my game is able to create movable pipes with random sizes and changeable speeds, a controllable player that can play the game, a score counter that is increased upon passing pipes and that is saved into a high score leaderboard that is saved into a separate file between game restarts, and the game can be played infinitely, I consider all of my success criteria met and the game complete under the criteria of this IA specifically.

However, I believe that this game can be further improved in lots of different ways. One example would be to make the GUI more appealing by creating icons for players, pipes and backgrounds, and changing the menus currently visible. Another way that the game could be heavily improved would be to add a shop where the players can buy items using their obtained points, so that the players would have something to look forward to other than high scores, this can include buy player skins and icons, powerups, etc.