**Introduction**

These papers detail the process of planning, designing, developing and testing a fully functional computer game.

Throughout this report I will be discussing each separate section of the base of the game, with different algorithms.

**Overview of the game**

The game I programmed is a 2D automated side scroller clone of the game Flappy bird. The objective of the game is to get past as many pipes as possible without crashing into one of them.

Due to the game being endless there is no win condition the only way to “win” would be to get the highest score as you possibly can. The lose condition would be to take a hit from the obstacles which block the path and top and bottom of the screen.

**Mechanics of the game**

**Scoring**

The counter increases as the bird passes through pipes of varying heights.

If the pipe obstacle or the border of the game window is hit, the game session ends with the final score displayed in a dialog window.

**Controls**

The controls of the game are simple, press the space bar and upwards facing arrow key to increase the height of the bird by a certain number of pixels. Leaving the input will cause the bird to drop.

**Programming language and modules used**

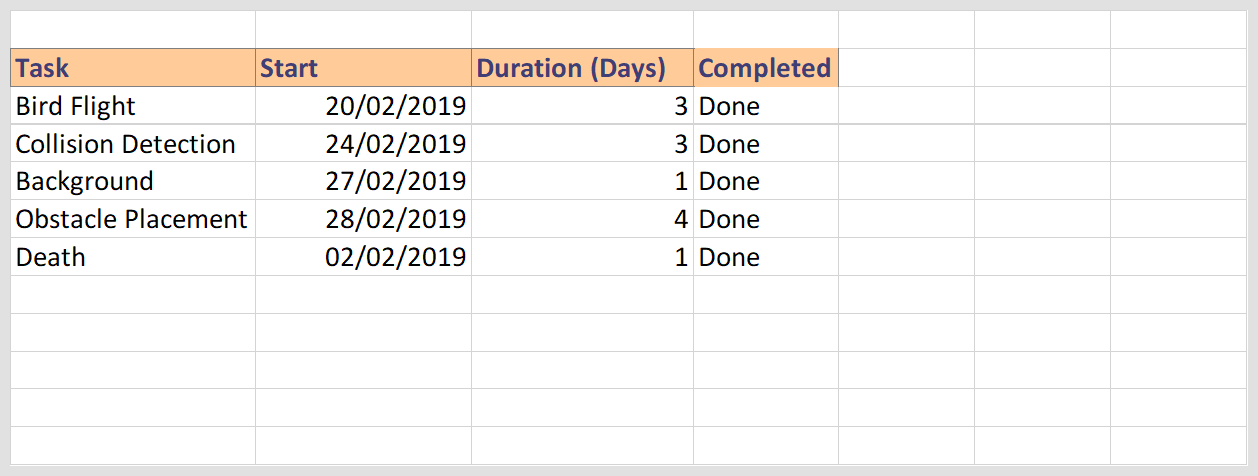
To develop my game, I used the programming language python alongside the gaming module pygame.

I picked this language due to me already understanding the language and how relatively easy it is to use. However, with python alone developing the game would have been a difficult task so alongside python I used the module pygame.

Pygame is a module/software which allows the development of games in Python to be more efficient and easier due to python not being a popular language to use for game development.

**Project Plan**

I separated the development of the game into 5 tasks. I prioritize each task and give the more difficult tasks a longer duration and the easier tasks a shorter duration.



**Test Plan**

Here are the number of tests I undertook to help me identify possible errors within my code and to make sure the game works to a satisfactory level.

|  |  |  |
| --- | --- | --- |
| **Possible Tests** | **Expected Outcome** | **Actual Outcome** |
| Pygame been imported? | When I run the program in the dialog window  It should show  Pygame (the version which has been used)  Hello from the pygame community. | In the output window it displays Pygame 1.9.4  Hello from the pygame community |
| Are the window dimensions a sensible size? | I decided that for the window size I should it make it the same dimensions as the background. This will make the game look more presentable and when programming the obstacle placement, it will make it easier.  When I run the program, I expect to see the window size align with the background image of the game. | The background fits perfectly with the window size.  So, no changes are to be made. |
| Images loaded in properly. | The images should load without any background meaning that the image has an alpha channel.  When loaded into the game it shouldn’t have any background part of the image. | The images loaded properly without any extra colours interfering with background. |
| Controls mapped to the correct buttons. | After the space-bar and up-arrow are pressed the bird should move up the y-axis. | When the buttons were pressed the bird moved up the y-axis. |
| Collision detection working properly | When the bird collides with a pipe, the game should crash and in the output window your high score should be displayed. | Game crashes and the high score is displayed in the output window |
| Score counter working properly | After a pair of pipes are passed by the bird, the score counter should increase by 1 and in the input window the high score should be displayed.  The same logic applies to ring counter within the game. | Score counter increases by 1 each time. |
| Music has been imported and is playing during the game | Music should start when the game runs and should stop when the player loses. | Music does not play but action has been taken to fix the problem |

|  |  |  |
| --- | --- | --- |
| Test | Test Description and Outcome | Action Taken |
| Has Pygame been imported? | I import pygame at the beginning of the python file and  in the output window the following text is displayed  "pygame 1.9.4"  "Hello from the pygame community" | None |
| Are the window dimensions a sensible size? | Vary the window size by adjusting the variables "WIN\_WIDTH" and "WIN\_HEIGHT"  I set the background to have a width of 568 pixels and height of 512 pixels  The window is displayed at that size when the game is ran. | None |
| Have the images been loaded in properly. | I let the game run and check if the images of the bird, pipe and background appear. | None |
| Are the controls mapped to the correct keys. | I mapped the controls to the space-bar and up-arrow key. While the game is running, I press both keys at separate times and the bird moves up the y-axis. | None |
| Collide with pipes to check if collision detection is working. | Let the bird collide with the first obstacle that appearance, When the collision occurs the game crashes and in the output window, the score of the run is shown. | None |
| Check if score counter increases by 1 each time a pipe is passed. | Play the game and after an obstacle is passed make sure the counter increases by 1. | None |
| Check if music file is corrupted during gameplay | Music file has been loaded into memory, but it causes the game to crash when the program is run. | I have attempted to change the size of the music file and the format which suits pygame but to my avail the error still occurred. |

**Performance**

The game performs at 60 fps which was one of the first constants I set when coding the game. Due to the game’s graphical simplicity, it only utilized roughly 10% of my laptop’s CPU and 8% of the GPU to run the game. Which should not be much for any modern-day computer. Therefore, I have not encountered any constraints with my hardware not being able to run the game.

The platform which I intend to distribute is any computer that runs windows 10. Due to this OS being the most popular in the world it would make the most sense for it to be released on this platform as an executable file.

Machine tested on:

Microsoft Surface Pro 4 Laptop

1. CPU - Intel® Core ™ i7-6650U CPU 2.20GHz
2. GPU - Intel Iris® Graphics 540
3. Memory - 8GB
4. OS - Windows 10

**Errors I encountered when programming**

**Error 1**

C: \ Users ta \ Local \ Programs \ Python \ Python37 \python. exe 
'bird wing down': 
load image ('bird wing down .png) 
Syntaxerror: invalid syntax 
Process finished with exit code 
"C: /OnGi thub 
line 122 

This syntax error appeared when I ran the code due to a comma being missing after the closing parenthesis. This was an error which required a simple check of missing syntax at the end of the line of code.

**Error 2**

The error here is that I commented out the line of code which initializes the pipe class. Commenting a line of code causes that line not to run. This caused the variable self.x to have no value. Therefore, the error showed was an 'indentation error'. To fix this I removed the comment on the initialization.

init 
(self, 
pipe end irng, 
pipe_body irng) 
The nea pipepair automatically be assigned an x attribute of K 113TH 
The number of pieces "hich make up the top pipe. 
bottom_pieceg 
The number of pieces "hich make up the bottom pipe. 
gelf_x = 

**Error 3**

Traceback (most 
main() 
recent 
call 
last): 
loact_imaqes ( ) 
Images - 
return 'background': load imago( , 
Filo 
pygame . image .load(img file name) 
pygame. error: Couldn't open background 
Process finished with exit code L 
line 
line 
line 
line 
427, 
281, 
159, 
151, 
in 
In maln 
in load images 
in load image 

The error displayed here shows that the code cannot find and open the background image. Since python is only able to search for files if it is within the folder which the program is stored. To fix the error I had to place the image into the same folder which the game’s code is stored on my c drive.

**Error 4**

FlappyBirdGame.py 
backgroundmusic 7. wav X 
The file is not displayed in the editor because it is too large (302.82MB). 

Due to the mp3 files being incompatible with pygame, I had to convert the music file into a wav file. Which is an uncompressed version of an mp3 file. The music I converted from an mp3 to a wav file turned out to be 302.82 MB in size. When I attempted to play the file during the game. The output window displayed the error above and below.

Machine generated alternative text:
20 
21 
background_music_file = os.path.join(' 
' backgroundmusicl . wav ' 
sounds ' 
background_music - 
pygame. mixer 
• load( ' backgroundmusicl . wav' ) 
music. 
Exception has occurred: pygame.error 
Couldn't open 'backgroundmusicl.wav' 
File "C: \OnGithub\BTEC_Computing\Computing-Project\F1appyBirdGame.py", line 21, 
background _ music = 
pygame. mixer_music . load( ' backgroundmusicl. wav ' ) 
in <module> 
22 
23 
24 
25 
26 
27 
# Appropriate music to be found and played. 
# PLAY BACKGROUND MUSIC 
background_music_file. load ( " backgroundmusicl . wav" ) 
background_music . mixer_music . set _ volume (0.5) 
background_music . mixer_music . play( -1) 

To attempt to fix this I downloaded a drastically smaller version (47.3 MB) of the music file. However, I received the same error when I loaded the file into pygame. This leads me to believe that there is a fault with pygame itself which is not allowing me to play music files or the way which I program the music to play is incorrect and the error is not being identified.

**Pseudo-Code Analysis**

**Game loop – Pseudo-code**

WHILE

# Set refresh rate for screen

Clock.tick(FPS)

# define a list of pipes in display

#Add a new pipe after a set periodic interval

IF paused = false

pp=PipePair(images['pipe-end'], images['pipe-body'])

pipes.append(pp)

ENDIF

# define a list of rings in display

#Add a new ring after a set periodic interval and at a random y-axis coordinate

IF paused = false

rr=Rings(images['ring'])

rings.append(rr)

ENDIF

# monitor keyboard events for when the quit or pause buttons have been pressed.

FOR event in pygame.event.get()

IF event = QUIT or ESCAPE

Done = true

ENDIF

IF event = PAUSE or P

Paused = not paused

ENDIF

If event = UP or SPACE\_BAR

Bird.millisecond\_to\_climb = CLIMB\_DURATION

ENDIF

IF paused = true

# jump over the drawing logic

continue

ENDIF

ENDFOR

# Check if the bird has collided with any of the pipes during the last game frame.

FOR pipe in pipes

# check if the bird has collided with it.

Collision = pipe.collides\_with(bird, pipe)

IF Collision = true

Pipe\_collision = true

Break

ENDIF

ENDIF

IF pipe\_collision = true OR 0 >= bird.y >= (WINDOW\_HEIGHT - Bird\_HEIGHT)

Done = true

ENDIF

FOR x in 0 to (WINDOW\_WIDGTH/2)

#refresh the background display calling the pygame blit function

Display\_surgace.blit('background.png')

ENDFOR

#Remove the oldest pipes if it is no longer visible

IF Pipes[0].visible == false

Pipes.popleft()

ENDIF

#Remove the oldest rings if it is no longer visible

IF Rings[0].visible == false

Rings.popleft()

ENDIF

#refresh pipes on screen

FOR p in Pipes

#for each ring in the list of pipesto display we reduce x-axis coordinate to move it across the screen.

#update is a method defined in PipePair class

p.update()

# redraw passing pygame image, it's current position

Display\_surface.blit(p.image, p.rect)

ENDFOR

#refresh rings on screen

FOR r in Rings

#for each ring in the list of rings to display we reduce x-axis coordinate to move it across the screen.

#update is a method defined in PipePair class

r.update()

# redraw passing pygame image, it's current position

Display\_surface.blit(r.image, r.rect)

ENDFOR

#update the bird position

Bird.update()

# update the score based on number of pipes navigated

FOR p in Pipes

IF p.x + Pipe.WIDTH < bird.x and not p.score\_counted = False

Score += 1

p.score\_counted = True

ENDIF

ENDFOR

# update on screen score display

score\_surface=score\_font.render(str(score), True, (255, 255, 255))

score\_x=WIN\_WIDTH/2 - score\_surface.get\_width()/2

display\_surface.blit(score\_surface, (score\_x, PipePair.PIECE\_HEIGHT))

# Calling pygame flip function update and draw the score on screen

pygame.display.flip()

UNTIL done == True

**Menu - Pseudo-code:**

# initialize Pygame here.

Pygame.Init()

WHILE

*# Check Message Pump using Pygame.Event class for if the Quit Key has been pressed, exit game.*

Keyevents = pygame.Events.get()

FOR event IN keyevent

IF event.Type == QUIT

Pygame.Quit()

ENDIF

ENDFOR

# Using pygame's fill function - change the background to white

Display\_surface.fill(white)

***# Write the text to menu screen***

large\_text=pygame.font.Font('freesansbold.ttf', 100)

text\_surf, text\_rect=text\_objects("Flappy Bird", large\_text)

# Show action 'Start' and 'Quit'buttons on the menu screen. Calling the Button function defined below.

button("GO", 100, 450, 80, 50, green, bright\_green, "play")

button("Quit", 400, 450, 80, 50, red, bright\_red, "quit")

# refresh screen

Pygame.display.update()

UNTIL show\_menu == false

**Bird Flight**

# **Set variables**

**# These are the variables I used to make identification of objects easier**

self.millisecond\_to\_climb = millisecond\_to\_climb

self. \_img\_wing\_up, self.\_img\_wing\_down = images

self. \_mask\_wing\_up = pygame.mask.from\_surface(self.\_img\_wing\_up)

self. \_mask\_wing\_down = pygame.mask.from\_surface(self.\_img\_wing\_down)

Pseudo-code

BEGIN

IF millisecond\_to\_climb > 0:

Fraction\_climb\_done = 1 - (millisecond\_to\_climb)/Bird.CLIMB\_Duration

# This determines the bird's position on the y-axis

y\_coordinate =

(Bird.CLIMB\_SPEED \* frames\_to\_millisecond(delta\_frames) \*

(1 - cos(fraction\_climb\_done \* pi)))

Millisecond\_to\_climb = frames\_to\_millisecond(delta\_frames)

ENDIF

END

Bird Flight works by using the cosine function to get a nice curved path which the bird will travel along. Within the first few frames of the climb the bird won’t travel high up the y-axis but over time the bird will reach its maximum height based on the length. The height the bird reaches depends on the number of milliseconds the input is held for.

**Pipe Placement**

Convert the image of the pipe into the format Pygame uses.

Split the pipe image into two separate pieces. - The trunk and the top.

To display the image of the pipe on the screen it must be 'blitted' (displayed) on to the surface of the canvas.

The x-coordinate is the only value needed to determine the position of the pipe on the screen since it has no y coordinate.

Since each of the pipes have varying heights after they move across the screen. I alter the height of the trunk using the 'randint' function and add it to the constant height of the top of the pipe then 'blit' to the screen.

* + Determine/set x-coordinate from where we will start drawing the pipe. Note the pipes will be scrolling from left to right through the game screen.
    - Window width - 1
  + Drawing the pipe column.
    - Given ***pipe\_body\_img*** variable pointing to a jpg file for a pipe column
    - Given a constant pipe image size = ***pipe\_size*** (pixels)
    - Given game window height, which is a variable dependent on the screen size window\_height (pixels)
    - Determine size of gap in a pipe column, ***pipe\_window*** (pixels)
      * Logic

(3\****bird\_height***)+(3\****pipe\_body\_height***)

* + Determine number of pipes to be stacked vertical across the window with a set gap (pipe\_window)
    - Logic
    - ***Window\_height*** - ***pipe\_window***/***pipe\_body\_height*** = ***num\_of\_pipes***
  + Determine number of pipes at top and bottom of the screen. To vary the ***pipe\_window*** placement on the y-axis. So the ***pipe\_window*** does not stay in the same place on the y-axis as the column scrolls across the screen.
    - Logic
    - Given the ***num\_of\_pipes*** 
      * ***Bottom\_pipes*** = Take a random number between 1 and the ***num\_of\_pipes***
      * ***Top\_pipes = num\_of\_pipes - bottom\_pipes***
  + Drawing
    - Logic
    - Using Pygame Lib. The surface class provides the necessary functionality for drawing images on the screen.

pipe\_image = pygame.Surface((PipePair.WIDTH, WIN\_HEIGHT), SRCALPHA) -> assigns an area to represent our Pipe image.

To draw the top part of the column

* + Iterate through Top\_pipes calling the 'blit' function

**Pseudo code:**

Dealing in a top left coordinate system

For I in (i, 0 to top\_pipes)

Pipe\_position = (index \* PipePair.Pipe\_body\_height)

Pipe\_image.blit(pipe\_body\_img, (0, pipe\_position)

* + After drawing the body of the top column, draw the base of the column

|  |  |
| --- | --- |
| top\_pipe\_end\_y = self.top\_height\_px |  |
| Pipe\_image.blit(pipe\_end\_img,(0,top\_pipe\_end\_y)) |  |

To draw the bottom part of the column

* + Iterate through bottom\_pipes calling the 'blit' function

**Pseudo code:**

For I in (i, 1 to bottom\_pipes+1)

Pipe\_positon = (index \* PipePair.pipe\_body\_height)

Pipe\_image.blit(pipe\_body\_img, (0, pipe\_position)

**Death**

The logic used to code death is the same as collision detection

When the bird touches the bottom, top and pipes on the screen the game crashes.

Initial Logic:

* Determine if the bird y-coordinate is at the top or bottom bounds of the game window.

* Vertical\_OutOfBounds = (0 >= bird.y or bird.y >= WIN\_HEIGHT - Bird.HEIGHT )
* Determine if there has been a pipe collision or the bird has gone out of bounds within the current frame.
  + if pipe\_collision or Vertical\_OutOfBounds:

Done (Death) = True

**Pseudo-code:**

BEGIN

Bird.y = bird y coordinate

# **If the bird collides with any pipe**

Pipe\_collision = any(pipe.collides\_with(bird, pipe)

IF (pipe\_collision) or (bird.y <= 0) or (window\_height - bird.height <= bird.y)

Done(Death) = True

ENDIF

END

**Collision Detection**

Initial Logic

* 'Cover' the image of the pipe with a masked rectangle.
* When the bird's mask overlaps with the pipes rectangle mask it counts as a collision.

Initial Logic:

* pipe\_collision = false

for p in pipes:

if(p.collides\_with(bird))

pipe\_collision = true

* PipePair.collides\_with function is defined in the PipePair Class. In the sprite class in the pygame module, we use the collide\_mask function to check if two sprites (pipe and bird) have collided.
* pygame.sprite.collides\_with(self, bird)

**Pseudo-code**

BEGIN

Pipe\_collision = false

For p in pipes():

IF pipe.collides\_with(bird)

Pipe\_collision = true

ENDIF

END

**Feedback received**

Adam Castles - To improve the game's quality you should add music to make it feel more interactive.

Jude Mason - Add incoming collectibles to make the game have more of a purpose instead of just dodging incoming pipes.

Kevin Ramai - To help players with controls you should write a instruction manual which the players can read before playing the game.

Mark Bachadel - To increase the game difficulty adjust the speed at which the pipes in game appear.

**Response to feedback**

I will add music to make the game feel more interactive but currently I have not been able to find any appropriate music which suits the game.

I have added collectibles which increase the players score. This gives the game an extra way of scoring points.

I have a short manual which show the keys used when playing the game.

I adjusted the rate at which pipes appear in the game window to make the game more difficult.

**Legal Constraints**

There are legal constraints which I have to take into account when creating this game. The code I used to create the game came was sourced on GitHub, so I would need to credit the original developer. Since some of the assets used within game are property of other game development companies, I would have to credit them if I was to ever release the game public. Due to the graphical effects being copyright claimed if I ever charged money for the game without the company knowing it could cause a huge lawsuit if discovered.

The assets which I need to give credit for are,

The ring sound effect - belongs to SEGA

The background - belongs to the artist

Obstacles (Pipes) - belongs to Nintendo

Avatar - belongs to original creator of flappy bird

**PEGI (Pan European Game Information)**

The game I have coded can be played by people of all ages. To decide the rating, I would give the game I approached some colleagues of mine and got them to test it. The general feedback I got from them was that the game was easy to play and if ever released it should have a rating of 3. This number is the lowest of the PEGI ratings therefore the game is considered suitable for all ages and the violence in game is comical at best. Which I believe is the best way to describe the game.

**Review of client requirements, evaluation of the design, development and optimisation process.**

The main requirement of this project was to create a game which appealed to a child between the ages of 8 to 12. Due to the game being rather simplistic in appearance and design I believe that I met that requirement with ease.

To make the game more professional I added a start menu. This allows the player not to immediately jump straight into the gameplay without any preparation. The second most important requirement for this project was to make sure the game interesting to play. Since pipes vary in height and the bird travels at the same speed, I added objects which the player needs to collect and made them appear at random positions on screen. Implementing collectibles makes the game more challenging since the player may need to collect an item at top of the screen and the gap the player needs to pass through maybe at the bottom of the screen.

To optimise the game, I removed unnecessary lines of code to make the game use less resources of the PC. I added comments alongside the code to make sure that whoever may come across it online would understand what I was doing when trying to program the game.