**Final Project**

**Algorithm and Programming**

A picture containing logo

Description automatically generated

Project Name: “Jump-Jump”

Student Name: Wallace Louis Tjang

Student ID:2602169705

1. Brief Description

Creating a game can be pretty hard and so I tried to search Python projects that I am able to create as this is a Final Project in creating a program with Python. Searching for days and weeks, I finally found a project that interests me and made me motivated to create. I came across a project of making and Endless Vertical Platformer which was for beginners in Python. The game was called Jumpy originally but with my own version I called it “Jump-Jump”

<https://www.youtube.com/watch?v=5FMPAt0n3Nc&t=3s>

This tutorial helped me create the basics of vertical platformer game which includes the physics of the game and how it works with the code. Dreaming to be a game developer in the future, I update more things to the game to make it more interesting.

The game that I created called Jump-Jump is a game where you will control a pink man, who is constantly jumping. Your objective is to jump from platform to platform to climb higher but make sure there is always a platform left to land on. When the game begins there will be distractions until the game ends. As you climb further up, the game becomes more difficult with as there will be moving platforms and obstacles. There is a score system keeps track of your current score and saves the high score.

Keys:

A - to go left

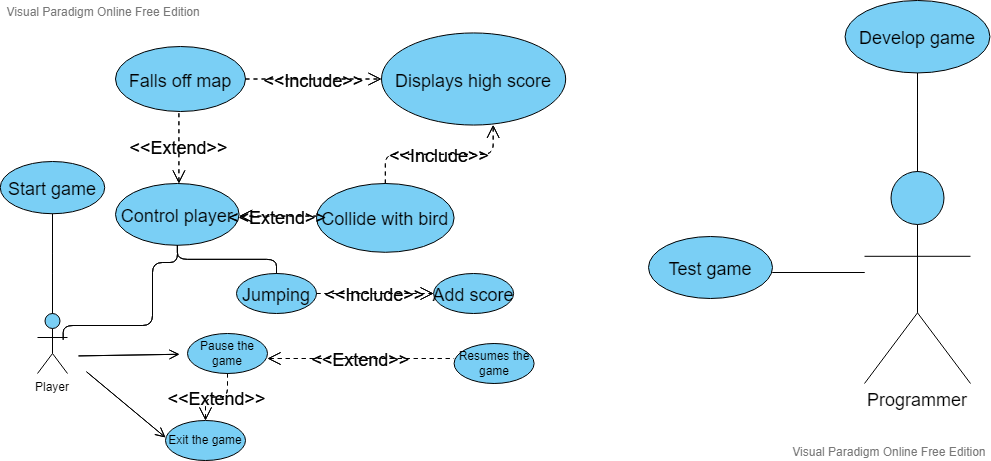
D – to go right

P – to pause the game

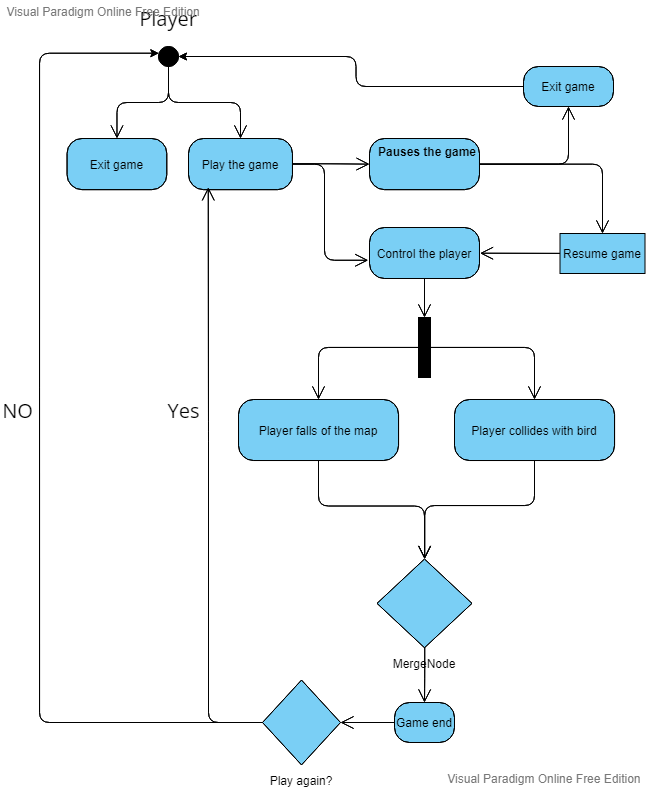
C – to continue the game

Q- to quit the game

1. Use-case Diagram



1. Activity Diagram



1. Class Diagram – name of the class, multiplicities and relationships

|  |  |  |
| --- | --- | --- |
| Player | Enemy | Platform |
| image  width  height  rect  rect.center  vel\_y  flip | animation\_list  frame\_index  update\_time  direction  flip  image  rect  animation\_steps  self.image  self.rect | Image  Moving  Move\_counter  Direction  Speed  Rect  Rect.x  Rect.y |
| move()  draw() | Update(scroll,screen\_width)  Draw(screen)  Empty() | Update()  Add(platform)  Update(scroll)  Draw(screen) |

|  |  |  |
| --- | --- | --- |
| Button | Button2 | SpriteSheet |
| Width  Height  Self.image  Self.rect  Self.rect.topleft  Self.clicked | Self.image  Self.scale  Self.image  Self.rect  Self.rect.x  Self.rect.y  Self.clicked | Self.sheet |
| Draw(surface) | Update\_image(img)  Draw(surface) | Get\_image(frame, width, height, scale, colour) |

|  |  |  |
| --- | --- | --- |
| Distraction |  |  |
| Self.animation.list  Self.frame\_index  Self.update\_time  Self.direction  Animation\_steps  Self.image  Self.rect |  |  |
| Update(scroll, SCREEN\_WIDTH) |  |  |

1. Modules (list down and short description)

jump-jump.py

mixer.init()

pygame.init()

These two functions initialized the pygame and mixer allowing the game to be loaded along with the music

SCREEN\_WIDTH = 400

SCREEN\_HEIGHT = 600

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption('Jumpy')

These functions are to set the width and height of the game screen and allows the game title at the bar to be named Jumpy

FPS: This is a variable to determine the game’s FPS. (60)

pygame.mixer.music.load: Loads the ingame music

pygame.mixer.music.set\_volume: Sets the volume of the music(0.6)

pygame.mixer.music.play: Plays the in game music (set to minus 1 so that the music repeats indefinitely) (-1, 0.0)

jump\_fx = pygame.mixer.Sound

jump\_fx.set\_volume

Load the sound effect when character jumps and set the volume

death\_fx = pygame.mixer.Sound

death\_fx.set\_volume(0.5)

Load the sound effect when character dies and set the volume

SCROLL\_THRESH = sets the max of the screen before background rolls again. (200)

GRAVITY = sets gravity of character (1)

MAX\_PLATFORMS = maximum platforms allowed in one background (10)

scroll = Start of the variable scroll ( 0)

bg\_scroll = Start of the variable scroll bg\_scroll (0)

game\_over = determines when game is over (Boolean: False)

score = Sets the beginning of the score (0)

fade\_counter = Sets the fade (game over screen transition) (0)

start\_game = determines if game has started (Boolean : False)

sound\_on = determines if music is (Boolean : True)

if os.path.exists('score.txt'):

with open('score.txt', 'r') as file:

high\_score = int(file.read())

else:

high\_score = 0

Read the score.txt if there are any high score saved yet. If not then high score is equal to 0

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

PANEL = (153, 217, 234)

Sets the variable for colours wanted (R,G,B)

font\_small = pygame.font.SysFont('Lucida Sans', 20)

font\_big = pygame.font.SysFont('Lucida Sans', 24)

Sets the variables for the fonts (font\_family,font\_size)

start\_img = pygame.image.load('start\_btn.png').convert\_alpha()

exit\_img = pygame.image.load('exit\_btn.png').convert\_alpha()

on = pygame.image.load('soundOnBtn.png').convert\_alpha()

off = pygame.image.load('soundOffBtn.png').convert\_alpha()

jumpy\_image = pygame.image.load('assets/jump.png').convert\_alpha()

bg\_image = pygame.image.load('assets/bg.png').convert\_alpha()

platform\_image = pygame.image.load('assets/wood.png').convert\_alpha()

distraction\_image = pygame.image.load('distraction.png').convert\_alpha()

Loads image of each variable (image)

bird\_sheet\_img = pygame.image.load('assets/bird.png').convert\_alpha()

bird\_sheet = SpriteSheet(bird\_sheet\_img)

Loads the bird image along with its Spritesheet

distraction\_sheet\_img = pygame.image.load('distraction.png').convert\_alpha()

distraction\_sheet = SpriteSheet(distraction\_sheet\_img)

Loads the disraction image along with its Spritesheet

def pause():

paused=True

while paused:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.KEYDOWN:

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

paused = False

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

pygame.quit()

quit()

screen.fill((255,255,255))

draw\_text('Paused',font\_big, BLACK, 150, 200)

draw\_text('Press C to continue or Q to quit.',font\_big, BLACK, 12, 300)

pygame.display.update() (Updates the function to be displayed)

clock.tick(5) (Sets the amount of frames for every secoond)

Function where paused= True when P is pressed (code at the bottom) and when player press C then game is continued (paused = False) where as Q game is quit (pygame.quit)

def draw\_text(text, font, text\_col, x, y):

img = font.render(text, True, text\_col)

screen.blit(img, (x, y))

def draw\_panel():

pygame.draw.rect(screen, PANEL, (0, 0, SCREEN\_WIDTH, 30))

pygame.draw.line(screen, WHITE, (0, 30), (SCREEN\_WIDTH, 30), 2)

draw\_text('SCORE: ' + str(score), font\_small, WHITE, 0, 0)

def draw\_bg(bg\_scroll):

screen.blit(bg\_image, (0, 0 + bg\_scroll))

screen.blit(bg\_image, (0, -600 + bg\_scroll))

Allows text and images to be drawn to the screen causing it to be displayed.

Player Class

\_\_init\_\_(self, x, y):

self.image = pygame.transform.scale

Defines image and resize image into new resolution

self.width = 25

Sets width of player

self.height = 40

Sets height of player

self.rect = pygame.Rect(0, 0, self.width, self.height)

Sets rectangluar of player (boundary) and then stores the coordinates

self.rect.center = (x, y)

Centers the rectangle

self.vel\_y = 0

Sets velocity of character

self.flip = False

Sets the image flip

move(self): changes position of player

scroll = 0

dx = 0

dy = 0

Variables of the player before game is started

key = pygame.key.get\_pressed()

if key[pygame.K\_a]:

dx = -10

self.flip = True

If you press A allowing the character moving to the left, then the character will move 10 px and then the image is flip from facing right to left

if key[pygame.K\_d]:

dx = 10

self.flip = False

If you press D allowing the character moving to the right, then the character will move 10 px and then the image will stay the same.

self.vel\_y += GRAVITY

dy += self.vel\_y

Logic where the velocity increases with gravity and dy

if self.rect.left + dx < 0:

dx = -self.rect.left

This makes character don’t go over the max of the left screen

if self.rect.right + dx > SCREEN\_WIDTH:

dx = SCREEN\_WIDTH - self.rect.right

This makes the character don’t go over the max of the right screen

platform.rect.colliderect(self.rect.x, self.rect.y + dy, self.width, self.height):

Check the collision of the player invisible rectangle with the platform.

if self.rect.bottom < platform.rect.centery:

if self.vel\_y > 0:

self.rect.bottom = platform.rect.top

dy = 0

self.vel\_y = -20

jump\_fx.play()

The logic is that when the bottom of the player’s invisible rectangle has touched the platform then the jump fx will play and then the player will jump again

if self.rect.top <= SCROLL\_THRESH

Allows to check if character top invisble rectangle has reached the SCROLL\_THRESH value

if self.vel\_y < 0:

scroll = -dy

Allows player to not fall with dy variable

self.rect.x += dx

self.rect.y += dy + scroll

Updates the invisible rectangle of character so that the game can keep on going

self.mask = pygame.mask.from\_surface(self.image)

Creates. Mask for the given surface used for fast pixel perfect collision detectio

draw(self):

screen.blit(pygame.transform.flip

Blits the players onto the game screen

Platform Class

\_\_init\_\_(self, x, y, width, moving):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.image = pygame.transform.scale(platform\_image, (width, 10))

Defines image and resize image into new resolution

self.moving = moving

Determines that platforms are moving or not

self.move\_counter = random.randint(0, 50)

Sets a random integer of how much the platforms will move

self.direction = random.choice([-1, 1])

Sets a random direction if platforms will be moving from left or right

self.speed = random.randint(1, 2)

Sets a random speed for the platforms

self.rect = self.image.get\_rect()

Sets a rectangle for the boundaries of the platform

self.rect.x = x

Sets variable x of rectangle

self.rect.y = y

Sets variable y of rectangle

update(self, scroll):

Updates the platforms status/ position

if self.moving == True:

self.move\_counter += 1

self.rect.x += self.direction \* self.speed

Sets the platform movement (checks that if its moving then go from left to right and back repeatedly)

if self.move\_counter >= 100 or self.rect.left < 0 or self.rect.right > SCREEN\_WIDTH:

self.direction \*= -1

self.move\_counter = 0

Allows platform to go left or right when hitting the game screen boundaries

self.rect.y += scroll

Updates the vertical position of the screen

if self.rect.top > SCREEN\_HEIGHT:

self.kill()

Checks if platform has gone off the screen by checking if the rectangle of platform is already higher than the screen height (self.kill() to remove the previous platform preventing overload).

start\_button = button.Button(SCREEN\_WIDTH // 2 - 130, SCREEN\_HEIGHT // 2 - 150, start\_img, 1)

exit\_button = button.Button(SCREEN\_WIDTH // 2 - 110, SCREEN\_HEIGHT // 2 + 50, exit\_img, 1)

sound\_btn = button.Button2(on, (24, 24), SCREEN\_WIDTH//2 - 15, SCREEN\_HEIGHT//2 + 200)

Creates the buttons for the main menu -> start and exit (python file, class used, width, height, image, value)

->sound(status,size,width,height)

platform\_group = pygame.sprite.Group()

enemy\_group = pygame.sprite.Group()

distraction\_group = pygame.sprite.Group()

Create a sort of lists to manage multiple sprites

platform = Platform(SCREEN\_WIDTH // 2 - 50, SCREEN\_HEIGHT - 50, 100, False)

platform\_group.add(platform)

Add amount of platform to group and then sets the starting position of the first platform on the screen

run = True

Determines wheter game is running or not (Boolean: True)

while run:

clock.tick(FPS)

if start\_game == False:

#draw menu

screen.fill((0,0,0))

if start\_button.draw(screen):

start\_game = True

if exit\_button.draw(screen):

run = False

if sound\_btn.draw(screen):

sound\_on = not sound\_on

This is to say that if the game hasn’t started (start\_game == False) then the start, exit and sound button should load on the screen on a black background

if sound\_on:

sound\_btn.update\_image(on)

pygame.mixer.music.unpause()

else:

sound\_btn.update\_image(off)

pygame.mixer.music.pause()

This is to say that if the song is on then the music should play and displays music playing button whereas if it is off the button will be updated to sound off image and song is paused

else:

if game\_over == False:

scroll = jumpy.move()

If game is over then you can’t move no more

bg\_scroll += scroll

if bg\_scroll >= 600:

bg\_scroll = 0

draw\_bg(bg\_scroll)

Blits the background and making sure character is not over the set value

if len(platform\_group) < MAX\_PLATFORMS:

p\_w = random.randint(40, 60)

Sets random width for platforms

p\_x = random.randint(0, SCREEN\_WIDTH - p\_w)

Sets random x position of platforms and making sure it doesn’t generate off the screen

p\_y = platform.rect.y - random.randint(80, 120)

Sets random y position of platforms and making sure it doesn’t generate off the screen

p\_type = random.randint(1, 2)

Set a random integer to determine if platform changes size or not

if p\_type == 1 and score > 500:

p\_moving = True

else:

p\_moving = False

platform = Platform(p\_x, p\_y, p\_w, p\_moving)

platform\_group.add(platform)

Allows platforms to be generated and says that if score is over 500 then the platforms will start moving

platform\_group.update(scroll)

Updates platform to screen

if len(distraction\_group) == 0:

distraction = Distraction(SCREEN\_WIDTH, 100, distraction\_sheet, 10)

distraction\_group.add(distraction)

distraction\_group.update(scroll, SCREEN\_WIDTH)

Updates distractions to screen and pops up from beginning of game

if len(enemy\_group) == 0 and score > 1500:

enemy = Enemy(SCREEN\_WIDTH, 100, bird\_sheet, 1.5)

enemy\_group.add(enemy)

Says that if your score is over 1500 and there are no enemies then generates enemies

enemy\_group.update(scroll, SCREEN\_WIDTH)

Updates enemy to screen

if scroll > 0:

score += scroll

Updates the score of user by taking in the scroll variable

pygame.draw.line(screen, WHITE, (0, score - high\_score + SCROLL\_THRESH), (SCREEN\_WIDTH, score - high\_score + SCROLL\_THRESH), 3)

Sets the line of previous high score

draw\_text('HIGH SCORE', font\_small, WHITE, SCREEN\_WIDTH - 130, score - high\_score + SCROLL\_THRESH)

Draws a High Score writing for game over screen

platform\_group.draw(screen)

enemy\_group.draw(screen)

distraction\_group.draw(screen)

jumpy.draw()

Draws the spites onto the screen

draw\_panel()

Draws the panel onto the screen

if jumpy.rect.top > SCREEN\_HEIGHT:

game\_over = True

death\_fx.play()

Checks if game is over by checking if the rectangle of player is over the max screen height then game ends and the death sound effect plays.

key = pygame.key.get\_pressed()

if key[pygame.K\_p]:

pause()

Says that if you press P, game will be paused

if pygame.sprite.spritecollide(jumpy, enemy\_group, False):

if pygame.sprite.spritecollide(jumpy, enemy\_group, False, pygame.sprite.collide\_mask):

game\_over = True

death\_fx.play()

else:

if fade\_counter < SCREEN\_WIDTH:

fade\_counter += 5

pygame.draw.rect(screen, BLACK, (0, y \* 100, fade\_counter, 100))

else:

draw\_text('GAME OVER!', font\_big, WHITE, 130, 200)

draw\_text('SCORE: ' + str(score), font\_big, WHITE, 130, 250)

draw\_text('PRESS SPACE TO PLAY AGAIN', font\_big, WHITE, 40, 300)

if score > high\_score:

high\_score = score

with open('score.txt', 'w') as file:

file.write(str(high\_score))

Checks if score has passed the current highscore by opening the txt filw

key = pygame.key.get\_pressed()

if key[pygame.K\_SPACE]:

When space is pressed, all variable will reset to the beginning

for event in pygame.event.get():

if event.type == pygame.QUIT:

Checks if user has quitted the game or not

if score > high\_score:

high\_score = score

with open('score.txt', 'w') as file:

file.write(str(high\_score))

Checks if the user score is higher than current highscore, then score will be overwritten to txt.

pygame.display.update() Allow a portion of the screen to be updated

pygame.quit() Quits the pygame

7. Screenshots of your application

Graphical user interface, text, application

Description automatically generatedQr code

Description automatically generated

8. Lessons learned/Reflection

Lesson learned

Through this session, I learned that everything that we have learned in the Algorithm and Programming has been very useful in creating the Python game. The exercise where we had to make the Alien Invasion game helped a lot because it helped me to train my logic. The parts such as the scoring and how to move left and right. The exercise in the class such as defining functions, creating classes, functions and loops also helped the project quite a lot. I learned that having an idea of a game and implementing it into coding is not as easy as it sounds. As a programmer, you gotta learn many kinds of libraries that can be used in coding. The errors are sometimes quite hard to understand because it is usually caused by misunderstanding or lack of knowledge. Coding this game definitely help me a lot in understanding that even a game that’s a virtual endless plat-former is very complex.

Python has become more interesting because it’s just very diverse and there’s always logic in every single element. In the jumping movement you put in a variable called Gravity, in the movement you use x and y to limit the movements on how far a player can jump and much more. Positioning the elements was quite hard cause my head can’t just compute how do you specifically position the stuffs you want at the right position without making it messy. Sprite was something cool because it was like the object that we control and that made like the animations come out such as the moving bird, platforms and players. Coding is actually tiring and respect to the programmers out there who can create great triple A games.

Reflection

Coming into this semester, I was blind to coding. I didn’t know what Python was. For this semester we were tasked to create a Python game and with 0 experience it was quite a heavy task. I am weak in terms of coding and I hope that in the second semester I can focus more into learning deeper stuffs in coding in order to improve and graduate university. I should also learn soft skills such as design and strengthening my logic because game physics can sometimes trick people’s mind and make them go crazy.