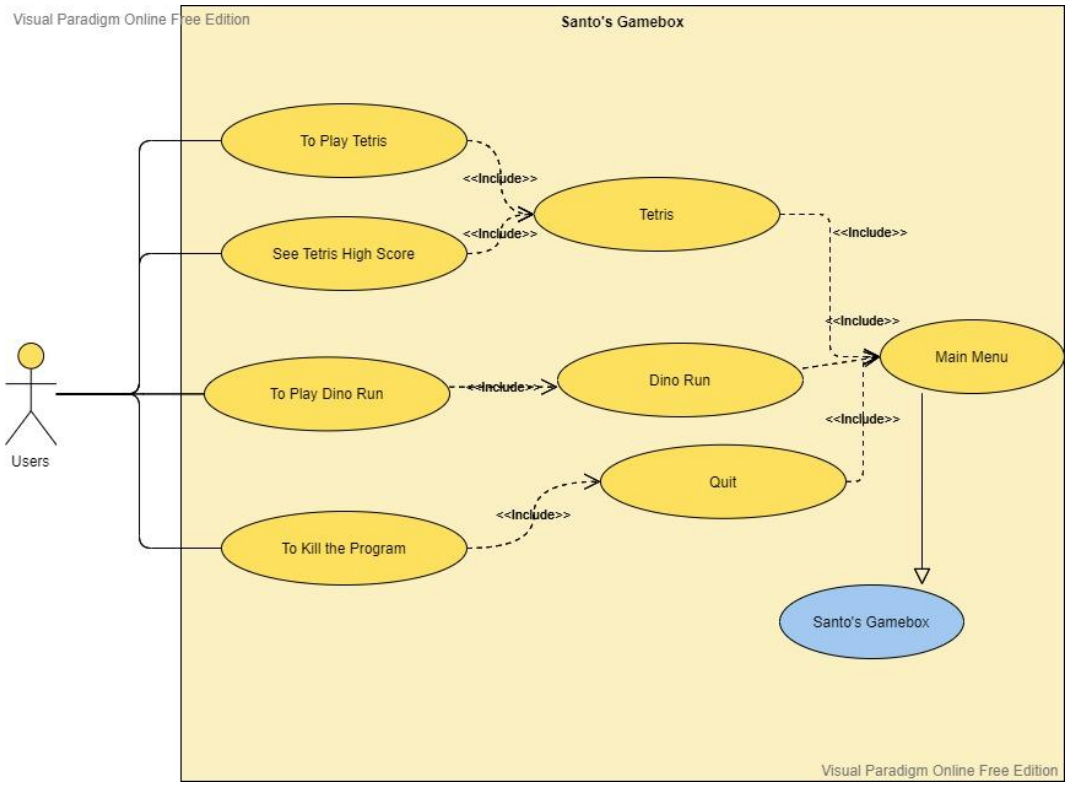
A. Santo’s Gamebox - Algorithm and Programming Final Project

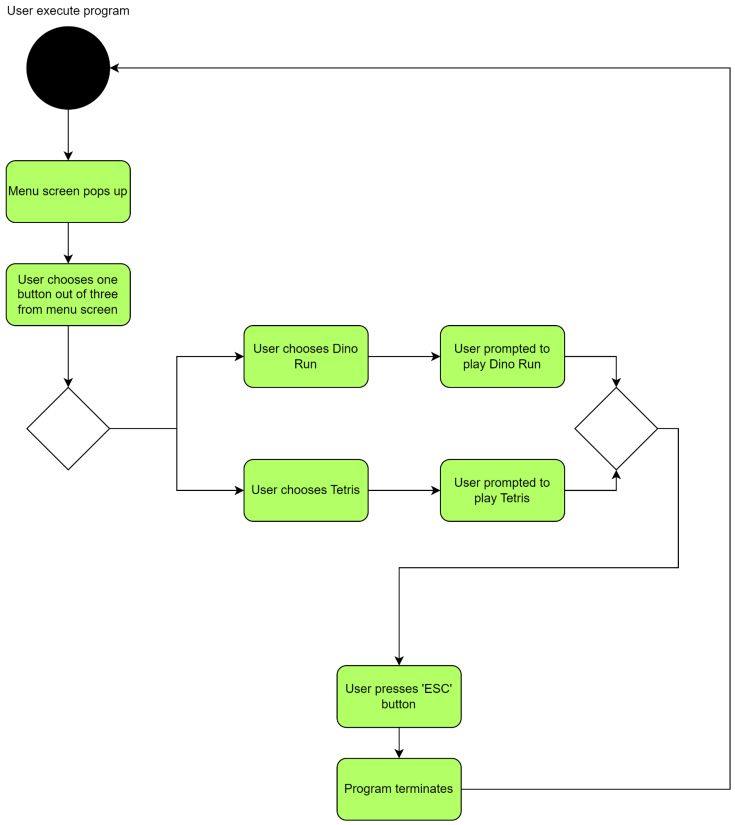
1. Santo’s Gamebox is an app that stores mini games for users to be able to play to kill their boredom. The games include games such as the running dinosaur game that originally belongs to Chrome and also another game that is a clone of the original Tetris game. The main purpose of this application or program is to help people kill their boredom by having a selection of games to play with.
2. The Use Case Diagram of this program would be as follows:



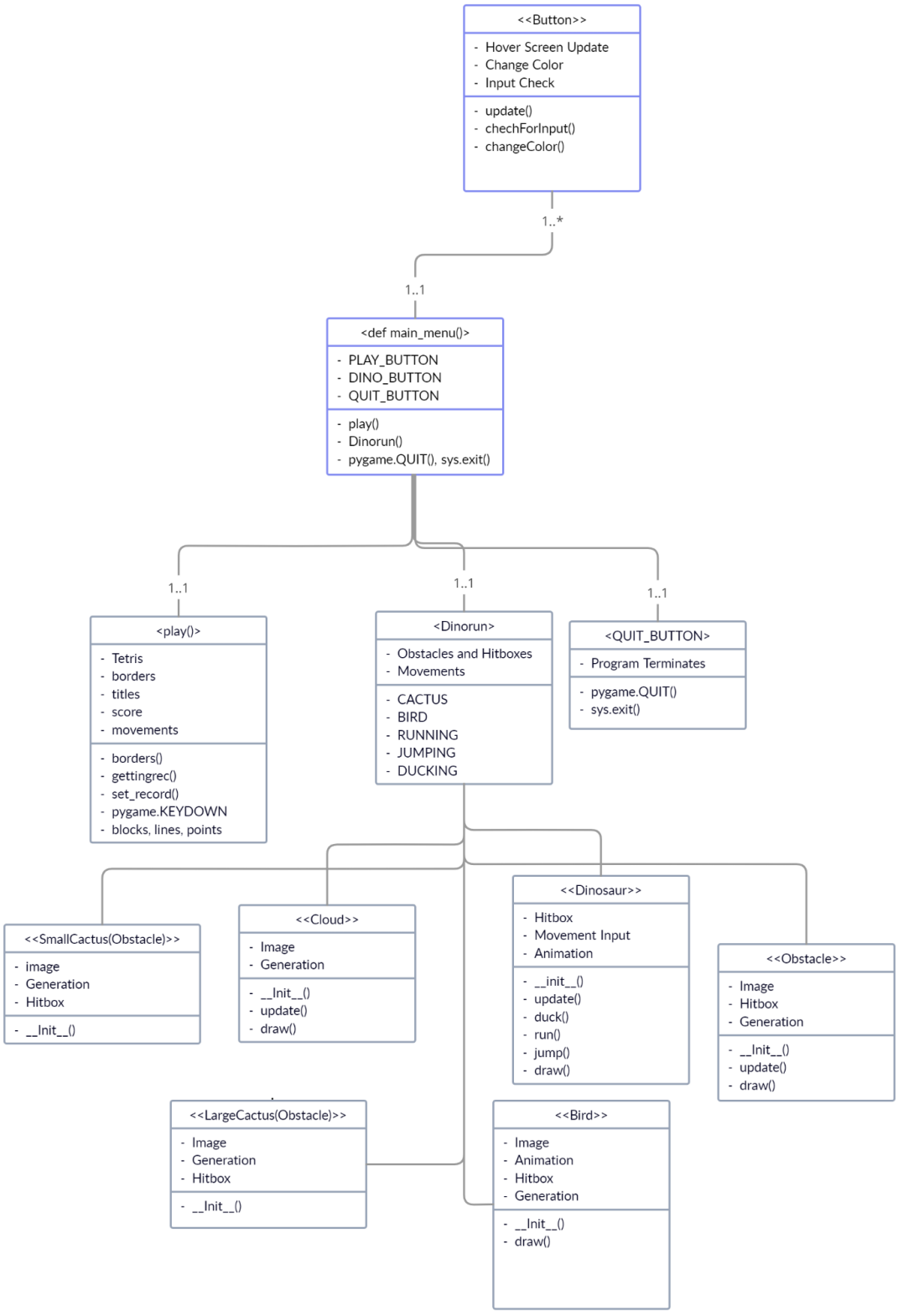
i.

3. The Activity Diagram of this program would be as follows:

i.



4. The class diagram of this program would be as follow:



i.

5. The modules that I have used in this program are as follows :
    i. pygame

ii. sys
iii. os
iv. copy
v. Random

The mostly used module would be pygame as it is the base module or the base module for the whole program. Pygame is used as a form of input regarding the movements of both games and it is also used to set the resolution, the screen, the display, the animation of both games and many more. The sys module is used to help terminate the game properly whenever users want to exit the gamebox or quit playing the games. The os module is used to help find the locations or path of images that are used in the game as a part of the background, animation, and others. The copy module, especially deepcopy, is used to act as a moving animation or moving the blocks themselves in the game of Tetris. Lastly, the random module is used mainly as a random generator for the blocks in the game of Tetris and also as the random generator for the obstacles in the Dino Run game.

6. Some of the essential algorithms are as follows :

i.
```
blocks_pos = [[(-1, 0), (-2, 0), (0, 0), (1, 0)],
              [(0, -1), (-1, -1), (-1, 0), (0, 0)],
              [(-1, 0), (-1, 1), (0, 0), (0, -1)],
              [(0, 0), (-1, 0), (0, 1), (-1, -1)],
              [(0, 0), (0, -1), (0, 1), (-1, -1)],
              [(0, 0), (0, -1), (0, 1), (1, -1)],
              [(0, 0), (0, -1), (0, 1), (-1, 0)]]

blocks = [[pygame.Rect(x + W // 2, y + 1, 1, 1) for x, y in fig_pos] for fig_pos in blocks_pos]
block_rect = pygame.Rect(0, 0, TILE - 3, TILE - 4)
```

This algorithm is used as a map for the blocks in the Tetris game

ii.
```
field = [[0 for i in range(W)] for k in range(H)] #this will be the map of the tetris blocky area

anim_count, anim_speed, anim_limit = 0, 60, 2000

bg = pygame.image.load('imgs/blue.png').convert()
game_bg = pygame.image.load('imgs/black.jpg').convert()

main_font = pygame.font.Font('font.ttf', 65)
font = pygame.font.Font('font.ttf', 45)
```

The combination of this many algorithms are responsible for the map of the playing field of the game, the speed and animation of the game, the background and wallpaper of the game, and the texts and fonts of the game of Tetris.

iii.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        exit()
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_LEFT:
            dx = -1
        elif event.key == pygame.K_RIGHT:
            dx = 1
        elif event.key == pygame.K_DOWN:
            anim_limit = 100
        elif event.key == pygame.K_UP:
            rotate = True
        elif event.key == pygame.K_ESCAPE:
            main_menu()
            pygame.quit()
# x-axis movement
block_old = deepcopy(block)
for i in range(4):
    block[i].x += dx
    if not borders():
        block = deepcopy(block_old)
        break
# y-axis movement
anim_count += anim_speed
if anim_count > anim_limit:
    anim_count = 0
    block_old = deepcopy(block)
    for i in range(4):
        block[i].y += 1
        if not borders():
            for i in range(4):
                field[block_old[i].y][block_old[i].x] = color
            block, color = next_block, next_color
            next_block, next_color = deepcopy(choice(blocks)), get_color()
            anim_limit = 2000
            break
```

This algorithm is used for the movement in the game of Tetris alongside the quit button.

iv.

```
SCREEN_HEIGHT = 720
SCREEN_WIDTH = 1280
SCREEN = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
#Movement
RUNNING = [pygame.image.load(os.path.join("Assets/Dino", "DinoRun1.png")),
        pygame.image.load(os.path.join("Assets/Dino", "DinoRun2.png"))]
JUMPING = pygame.image.load(os.path.join("Assets/Dino", "DinoJump.png"))
DUCKING = [pygame.image.load(os.path.join("Assets/Dino", "DinoDuck1.png")),
        pygame.image.load(os.path.join("Assets/Dino", "DinoDuck2.png"))]
#Obstacle
SMALL_CACTUS = [pygame.image.load(os.path.join("Assets/Cactus", "SmallCactus1.png")),
                pygame.image.load(os.path.join("Assets/Cactus", "SmallCactus2.png")),
                pygame.image.load(os.path.join("Assets/Cactus", "SmallCactus3.png"))]
LARGE_CACTUS = [pygame.image.load(os.path.join("Assets/Cactus", "LargeCactus1.png")),
                pygame.image.load(os.path.join("Assets/Cactus", "LargeCactus2.png")),
                pygame.image.load(os.path.join("Assets/Cactus", "LargeCactus3.png"))]

BIRD = [pygame.image.load(os.path.join("Assets/Bird", "Bird1.png")),
        pygame.image.load(os.path.join("Assets/Bird", "Bird2.png"))]

CLOUD = pygame.image.load(os.path.join("Assets/Other", "Cloud.png"))

BG = pygame.image.load(os.path.join("Assets/Other", "Track.png"))
```

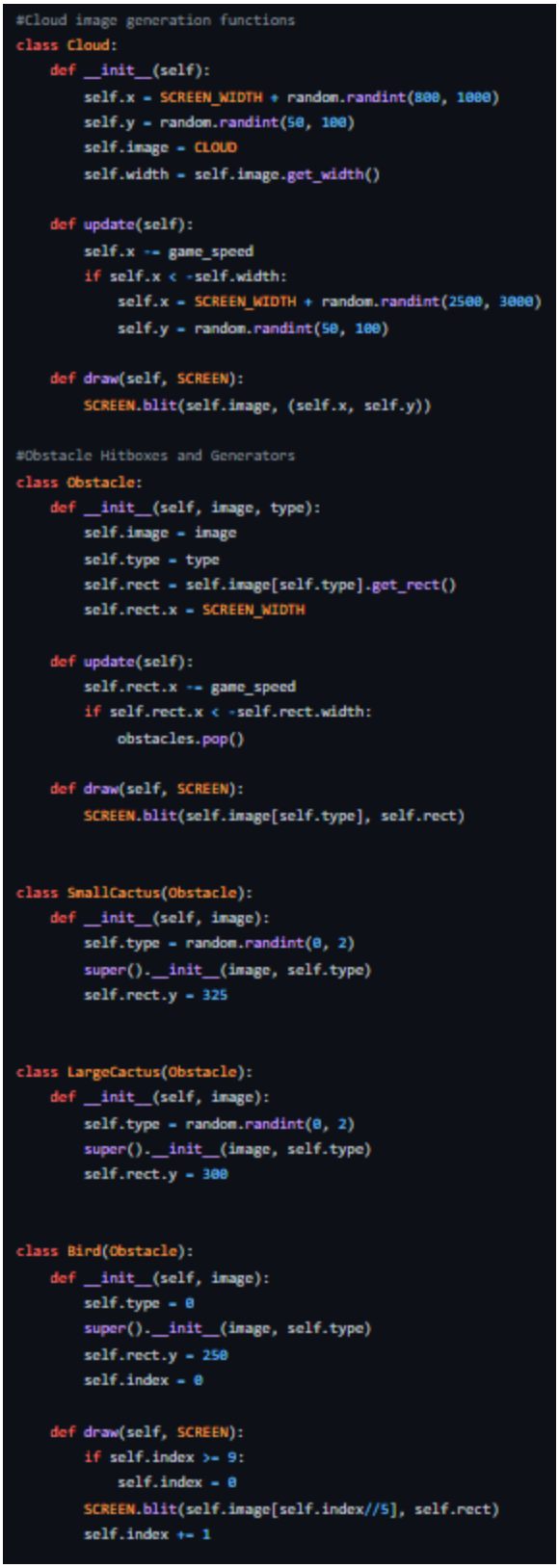
This combination of algorithms is used to set the base images for every position in the Dino Run game such as jumping, running, and ducking alongside the base image of the obstacles.

v.

```
def update(self, userInput):
    if self.dino_duck:
        self.duck()
    if self.dino_run:
        self.run()
    if self.dino_jump:
        self.jump()

    if self.step_index >= 10:
        self.step_index = 0

    if userInput[pygame.K_UP] and not self.dino_jump:
        self.dino_duck = False
        self.dino_run = False
        self.dino_jump = True
    elif userInput[pygame.K_DOWN] and not self.dino_jump:
        self.dino_duck = True
        self.dino_run = False
        self.dino_jump = False
    elif not (self.dino_jump or userInput[pygame.K_DOWN]):
        self.dino_duck = False
        self.dino_run = True
        self.dino_jump = False
```

This whole algorithm is responsible for the movement input in the Dino Run game.

vi.



```
#Cloud image generation functions
class Cloud:
    def __init__(self):
        self.x = SCREEN_WIDTH + random.randint(800, 1000)
        self.y = random.randint(50, 100)
        self.image = CLOUD
        self.width = self.image.get_width()

    def update(self):
        self.x -= game_speed
        if self.x < -self.width:
            self.x = SCREEN_WIDTH + random.randint(2500, 3000)
            self.y = random.randint(50, 100)

    def draw(self, SCREEN):
        SCREEN.blit(self.image, (self.x, self.y))

#Obstacle Hitboxes and Generators
class Obstacle:
    def __init__(self, image, type):
        self.image = image
        self.type = type
        self.rect = self.image[self.type].get_rect()
        self.rect.x = SCREEN_WIDTH

    def update(self):
        self.rect.x -= game_speed
        if self.rect.x < -self.rect.width:
            obstacles.pop()

    def draw(self, SCREEN):
        SCREEN.blit(self.image[self.type], self.rect)


class SmallCactus(Obstacle):
    def __init__(self, image):
        self.type = random.randint(0, 2)
        super().__init__(image, self.type)
        self.rect.y = 325


class LargeCactus(Obstacle):
    def __init__(self, image):
        self.type = random.randint(0, 2)
        super().__init__(image, self.type)
        self.rect.y = 300


class Bird(Obstacle):
    def __init__(self, image):
        self.type = 0
        super().__init__(image, self.type)
        self.rect.y = 250
        self.index = 0

    def draw(self, SCREEN):
        if self.index >= 9:
            self.index = 0
        SCREEN.blit(self.image[self.index//5], self.rect)
        self.index += 1
```

This combination of algorithm is responsible for the hitboxes, the

animation, and the random generation of the obstacles that are available in the Dino Run game.

vii.

```
while run:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    SCREEN.fill((255, 255, 255))
    userInput = pygame.key.get_pressed()

    player.draw(SCREEN)
    player.update(userInput)

    if len(obstacles) == 0:
        if random.randint(0, 2) == 0:
            obstacles.append(SmallCactus(SMALL_CACTUS))
        elif random.randint(0, 2) == 1:
            obstacles.append(LargeCactus(LARGE_CACTUS))
        elif random.randint(0, 2) == 2:
            obstacles.append(Bird(BIRD))

    for obstacle in obstacles:
        obstacle.draw(SCREEN)
        obstacle.update()
        if player.dino_rect.colliderect(obstacle.rect):
            pygame.time.delay(100)
            death_count += 1
            menu(death_count)

    background()
    cloud.draw(SCREEN)
    cloud.update()
    score()
    clock.tick(30)
    pygame.display.update()
```

This algorithm is the main algorithm that runs the Dino Run game.

viii.

```
def menu(death_count):
    global points
    run = True
    while run:
        SCREEN.fill((255, 255, 255))
        font = pygame.font.Font('freesansbold.ttf', 30)
        userInput = pygame.key.get_pressed()

        if death_count == 0:
            text = font.render("Press SPACE to Start", True, (0, 0, 0))
        elif death_count > 0:
            text = font.render("Press SPACE to Restart or ESC to Stop", True, (0, 0, 0))
            score = font.render("Your Score: " + str(points), True, (0, 0, 0))
            scoreRect = score.get_rect()
            scoreRect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2 + 50)
            SCREEN.blit(score, scoreRect)
        textRect = text.get_rect()
        textRect.center = (SCREEN_WIDTH // 2, SCREEN_HEIGHT // 2)
        SCREEN.blit(text, textRect)
        SCREEN.blit(RUNNING[0], (SCREEN_WIDTH // 2 - 20, SCREEN_HEIGHT // 2 - 140))
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            if userInput[pygame.K_SPACE]:
                main()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    main_menu()


menu(death_count=0)
```

This whole algorithm is what's responsible for the death screen of the Dino Run game.

ix.

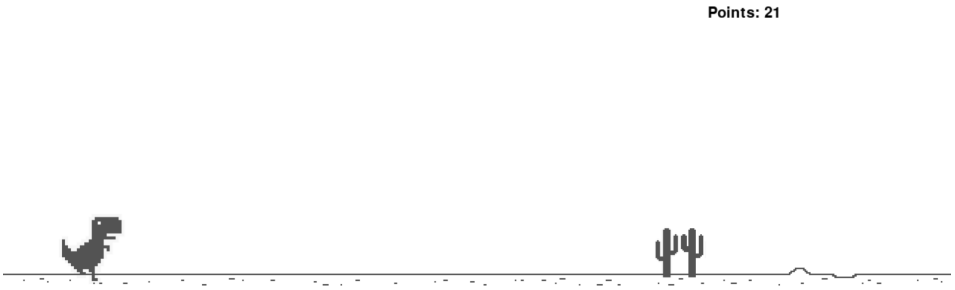
```
def main_menu():
    while True:
        SCREEN.blit(BG, (0, 0))

        MENU_MOUSE_POS = pygame.mouse.get_pos()

        MENU_TEXT = get_font(70).render("Santo's Gamebox", True, "#b68f40")
        MENU_RECT = MENU_TEXT.get_rect(center=(640, 100))

        PLAY_BUTTON = Button(image=pygame.image.load("assets/Options Rect.png"), pos=(640, 250),
                             text_input="Tetris", font=get_font(75), base_color="#d7fcd4", hovering_color="White")
        DINO_BUTTON = Button(image=pygame.image.load("assets/Options Rect.png"), pos=(640, 400),
                             text_input="Dino Run", font=get_font(65), base_color="#d7fcd4", hovering_color="White")
        QUIT_BUTTON = Button(image=pygame.image.load("assets/Quit Rect.png"), pos=(640, 550),
                             text_input="QUIT", font=get_font(75), base_color="#d7fcd4", hovering_color="White")

        SCREEN.blit(MENU_TEXT, MENU_RECT)

        for button in [PLAY_BUTTON, DINO_BUTTON, QUIT_BUTTON]:
            button.changeColor(MENU_MOUSE_POS)
            button.update(SCREEN)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if PLAY_BUTTON.checkForInput(MENU_MOUSE_POS):
                    play()
                if DINO_BUTTON.checkForInput(MENU_MOUSE_POS):
                    Dinorun()
                if QUIT_BUTTON.checkForInput(MENU_MOUSE_POS):
                    pygame.quit()
                    sys.exit()

        pygame.display.update()

main_menu()
```

This combination of algorithms is what composes the main menu screen.

7. Here are some screenshots of what the program would actually look like :

i.

ii. 



iii.

8. What I have learned through building this program :
    i. Throughout creating this program, I've learned how important it is to actually have a goal on what the code was supposed to do and how it was supposed to act. One of my biggest challenges on creating this program is on what the code was supposed to look like and how to make them. I've also learned a lot of new things that you can apparently do with python and how python can be used to actually play games and process data. I had so much fun creating this program and I had wished that I had more time to add in a lot more games to the program. But in the end, I got a satisfying result so I am grateful to be able to create this program.

B. The source code of this program can be found using this link down below :
    1. https://github.com/SAD-Nich/Santo-sGamebox-AlgoPro-Final-Project.git

C. A demo video of this program can also be found using this link down below :
    1. https://drive.google.com/file/d/1NsiZluKc-dQ9I96ed93UUwbj2YxeKf5R/view?usp=share_link