OROCHIMARU

Programming Fundamental Project

Syed Muhammad Mustafa (06554)

Submitted to Dr. Abdul Samad

Contents

Introduction

The game 'Orichimaru' is a recreation of the popular snake game in mobile phones in the early 2000's. The game employs the python library, Pygames to display a window and use graphics to represent a snake and a fruit. The rules of the game are as follows:

1. If the snakes head touches its body then the game ends.
2. If the snake touches the boundary of the window the game ends.
3. Each time the snake eats a fruit it grows one block and score is incremented by one.
4. Initially the score is zero.
5. The objective of the game is to increase the size of the snake.

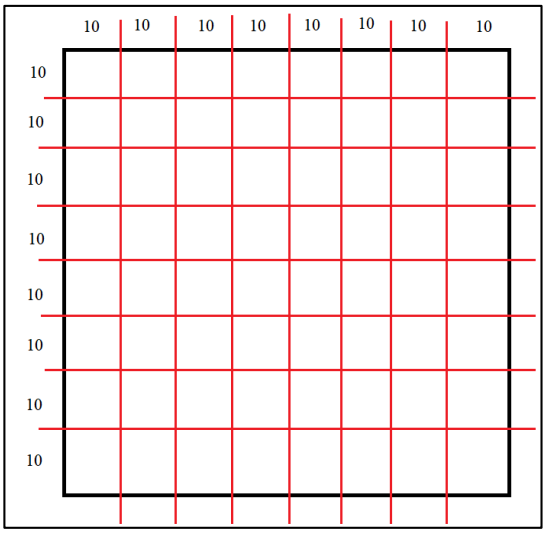
Methodology

1. Simulating a Grid:

- We have simulated a grid/map on our display window, so that we can map the positions of objects we intend to use in the game i.e. snake and fruit.

```
#Setting the board
box_area = 25
box_num = 20 #This is done to simulate a grid.
window = pygame.display.set_mode((box_area*box_num,box_area*box_num))
caption = pygame.display.set_caption('Snake Game ')
clock = pygame.time.Clock()
game_font = pygame.font.Font(None,25)
```

- The output of a game using Pygame is displayed on a window. We will create the window display and assign that object to the variable Windows. This was done by using the function pygame.display.set_mode((Tuple with the dimensions)).
- For the tuple for the dimensions of the window we created to variables box_num and box_area. Box_num is the number of boxes for a single axes in our simulated grid, and box_area is the length of one side of the box in the simulated grid.
- The figure above shows an example of the simulated grid. In this example the box_area is 10 and the box_num is 8.



2. Main Game Loop:

- The main game loop is an infinite while loop, where all the magic happens. A game is basically a set of moving pictures, so for the output to make sense it is necessary that the game is constantly updated i.e. the image at the previous location be deleted and the next location be kept. This creates an illusion for the user that the image, in this case snake, is moving. This updating is the main function of the main game loop. The main game loop also handles the user key input for movement of the snake. We learned this idea of using while loop for the basic framework (a logic loop) for the game through a YouTube tutorial [1]

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

        if event.type == pygame.KEYDOWN:
            if game_engine.snake.direction.x == 1 or game_engine.snake.direction.x == -1:
                if event.key == pygame.K_UP:
                    game_engine.snake.direction = Vector2(0, -1)
                if event.key == pygame.K_DOWN:
                    game_engine.snake.direction = Vector2(0,+1)
            elif game_engine.snake.direction.y == 1 or game_engine.snake.direction.y == -1:
                if event.key == pygame.K_RIGHT:
                    game_engine.snake.direction = Vector2(1, 0)
                if event.key == pygame.K_LEFT:
                    game_engine.snake.direction = Vector2(-1,0)
# when i did not write the nested if statement the snake was able to change his direction in a manne



        if event.type == continous_motion:
            game_engine.update()

        window.fill(blue)
        game_engine.drawing_fruit_and_snake()
```

- Pygame.event is basically a dictionary of all the events that occur in the game. We use the for loop and pygame.event.get() to temporarily store the event values in the event variable, using the line, $for\ event\ in\ pygame.event.get()$:
- Using this we can use the event variable to monitor user input and apply different conditional statements so that the program reacts to different inputs in a different manner. For instance, if the cross on the window is pressed the $pygame.quit()$ line is executed which ends the game or, the snake will change directions if the arrow keys are pressed.
- A problem that came up while we were programming the game was that the snake would be able to change its direction from left to right and vice versa or from up to down and vice versa. This meant that the snake was able to move its head through its body. To fix this we used a nested condition which basically meant that the snake could only turn to right or left if it was moving up or down and vice versa.
- We used $Vector2()$ function after importing it from $pygame.maths$. We used $Vector2()$ method instead of using a list because we found it to be more convenient. In a list to access the x value we would have to write (for example) $list[0]$. However, in $Vector2()$ method all we had to do was (for example) $vector.x$. Moreover addition and subtraction in the vector method is relatively easier.

3. Continuous Motion of the Snake:

- In the figure below it is shown that a custom event was created and subsequently stored in the continuous_motion variable. Then using the pygames timer function, the continuous_motion event was programmed in such a way that it occurred after every 100 nanosecond.
- Then we placed a conditional statement in the for loop that if event is equal to the continuous_motion event, i.e. 100 nanoseconds have passed, the update() method of the class MAINLOOP runs. The update() method is responsible for the movement of the snake, check if the conditions for the death of the snake is met, and if snake has eaten the fruit.

```
    if event.type == continous_motion:
        game_engine.update()
```

```
continous_motion = pygame.USEREVENT #custom event
pygame.time.set_timer(continous_motion,100)
```

4. Use of Classes and Object Oriented Programming:

- In the code there are three classes MAINLOOP, SNAKE, and FRUIT.
- A Class is like an object constructor, or a "blueprint" for creating objects.
- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created .
- Python Object Oriented Programming (OOP) - For Beginners is the online tutorial we used to understand the basics of object oriented programming. [2]

5. SNAKE Class:

```
class SNAKE():
    def __init__(self):
        self.bodypart = [Vector2(5,10),Vector2(4,10),Vector2(3,10),Vector2(2,10)]
        self.direction = Vector2(1 , 0)
        self.new_block = False
        self.eating_sound = pygame.mixer.Sound(r'C:\Users\Yousuf Traders\Desktop\P

    def draw_snake(self):···

    def move(self):···

    def add_snake(self):···

    def eat_sound(self):···
```

- The SNAKE is initialized with $self.bodypart$, $self.direction$, $self.new\_block$, and $self.eating\_sound$ attributes.

- The body of the snake is basically a list of vectors, using the $Vector2()$ method. The snake starts with four vectors, with same y value and adjacent x values. The first vector in the list ($self.bodypart[0]$) is treated like the head of the snake.
- There are four methods in the SNAKE class: $draw\_snake()$,$move()$,$add\_snake()$,$eat\_sound()$.
- The method $draw\_snake()$ is used draw the snake object. This by making by drawing a rectangle at each vector position in the $self.bodypart$ list. The rectangle is drawn using the $pygame.draw.rect(surface, color, dimensions)$. Each individual vector is excessed using the for loop.

```
def draw_snake(self):
    for block in self.bodypart:
        x = block.x * box_area
        y = block.y * box_area
        body_part_rect = pygame.Rect(x, y , box_area , box_area)
        pygame.draw.rect(window, red, body_part_rect)
```

- The movement of the snake and the growth of the snake is done using the methods of $move()$,and $add\_snake()$. What normally happens is the $bodypart$ list is sliced into the variable new_pos such that the last element of the list is not included. And the head of the snake i.e. the first element of the $bodypart$ list is added with $self.direction$ attribute, which initially is the vector (+1, 0) (direction is towards the right, this attribute changes as the user presses the arrow keys) creating a new position for the head.
- If the condition of snake and the fruit overlapping is met then the attribute $self.new.block$ becomes true. This causes the condition in the $move()$ method to execute in which new_pos is given the entire $bodypart$ list with the new position of the head inserted in the list. Then the $self.new\_block$ attribute becomes false, this is done to limit the snake growing only by one block.

```
def move(self):
    if self.new_block == True:
        new_pos = self.bodypart[:]
        new_pos.insert(0,new_pos[0] + self.direction)
        self.bodypart = new_pos[:]
        self.new_block = False # so that the snake do
    else:
        new_pos = self.bodypart[:-1] #we do this slic
        new_pos.insert(0,new_pos[0] + self.direction)
        self.bodypart = new_pos[:]

def add_snake(self):
    self.new_block = True
```

- The program plays a eating sound every time the snake and fruit over laps. It is important to know that the file with the audio should be saved in the same folder as the python file.

6. FRUIT Class:

- The FRUIT class acts like a blue print for the fruit object to be created. The class is initialize using the $\_\_init\_\_(\ self\ )$ method. In it the $self.change\_pos\_fruit()$ method is initialize, this means that this method will be called each time the FRUIT class instance is used. By doing this we would be able to change the position of the fruit if certain condition is met.
- The fruit object is displayed in the game using the $window.blit(\ )$ function, which displays an image (.PNG) file in the specified position. However, it is necessary to note, the image should be loaded in the program using $image.load(\ )$ function. Much like the audio file, the image file should also be in the same folder as the python file.
- The position of the fruit randomly changes after the snake overlaps the fruit. This is done by using the $random.randint()$ function (from the random python module) in the method $change\_pos\_fruit(\ )$.

```
class FRUIT():
    def __init__(self):
        self.change_pos_fruit() #By initializing this method we will be able to change the position of the fruit whe
        #create a square
    def drawing_fruit(self):
        fruit_rect = pygame.Rect(self.pos_fruit.x*box_area,self.pos_fruit.y*box_area,box_area/2,box_area/2)
        # pygame.draw.rect(window,green,fruit_rect) #if decide to change box to apple, load image, then use the blit
        window.blit(fruit_img,fruit_rect)

    def change_pos_fruit(self ):
        self.x = random.randint(0,box_num - 1)
        self.y = random.randint(0,box_num -1)
        #create an x and y coordinate| we will be using pygame.math.Vector2() for this, vector, we use vector becaus
        self.pos_fruit = Vector2(self.x,self.y)
```

7. MAINLOOP Class:

```
class MAINLOOP():
    def __init__(self):
        self.snake = SNAKE()
        self.fruit = FRUIT()

    def update(self):        #we have use

    def drawing_fruit_and_snake(self):

    def score(self):  #report

    def checkfruit_eat(self):

    def snake_death(self):

    def crashed(self):
```

- The MAINLOOP class is initialized using the $\_\_init\_\_(self)$ method. In it the $self.snake$ and $self.fruit$ become instances of class SNAKE and class FRUIT respectively.
- The $update()$ method is called in the main game loop after the condition of the custom user event ($continuous\_motion$). This method calls three methods, one from the class SNAKE and two from class MAINLOOP. The one from SNAKE is $self.snake.move()$, and the two from MAINLOOP are $self.snake\_death()$ and $checkfruit\_eat()$.

```
def update(self):        #we
    self.snake.move()
    self.checkfruit_eat()
    self.snake_death()
```

- The $checkfruit\_eat()$ runs a conditional statement, which checks if position of head of the snake and the fruit is same. If the condition is true, the $change\_pos\_fruit()$ method of the class FRUIT, $add\_snake()$ and $eat\_snake()$ from the class SNAKE

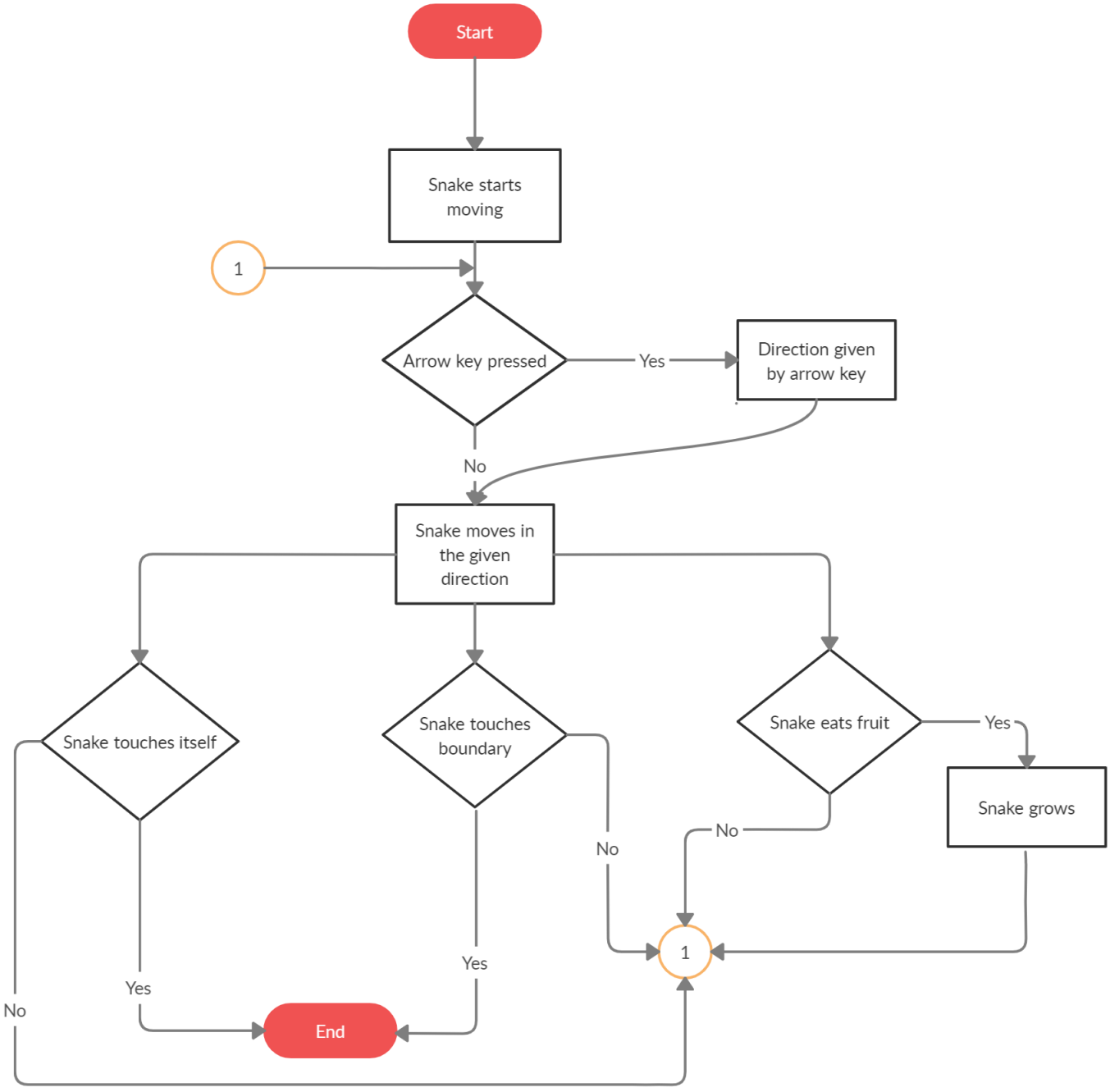
```
def checkfruit_eat(self):
    if  self.fruit.pos_fruit ==  self.snake.bodypart[0]:  #everything that hppens after snake munches on the fruit
        self.fruit.change_pos_fruit()
        self.snake.add_snake()# Ading another block to snake after eating
        self.snake.eat_sound()

def snake_death(self):
    if  self.snake.bodypart[0].x >= box_num  or self.snake.bodypart[0].x < 0 or self.snake.bodypart[0].y >= box_num  or self.snake.bodypart[0].y < 0:
        self.crashed() #if the snake is out of bounds

    #if the snake touches it self
    for block in self.snake.bodypart[1:]:
        if block == self.snake.bodypart[0]:
            self.crashed()
```

- The game will end if any one of two condition is met:
  - Snakes head overlaps any part of its body. Note, it is necessary to slice the $bodypart$ list such that it doesn't include the first element of the list. If this condition is met the program ends using the $pygame.quit()$ function.
  - The snake touches the boundary of the window. It is important to note that the condition should include both x and y of the $bodypart[0]$(head of the snake).
- The score in terms of the game is how many fruits the snake ate, which is equal to four units less than the length of the snake. The score is displayed in the bottom right corner of the window. The score is displayed using $windows.blit()$.
  A variable with the font object must be created in the program before $render()$ function is used. In the program the variable $game\_font$ is used for this purpose.

```
def score(self):  #report
    score_txt = str(len(self.snake.bodypart)-4)
    score = game_font.render(score_txt, True,red)
    s_x = int(box_area*box_num - 40)
    s_y = int(box_area*box_num - 30)
    scorerect = score.get_rect(center = (s_x,s_y))
    window.blit(score,scorerect)
```

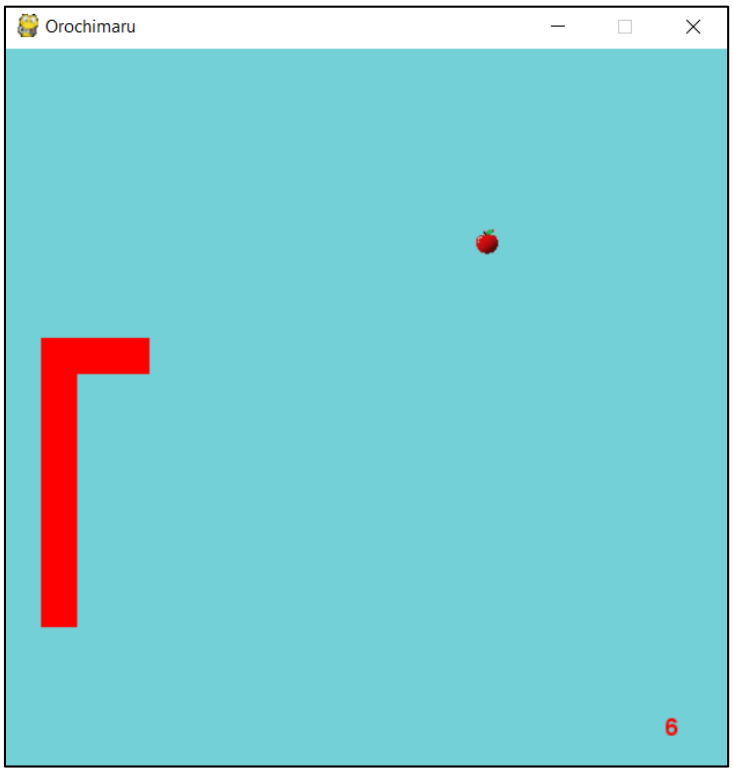
- The MAINLOOP function is used to snake and fruit. The MAINLOOP is initialized by calling instances of SNAKE and FRUIT. Then in the method $drawing\_fruit\_and\_snake()$ the methods from SNAKE and FRUIT are called to draw the respective figures.
- In the code $game\_engine$ is an instance of the class MAINLOOP.

Flowchart



Conclusion

The output of the code:



As it can be seen in the in the figure above, after running the program a window named Orochimaru pops up. With a red figure (the snake), an apple (fruit), and a score on the bottom right.

All the *pygame* syntax was learned from Pygames official website. [3].

References

[1] sentdex, "Game Development in Python 3 With PyGame - 1 - Intro," YouTube, 2014.

[2] Tech With Tim, "Python Object Oriented Programming (OOP) - For Beginners," Youtube, 2020.

[3] "pygame," pygame, [Online]. Available: https://www.pygame.org/docs/. [Accessed 26 November 2020].