Project Report

MiniGames

By:

Pothana Sravan Chaitanya

sravanchaitanya2003@gmail.com

Lovely Professional University, Punjab

Reg: 12017778

Table of Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty Chandan Mukherjee sir for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry me a long way in the journey of life on which I am about to embark.

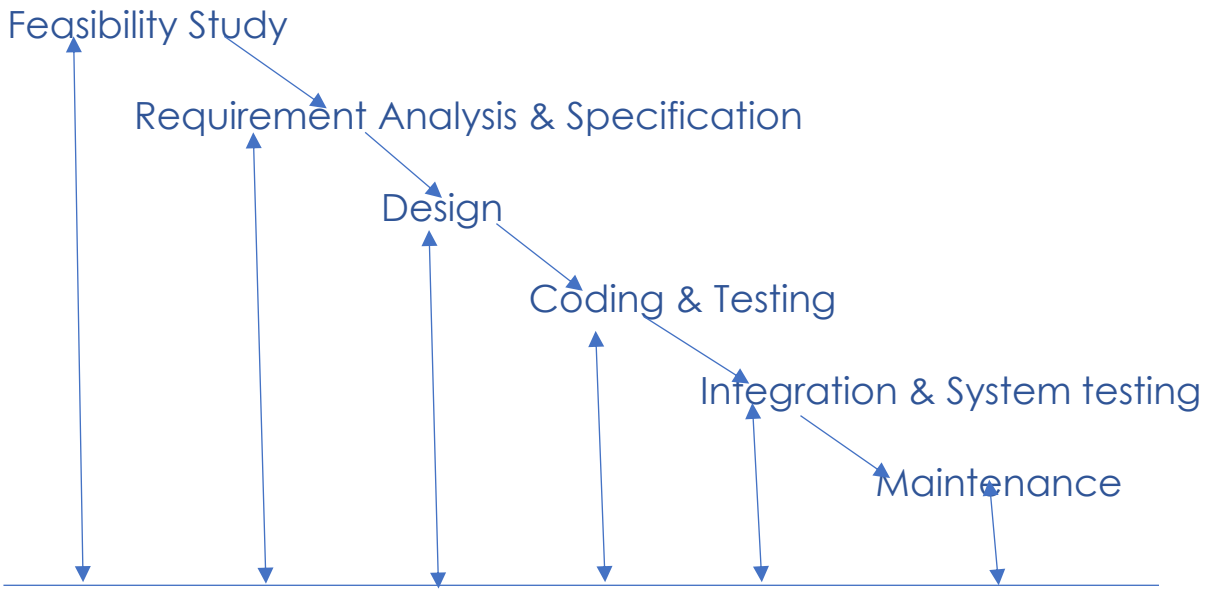
Introduction

This project is a simple Gaming Application built upon Tkinter and Pygame modules in python. In this project, I have built a Graphical User Interface (GUI) using Tkinter module. This GUI consists of various minigames which user can select and play. Each game visible on this GUI is built upon Pygame module. Until now, I've created three games. They are Flappy Bird, Snake and Tic Tac Toe.

Objective

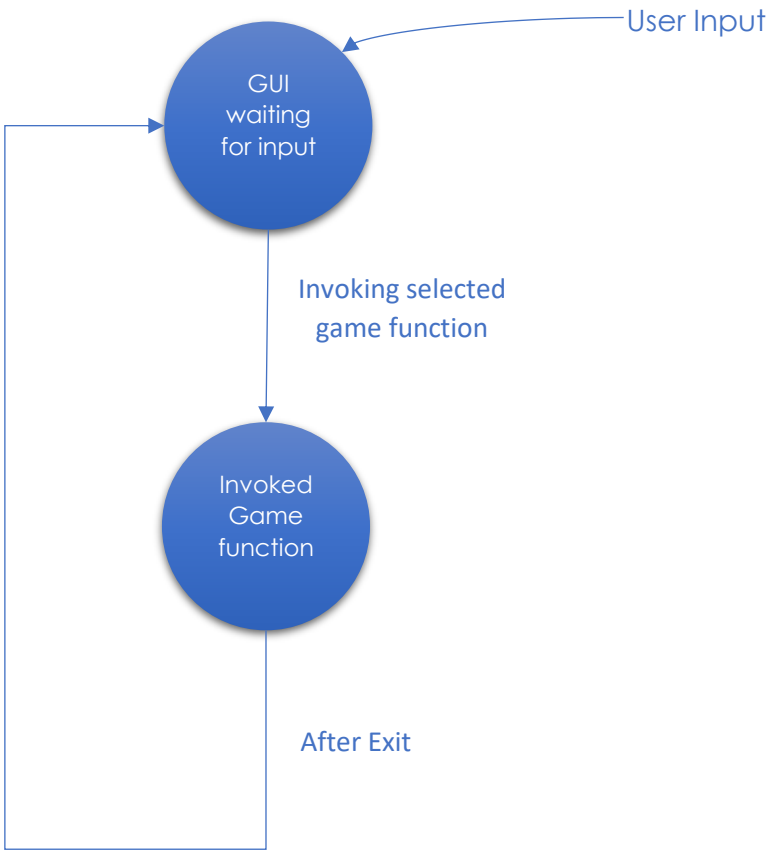
The objective of the project is to create a GUI that can display multiple minigames to the User. And developing minigames and integrating them, such that the main GUI can load them when the User wants to play.

Software Engineering Paradigm Applied

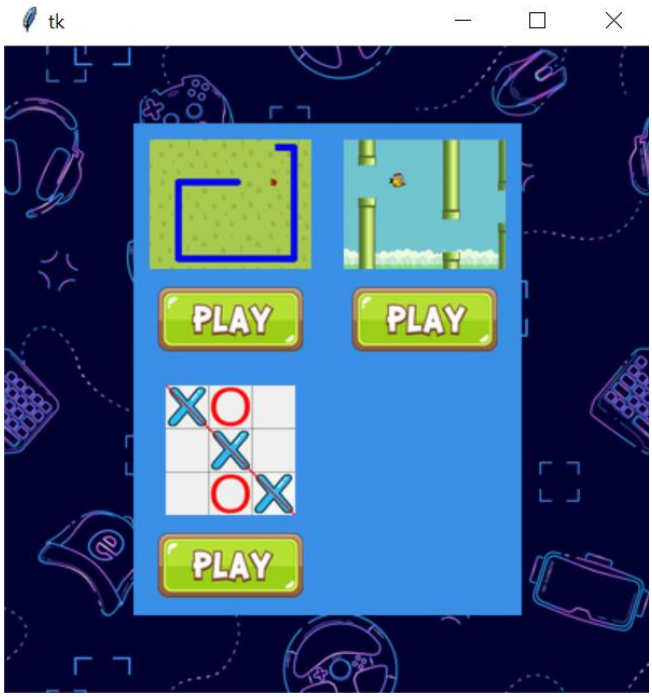


Application Workflow

In this application, each game is enclosed in a function. GUI manages all these functions. The flow of the application starts from this GUI. GUI is build using Tkinter. It consists of thumbnail of games along with the button that invokes the game function.



Screenshot of GUI



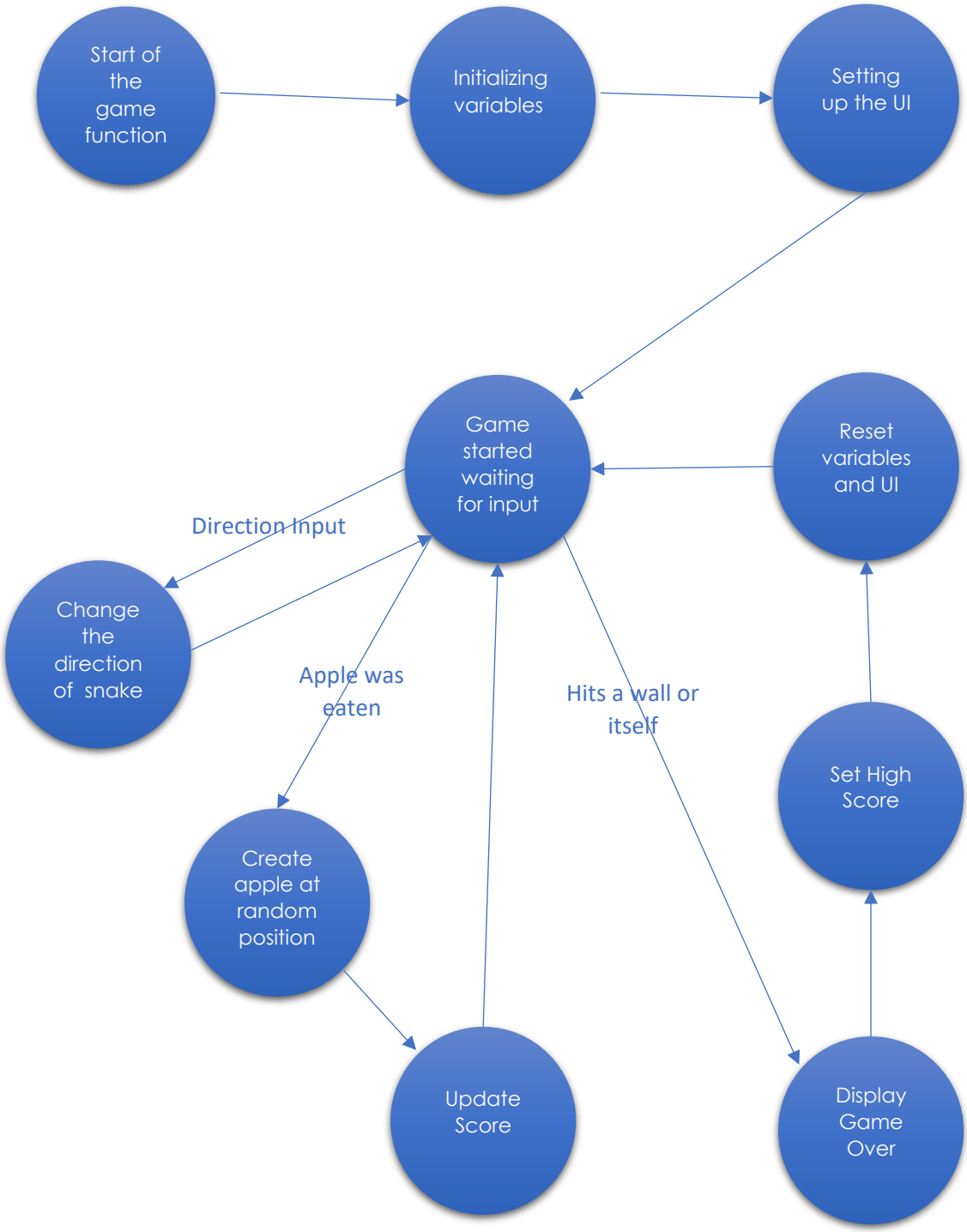
Workflow of Individual Games

Snake

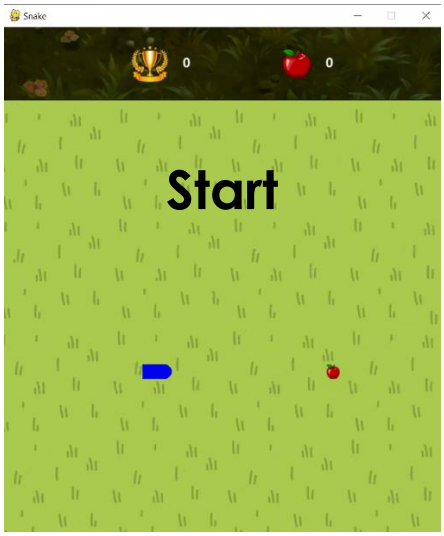
This game is built using Pygame module in python. Gameplay is simple. User need to guide a snake to reach its food (Apple) using arrow keys. Each time snake eats an apple, it grows a little. Apple is generated using random module. The game comes to an end when the snake hits the wall or hits itself. The score of the user is the number of apples the snake has eaten. High score is also

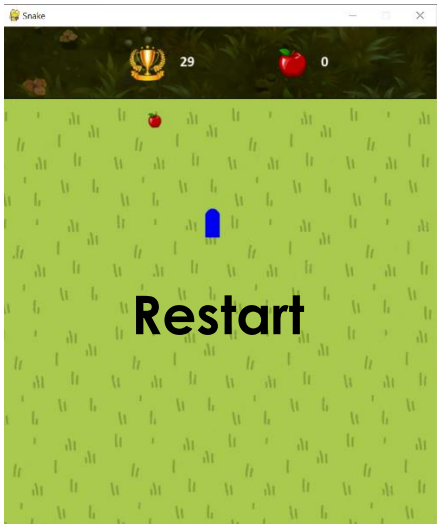
maintained throughout the running of the application.

Workflow



Screenshot of Snake GUI

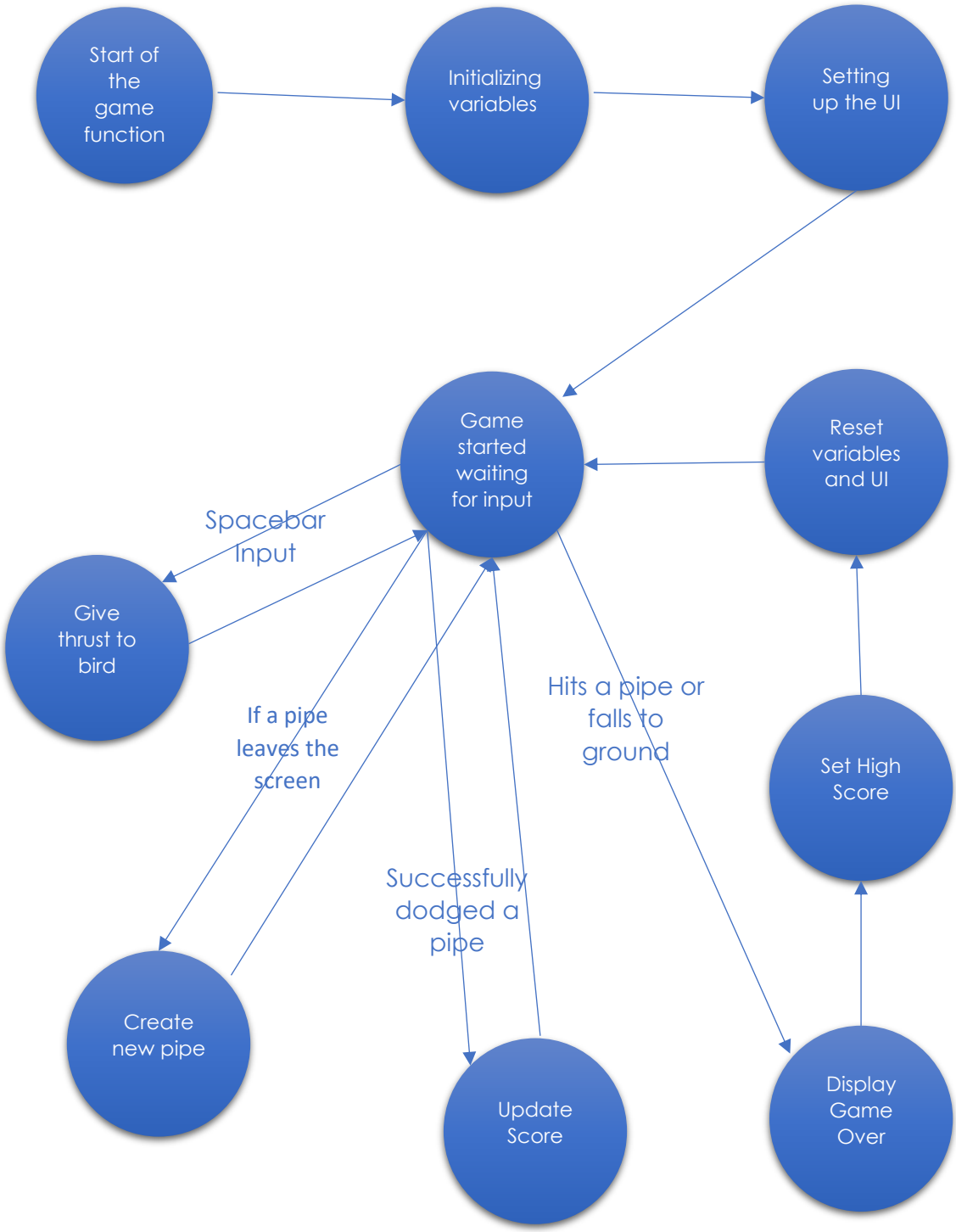




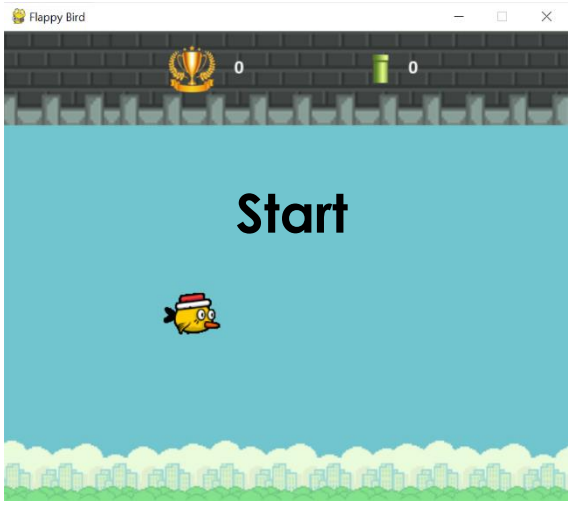
Flappy Bird

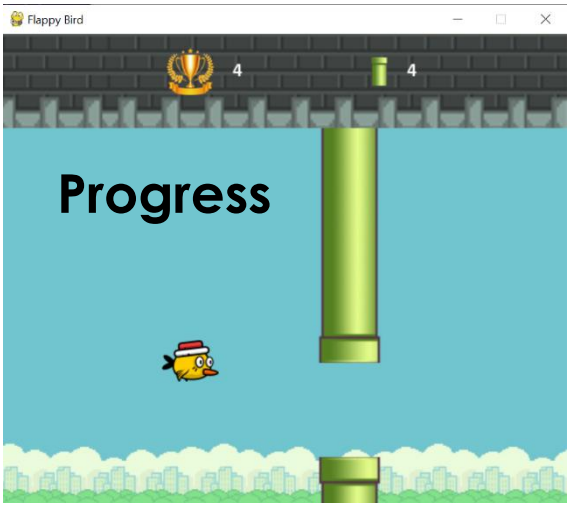
This is another minigame built using Pygame module. Gameplay is pretty simple. A bird flies in a tunnel trying to dodge some pipes in its way. When user presses the space bar, a thrust is given to the bird and it flies some height else it starts falling. User's work is to give the thrust at a specific time so that the bird is able to dodge the pipes. The game ends when the bird hits a pipe or falls to the ground. Score of the user is the number of pipes the bird has dodged.

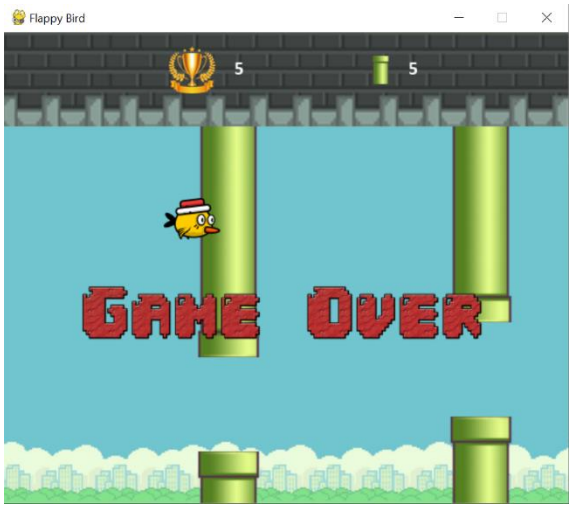
Workflow

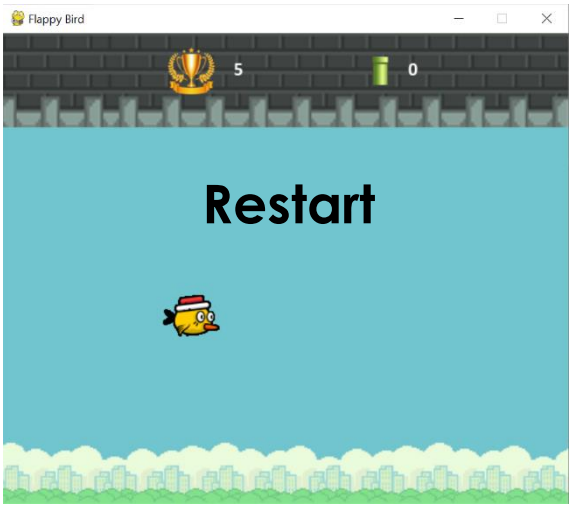

Start of the game function
Initializing variables
Setting up the UI
Game started waiting for input
Reset variables and UI
Spacebar Input
Give thrust to bird
If a pipe leaves the screen
Hits a pipe or falls to ground
Set High Score
Successfully dodged a pipe
Create new pipe
Update Score
Display Game Over

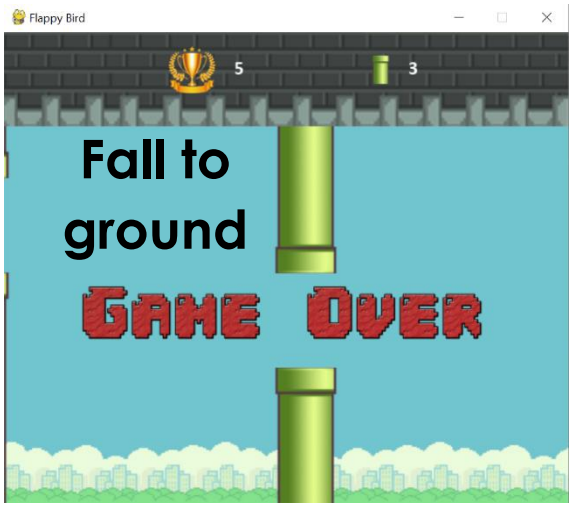
Screenshot of Flappy Bird GUI





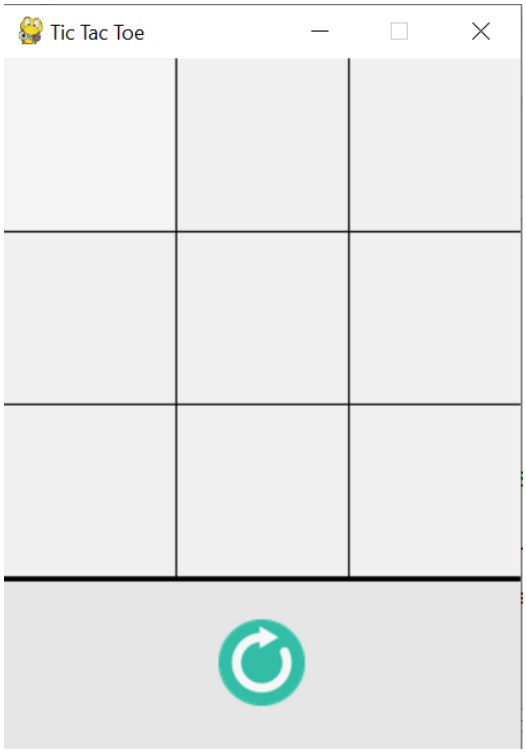


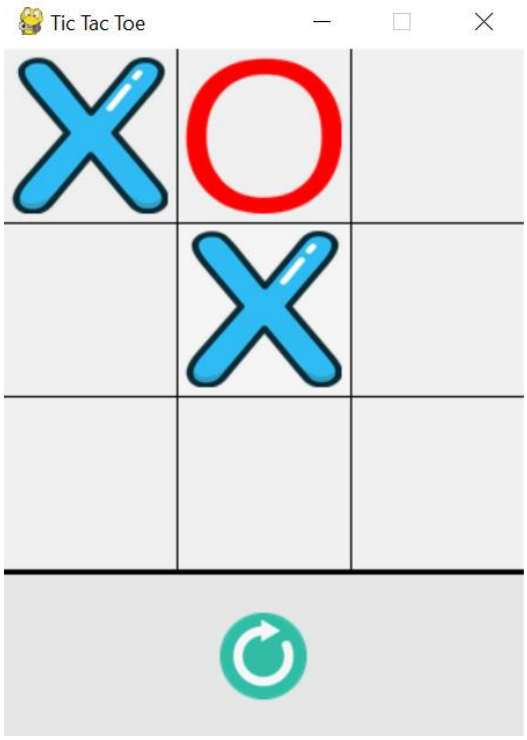


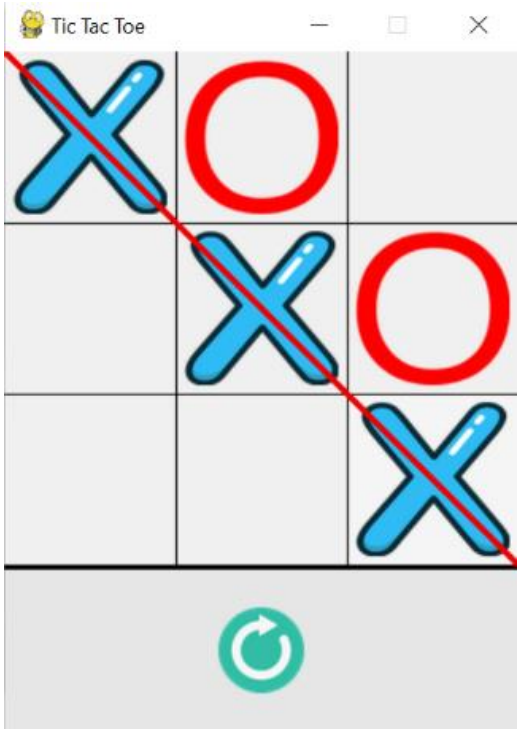


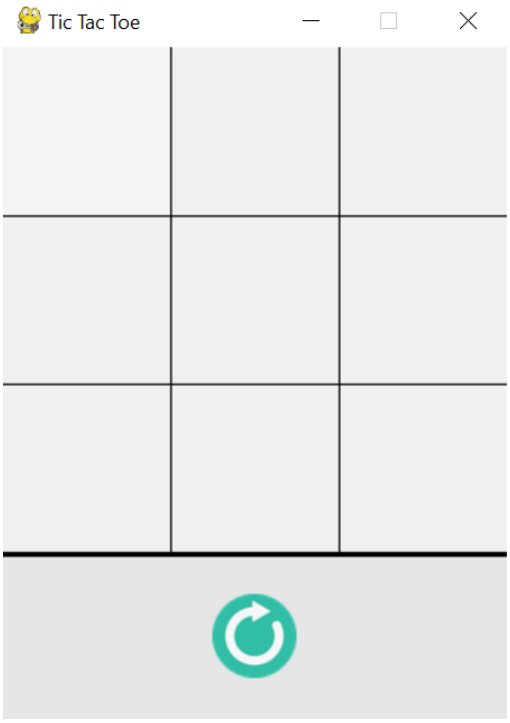
Tic Tac Toe

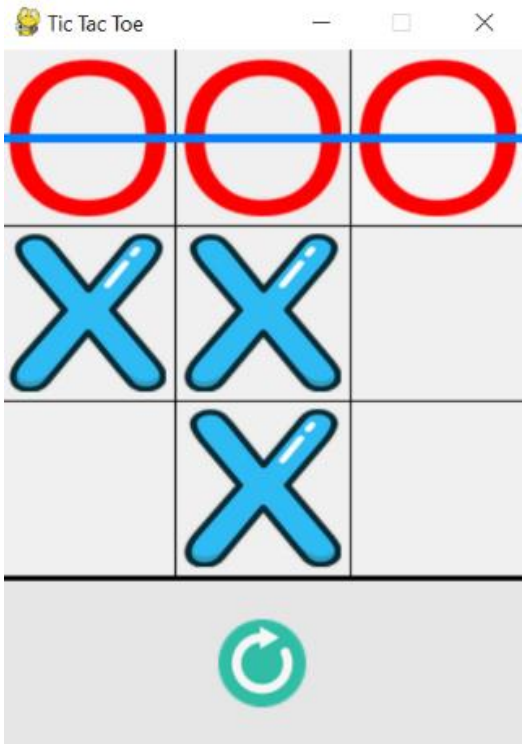
This is one of the most popular games in the world. My version of the game is built using Pygame. Most of the us know the gameplay. This game consists of 9 clickable areas. Two users can play at a time. Each is assigned either symbol X or O. In their respective turn, each can select an unoccupied area. Based on their respective symbol, the clicked is marked. If any user has managed to mark a full row or a full column or any diagonal with his respective symbol, then he is considered as winner. If both the users weren't able to do so then it is considered as a tie.



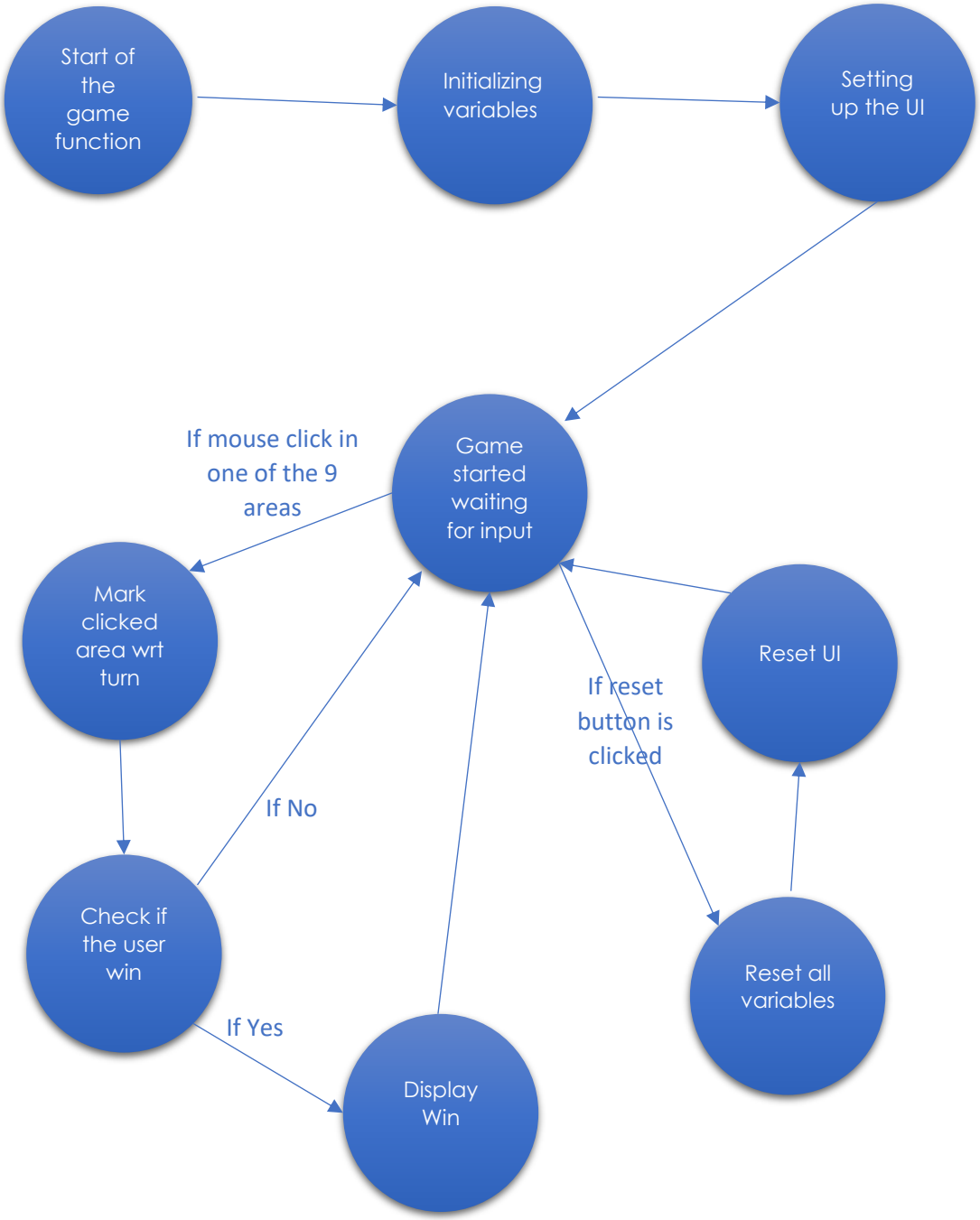



Tic Tac Toe


Tic Tac Toe


Tic Tac Toe

Workflow


Start of the game function
Initializing variables
Setting up the UI
Game started waiting for input
If mouse click in one of the 9 areas
Mark clicked area wrt turn
Check if the user win
If No
If Yes
Display Win
If reset button is clicked
Reset UI
Reset all variables

Future Scope of Improvements

- If other games were developed, they can be added to the Main GUI.
- Performance of games can be improved.
- AI based user can be generated as an opponent using minimax algorithm.
- Obstacles can be made dynamic in flappy bird.
- Appearance of snake can be improved in snake game.

Code

The whole program is coded in Jupyter Notebook.

Main GUI

```
snake_highscore = 0
flappy_bird_highscore = 0

def snake_init():
    snake()
def flappy_bird_init():
    flappy_bird()
def tictactoe_init():
    tictactoe()
def create_game_frame(command,display,dim):
    frame = Frame(gameframe,bg = '#3a90e6')
    image1 = Image.open(display)
    image1 = image1.resize(dim,Image.ANTIALIAS)
    image1 = ImageTk.PhotoImage(image1,master = frame)
    label = Label(frame,image = image1,bd = 0)
    label.image = image1
    image2 = Image.open('Main UI images\play.png')
    image2 = image2.resize((90,40),Image.ANTIALIAS)
    image2 = ImageTk.PhotoImage(image2,master = frame)
    button = Button(frame,command = command,image = image2,bd = 0,cursor = 'hand2',bg = '#3a90e6')
    button.image = image2
    label.grid(row = 0,column = 0,pady = 10)
    button.grid(row = 1,column = 0)
    return frame
```

```
root = Tk()
root.minsize(400,400)
root.maxsize(400,400)
root_bg = Image.open('Main UI images\\root_bg.jpg')
root_bg = root_bg.resize((400,400),Image.ANTIALIAS)
root_bg = ImageTk.PhotoImage(root_bg,master = root)
Label(root,image = root_bg).pack()
gameframe = Frame(root,bg = '#3a90e6')
snake_game = create_game_frame(snake_init,'Main UI images\snake_display.png',(100,80))
snake_game.grid(row = 0,column = 0,padx = 10)
flappy_bird_game = create_game_frame(flappy_bird_init,'Main UI images\\flappy_bird_display.png',(100,80))
flappy_bird_game.grid(row = 0,column = 1,padx = 10)
tictactoe_game = create_game_frame(tictactoe_init,'Main UI images\\tictactoe_display.png',(80,80))
tictactoe_game.grid(row=1,column=0,padx = 10,pady =10)
gameframe.place(relx = 0.5,rely = 0.5,anchor = 'center')
root.mainloop()
```

Snake

```
def snake():
    global snake_highscore
    def create_text(text,fg,bg=None,font = 'calibri',size=20):
        Font = pg.font.SysFont(font,size,bold = True)
        Text = Font.render(text,True,fg,bg)
        return Text
    def convert_to_surface(image):
        mode = image.mode
        size = image.size
        data = image.tobytes()
        return pg.image.fromstring(data,size,mode)

    pg.init()
    screen = pg.display.set_mode((600,700))
    pg.display.set_caption('Snake')
    blue = (0,0,240)
    white = (255,255,255)
    green = (0,240,0)
    gamespace = [0,100,600,700]

    screen_bg = pg.image.load('screen_bg.jpg')
    screen_bg_rect = screen_bg.get_rect()
    screen_bg_rect.topleft = (gamespace[0],gamespace[1])

    score_board = pg.Surface((600,100))
    score_board_rect = score_board.get_rect()
```

```
score_board_bg = pg.image.load('grass_bg.png')
score_board_bg_rect = score_board_bg.get_rect()
score_board_bg_rect.topleft = (0,0)

trophy = Image.open('trophy.png')
trophy = trophy.resize((50,50),Image.ANTIALIAS)
trophy = convert_to_surface(trophy)
trophy_rect = trophy.get_rect()
trophy_rect.center = (score_board_rect.width/3,score_board_rect.height/2)

big_food=Image.open('apple.png')
big_food = big_food.resize((40,40),Image.ANTIALIAS)
big_food = convert_to_surface(big_food)
big_food_rect = big_food.get_rect()
big_food_rect.centerx = score_board_rect.width*2/3
big_food_rect.centery = trophy_rect.centery

small_food = Image.open('apple.png')
small_food = small_food.resize((18,20),Image.ANTIALIAS)
small_food = convert_to_surface(small_food)
small_food_rect = small_food.get_rect()

game_over = pg.image.load('game_over.png')
game_over_rect = game_over.get_rect()

highscore_value = snake_highscore
score_value = 0
```

```
highscore_text = create_text(str(highscore_value),white)
highscore_rect = highscore_text.get_rect()
highscore_rect.left = trophy_rect.right+20
highscore_rect.centery = trophy_rect.centery
score_text = create_text(str(score_value),white)
score_rect = score_text.get_rect()
score_rect.left = big_food_rect.right+20
score_rect.centery = big_food_rect.centery

score_board.blit(score_board_bg,score_board_bg_rect)

score_board.blit(trophy,trophy_rect)
score_board.blit(big_food,big_food_rect)

score_board.blit(highscore_text,highscore_rect)
score_board.blit(score_text,score_rect)

screen.blit(screen_bg,screen_bg_rect)

screen.blit(score_board,(0,0))
pg.display.update()
```

```
def start_game():
    def create_food():
        small_food_rect.center = random.choice(tuple(empty_region))
        screen.blit(small_food,small_food_rect)
        return (x,y)
    def replay_routine():
        nonlocal highscore_value,score_value,newpos,currentpos,snake,start_flag,extend_flag,x,y,current_event
        screen.blit(screen_bg,screen_bg_rect)
        highscore_set_routine()
        score_reset_routine()
        for i in snake:
            addto_empty_region(i)
        currentpos = initpos
        newpos = currentpos
        snake = [currentpos]
        create_food()
        snake_head_rect.center = initpos
        pg.draw.circle(screen,blue,snake_head_rect.center,snake_head_rect.width/2)
        start_flag = False
        extend_flag = True
        current_event=None
        x=-1
        y=-1
```

```
        def highscore_set_routine():
            nonlocal highscore_value,score_value,highscore_rect
            if highscore_value>score_value:
                return
            highscore_value = score_value
            text = create_text(str(highscore_value),white)
            text_rect = text.get_rect()
            text_rect.topleft = highscore_rect.topleft
            highscore_rect = text_rect.copy()
            replacewith_subimage(score_board,score_board_bg,score_board_bg_rect,highscore_rect)
            score_board.blit(text,highscore_rect)
            screen.blit(score_board,(0,0))
        def score_reset_routine():
            nonlocal score_value,score_rect
            replacewith_subimage(score_board,score_board_bg,score_board_bg_rect,score_rect)
            score_value = 0
            text = create_text(str(score_value),white)
            text_rect = text.get_rect()
            text_rect.topleft = score_rect.topleft
            score_rect = text_rect.copy()
            score_board.blit(text,score_rect)
            screen.blit(score_board,(0,0))
```

```
        def score_increment_routine():
            nonlocal score_value,score_rect,highscore_value
            score_value+=1
            text = create_text(str(score_value),white)
            text_rect = text.get_rect()
            text_rect.topleft = score_rect.topleft
            score_rect = text_rect.copy()
            replacewith_subimage(score_board,score_board_bg,score_board_bg_rect,score_rect)
            score_board.blit(text,score_rect)
            screen.blit(score_board,(0,0))
            if highscore_value<score_value:
                highscore_set_routine()
        def game_over_routine():
            nonlocal running
            pg.display.update()
            pg.time.delay(500)
            game_over_rect.center = ((gamespace[2]+gamespace[0])/2,(gamespace[3]+gamespace[1])/2)
            screen.blit(game_over,game_over_rect)
            pg.display.update()
            pg.time.delay(500)
            run = True
            while run:
                for event in pg.event.get():
                    if event.type==pg.QUIT:
                        running = False
                        return
                    elif event.type == pg.KEYDOWN:
                        if event.key == pg.K_SPACE:
                            replay_routine()
                            return
                pg.display.update()
```

```
        def is_colliding(pos):
            if len(snake)<4:
                return False
            for i in snake:
                if pos==i:
                    return True
            return False
        def replacewith_subimage(surface,image,image_rect,rect):
            surface.blit(image,rect,(rect.left-image_rect.left,rect.top-image_rect.top)+(rect.width,rect.height))
```

```
def resolve_event(event):
    nonlocal x,y
    if event==None:
        return
    if event.key == pg.K_RIGHT:
        if x==0 and y == 1:
            return
        x = 1
        y = 0
    elif event.key == pg.K_UP:
        if x==0 and y == 0:
            return
        x = 1
        y = 1
    elif event.key == pg.K_LEFT:
        if x==1 and y == 0:
            return
        x = 0
        y = 1
    elif event.key == pg.K_DOWN:
        if x==1 and y == 1:
            return
        x = 0
        y = 0
    else :
        pass
```

```
def initialize_empty_region():
    for i in range(gamespace[0]+int(step/2),gamespace[2]-int(step/2),step):
        for j in range(gamespace[1]+int(step/2),gamespace[3]-int(step/2),step):
            empty_region.add((i,j))
    empty_region.discard(newpos)
def addto_empty_region(pos):
    empty_region.add(pos)
def removefrom_empty_region(pos):
    empty_region.discard(pos)
step = 20
no_of_minimov = 10
framerate = 60
empty_region = set({})
x = -1
y = -1
Headx=0
Heady=0
Tailx=0
Taily=0
```

```
initpos = (gamespace[0]+step/2+(15-1)*step,gamespace[1]+step/2+(15-1)*step)
currentpos = initpos
snake_head_rect = pg.Rect(currentpos[0],currentpos[1],step,step)
snake_head_rect.center = currentpos
snake_tail_rect = snake_head_rect.copy()
snake = [currentpos]
newpos = currentpos
initialize_empty_region()
foodpos = create_food()
pg.draw.circle(screen,blue,snake_head_rect.center,snake_head_rect.width/2)
running = True
start_flag = False
position_flag = True
extend_flag = True
current_event=None
initialize_empty_region()
clock = pg.time.Clock()
while running:
    clock.tick(framerate)
    if position_flag and start_flag:
        resolve_event(current_event)
        currentpos = newpos
        if x==0 and y==0:
            newpos = (currentpos[0],currentpos[1]+step)
        elif x==1 and y==0:
            newpos = (currentpos[0]+step,currentpos[1])
        elif x==0 and y==1:
            newpos = (currentpos[0]-step,currentpos[1])
```

```
            elif x==1 and y==1:
                newpos = (currentpos[0],currentpos[1]-step)
            else:
                pass
            if newpos[0]>gamespace[2] or newpos[0]<gamespace[0] or newpos[1]<gamespace[1] or newpos[1]>gamespace[3]:
                game_over_routine()
                continue
            elif is_colliding(newpos):
                game_over_routine()
                continue
            else:
                pass
            snake.insert(0,newpos)
            removefrom_empty_region(newpos)
            Headx = (newpos[0]-currentpos[0])/no_of_minimov
            Heady = (newpos[1]-currentpos[1])/no_of_minimov
            Tailx = (snake[-2][0]-snake[-1][0])/no_of_minimov
            Taily = (snake[-2][1]-snake[-1][1])/no_of_minimov
            snake_tail_rect.center = snake[-1]
            position_flag = False
```

```
        if start_flag:
            if not extend_flag:
                temp_rect = snake_tail_rect.copy()
                temp_rect.center = snake[-1]
                replacewith_subimage(screen,screen_bg,screen_bg_rect,temp_rect)
                snake_tail_rect.move_ip(Tailx,Taily)
                pg.draw.rect(screen,blue,snake_tail_rect)
            snake_head_rect.move_ip(Headx,Heady)
            pg.draw.circle(screen,blue,snake_head_rect.center,step/2)
            if snake_head_rect.center == newpos:
                position_flag = True
                if not extend_flag:
                    addto_empty_region(snake.pop())
                if snake_head_rect.center == small_food_rect.center:
                    create_food()
                    score_increment_routine()
                    extend_flag = True
                elif extend_flag:
                    extend_flag = False
                else:
                    pass
```

```
        for event in pg.event.get():
            if event.type == pg.QUIT:
                running = False
                pass
            elif event.type == pg.KEYDOWN:
                start_flag = True
                current_event = event
            else :
                pass
        pg.display.update()
    pg.quit()
start_game()
snake_highscore = highscore_value
```

Flappy Bird

```
def flappy_bird():
    global flappy_bird_highscore
    def create_text(text,fg,bg=None,font = 'calibri',size=20):
        Font = pg.font.SysFont(font,size,bold = True)
        Text = Font.render(text,True,fg,bg)
        return Text
    def convert_to_surface(image):
        mode = image.mode
        size = image.size
        data = image.tobytes()
        return pg.image.fromstring(data,size,mode)
    pg.init()
    screen = pg.display.set_mode((600,500))
    pg.display.set_caption('Flappy Bird')
    gamespace = [0,100,600,500]

    white = (255,255,255)

    score_board = pg.Surface((600,100))
    score_board_bg = pg.image.load('score_board_bg.png')
    score_board_bg_rect = score_board_bg.get_rect()
    score_board.blit(score_board_bg,(0,0))
    score_board_rect = score_board.get_rect()

    highscore_value = flappy_bird_highscore
    score_value = 0
```

```
    pipes = (pipeup,pipedown)
    pipe_rects = (pipeup_rect,pipedown_rect)
    pipe_initial_position = gamespace[2]

    flappy_bg = pg.image.load('flappy_bg.png')
    flappy_bg_rect = flappy_bg.get_rect()
    flappy_bg_rect.topleft = (gamespace[0],gamespace[1])

    game_over = pg.image.load('game_over.png')

    score_board.blit(highscore_text,highscore_rect)
    score_board.blit(score_text,score_rect)

    score_board.blit(trophy,trophy_rect)
    score_board.blit(small_pipe,small_pipe_rect)

    screen.blit(score_board,(0,0))

    screen.blit(flappy_bg,gamespace[0:2])

    screen.blit(flappy_bird,flappy_bird_rect)
```

```
    flappy_bird= pg.image.load('flappy_bird.png')
    flappy_bird_rect = flappy_bird.get_rect()
    flappy_bird_initial_position = ((gamespace[0]+gamespace[2])/3,(gamespace[1]+gamespace[3])/2)
    flappy_bird_rect.center = flappy_bird_initial_position

    trophy = Image.open('trophy.png')
    trophy = trophy.resize((50,50),Image.ANTIALIAS)
    trophy = convert_to_surface(trophy)
    trophy_rect = trophy.get_rect()
    trophy_rect.center = (score_board_rect.width/3,score_board_rect.height*2/5)
    small_pipe = pg.image.load('small_pipe.png')
    small_pipe_rect = small_pipe.get_rect()
    small_pipe_rect.center = (score_board_rect.width*2/3,score_board_rect.height*2/5)

    highscore_text = create_text(str(highscore_value),white)
    highscore_rect = highscore_text.get_rect()
    highscore_rect.left = trophy_rect.right+20
    highscore_rect.centery = trophy_rect.centery
    score_text = create_text(str(score_value),white)
    score_rect = score_text.get_rect()
    score_rect.left = small_pipe_rect.right+20
    score_rect.centery = small_pipe_rect.centery

    pipeup = pg.image.load('pipe.png')
    pipeup_rect = pipeup.get_rect()
    pipeup_rect.right = gamespace[2]
    pipeup_rect.bottom = gamespace[3]
```

```
def start_game():
    def highscore_set_routine():
        nonlocal highscore_value,score_value,highscore_rect
        if highscore_value<score_value:
            highscore_value = score_value
        else:
            return
        text = create_text(str(highscore_value),white)
        text_rect = text.get_rect()
        text_rect.topleft = highscore_rect.topleft
        highscore_rect = text_rect.copy()
        replacewith_subimage(score_board,score_board_bg,score_board_bg_rect,highscore_rect)
        score_board.blit(text,highscore_rect)
        screen.blit(score_board,(0,0))
    def score_reset_routine():
        nonlocal score_value,score_rect
        replacewith_subimage(score_board,score_board_bg,score_board_bg_rect,score_rect)
        score_value=0
        text = create_text(str(score_value),white)
        text_rect = text.get_rect()
        text_rect.topleft = score_rect.topleft
        score_rect = text_rect.copy()
        score_board.blit(text,score_rect)
        screen.blit(score_board,(0,0))
```

```
    def score_increment_routine():
        nonlocal score_value,score_rect
        score_value+=1
        text = create_text(str(score_value),white)
        text_rect = text.get_rect()
        text_rect.topleft = score_rect.topleft
        score_rect = text_rect.copy()
        replacewith_subimage(score_board,score_board_bg,score_board_bg_rect,score_rect)
        score_board.blit(text,score_rect)
        screen.blit(score_board,(0,0))
        highscore_set_routine()
    def replay_routine():
        nonlocal pipe_list,void_list,pipe_Dx,start_flag,current_pointer
        start_flag = False
        current_pointer = 0
        screen.blit(flappy_bg,gamespace[:2])
        flappy_bird_rect.center = flappy_bird_initial_position
        pipe_list = []
        void_list = []
        pipe_Dx = pipe_separation
        score_reset_routine()
```

```
    def game_over_routine():
        nonlocal running
        pg.display.update()
        pg.time.delay(500)
        game_over_rect = game_over.get_rect()
        game_over_rect.center = ((gamespace[2]+gamespace[0])/2,(gamespace[3]+gamespace[1])/2)
        screen.blit(game_over,game_over_rect)
        pg.display.update()
        pg.time.delay(500)
        pg.event.get()
        run = True
        while run:
            for event in pg.event.get():
                if event.type==pg.QUIT:
                    running = False
                    return
                elif event.type == pg.KEYDOWN:
                    if event.key == pg.K_SPACE:
                        replay_routine()
                        return
            pg.display.update()
    def generate_void(Range,height):
        number = random.randrange(Range[0],Range[1])
        return (number,number+height)
    def generate_pipes(void_range,pipes,pipe_rects):
        upimage = pipes[0].subsurface((0,0,pipe_rects[0].width,gamespace[3]-void_range[1]))
        downimage = pipes[1].subsurface((0,pipe_rects[1].height-(void_range[0]-gamespace[1]),
                                         pipe_rects[1].width,void_range[0]-gamespace[1]))
        return (upimage,downimage)
```

```
    def replacewith_subimage(surface,image,image_rect,rect):
        surface.blit(image,rect,(rect.left-image_rect.left,rect.top-image_rect.top)+(rect.width,rect.height))
    def set_pipe_separation():
        nonlocal pipe_separation
        pipe_separation =  random.randrange(pipe_separation_range[0],pipe_separation_range[1])
```

```
        frame_rate = 110
        clock = pg.time.Clock()
        flappy_bird_Vy=0
        fb_boundary_apx = ((flappy_bird_rect.height/100)*10,(flappy_bird_rect.width/100)*40)
        pipe_Vx = 230
        jump_height = 0.35
        g = 9.80665
        pipe_list=[]
        void_list = []
        pipe_separation_range = (200,500)
        current_pointer = 0
        pipe_separation = 300
        void_separation = 100
        pipe_Dx = pipe_separation
        start_flag = False
        running = True
```

```
        while running:
            clock.tick(frame_rate)
            if start_flag:
                flappy_bird_change = (flappy_bird_Vy*(1/frame_rate)+g*((1/frame_rate)**2)/2)*200
                pipe_change = -(pipe_Vx*(1/frame_rate))
                pipe_Dx+=abs(pipe_change)
                flappy_bird_Vy += g*(1/frame_rate)
                if pipe_Dx>=pipe_separation:
                    void_list.append(generate_void((gamespace[1]+50,gamespace[3]-(void_separation+50)),void_separation))
                    pipe_list.append(generate_pipes(void_list[-1],pipes,pipe_rects))
                    rect1 = pipe_list[-1][0].get_rect()
                    rect2 = pipe_list[-1][1].get_rect()
                    rect1.bottom = gamespace[3]
                    rect1.left = pipe_initial_position
                    rect2.top = gamespace[1]
                    rect2.left = pipe_initial_position
                    pipe_list[-1]+=(rect1,rect2)
                    pipe_Dx = 0
                    set_pipe_separation()
                if pipe_list[0][3].right<gamespace[0]:
                    pipe_list.pop(0)
                    void_list.pop(0)
                    current_pointer-=1
```

```
                if flappy_bird_rect.top+flappy_bird_change<=gamespace[1]:
                    flappy_bird_Vy = 0
                else:
                    replacewith_subimage(screen,flappy_bg,flappy_bg_rect,flappy_bird_rect)
                    for i in pipe_list:
                        replacewith_subimage(screen,flappy_bg,flappy_bg_rect,i[2])
                        replacewith_subimage(screen,flappy_bg,flappy_bg_rect,i[3])
                    flappy_bird_rect.move_ip(0,flappy_bird_change)
                    for i in pipe_list:
                        i[2].move_ip(pipe_change,0)
                        i[3].move_ip(pipe_change,0)
            for i in pipe_list:
                screen.blit(i[0],i[2])
                screen.blit(i[1],i[3])
            screen.blit(flappy_bird,flappy_bird_rect)

            if start_flag and pipe_list[current_pointer][2].left<flappy_bird_rect.right-fb_boundary_apx[1]:
                uppertouch = flappy_bird_rect.top+fb_boundary_apx[0]<void_list[current_pointer][0]
                lowertouch = flappy_bird_rect.bottom-fb_boundary_apx[0]>void_list[current_pointer][1]
                if uppertouch or lowertouch:
                    game_over_routine()
```

```
            if start_flag and pipe_list[current_pointer][2].right<flappy_bird_rect.left+fb_boundary_apx[1]:
                current_pointer+=1
                score_increment_routine()
            if start_flag and flappy_bird_rect.top>=gamespace[3]:
                game_over_routine()
            for event in pg.event.get():
                if event.type == pg.QUIT:
                    running = False
                elif event.type == pg.KEYDOWN:
                    if event.key == pg.K_SPACE:
                        start_flag = True
                        flappy_bird_Vy = -(2*g*jump_height)**(1/2)
                else:
                    pass
            pg.display.flip()
        pg.quit()
    start_game()
    flappy_bird_highscore = highscore_value
```

Tic Tac Toe

```
def tictactoe():
    def create_button_rects(button_rects):
        for i in range(9):
            rect = pg.Rect(0,0,button_size,button_size)
            rect.center = (int((i%3)*button_size+button_size/2),int((i//3)*button_size+button_size/2))
            button_rects.append(rect.copy())
    def draw_canvas():
        pg.draw.rect(screen,button_bg,pg.Rect(0,0,width,3*button_size))
        pg.draw.line(screen,black,(button_size,0),(button_size,3*button_size),line_width)
        pg.draw.line(screen,black,(2*button_size,0),(2*button_size,3*button_size),line_width)
        pg.draw.line(screen,black,(0,button_size),(width,button_size),line_width)
        pg.draw.line(screen,black,(0,height/2),(width,height/2),line_width)
        pg.draw.rect(screen,black,pg.Rect(0,3*button_size,3*button_size,3))
    def initialize_image_list():
        nonlocal image_list
        for i in range(9):
            image_list[i]=None
    def get_button_index(mousepos):
        return mousepos[0]//button_size+(mousepos[1]//button_size)*3
    def get_image_rect(rect,border):
        temp = rect.copy()
        temp.width-=border
        temp.height-=border
        temp.center = rect.center
        return temp
```

```
    def button_click(index,turn):
        if turn ==0:
            image_list[index]=X
            screen.blit(X,get_image_rect(button_rects[index],10))
        elif turn == 1:
            image_list[index]=O
            screen.blit(O,get_image_rect(button_rects[index],10))
    def convert_to_surface(image):
        mode = image.mode
        size = image.size
        data = image.tobytes()
        return pg.image.fromstring(data,size,mode)
    def iswon(position):
        for i in win[position]:
            result = True
            for j in i:
                result = result and image_list[j-1]==image_list[position-1]
            if result:
                return i
        return None
    def reset_routine():
        nonlocal turn
        draw_canvas()
        initialize_image_list()
        turn = 0
    def in_rect(rect,position):
        return position[0]>rect.left and position[0]<rect.right and position[1]>rect.top and position[1]<rect.bottom
```

```
def change_pointer(rect):
    if in_rect(rect,pg.mouse.get_pos()):
        pg.mouse.set_cursor(pg.SYSTEM_CURSOR_HAND)
    else:
        pg.mouse.set_cursor(pg.SYSTEM_CURSOR_ARROW)

def strike(buttons,turn):
    color = None
    if turn == 0:
        color = (240,0,0)
    if turn == 1:
        color = (0,128,255)
    start = None
    end = None
    length = None
    if buttons[0]==1 and buttons[1]==5 or buttons[0]==3 and buttons[1]==5:
        if buttons[0]==1:
            start = button_rects[buttons[0]-1].topleft
            end = button_rects[buttons[-1]-1].bottomright
        else:
            start = button_rects[buttons[0]-1].topright
            end = button_rects[buttons[-1]-1].bottomleft
        length = width*2**(1/2)
    else:
        if buttons[0] in (1,2,3):
            if buttons[0]==1 and buttons[1]==2:
                start = button_rects[buttons[0]-1]
                start = (start.left,start.centery)
                end = button_rects[buttons[-1]-1]
                end = (end.right,end.centery)
```

```
            else:
                start = button_rects[buttons[0]-1]
                start = (start.centerx,start.top)
                end = button_rects[buttons[-1]-1]
                end = (end.centerx,end.bottom)
        else:
            start = button_rects[buttons[0]-1]
            start = (start.left,start.centery)
            end = button_rects[buttons[-1]-1]
            end = (end.right,end.centery)
        length = width

    pg.init()
    framerate = 80
    inc = length/framerate
    line_width = 4
    widthinc = 0
    heightinc = 0
    count = 0
    line_rect = pg.Rect(0,0,10,length)
    clock = pg.time.Clock()
    running = True
```

```
    while running:
        clock.tick(framerate)
        mouse = pg.mouse.get_pos()
        if count <framerate:
            if start[0]!=end[0]:
                if start[0]>end[0]:
                    widthinc-=inc
                else:
                    widthinc+=inc
            if start[1]!=end[1]:
                if start[1]>end[1]:
                    heightinc-=inc
                else:
                    heightinc+=inc
            pg.draw.line(screen,color,start,[start[0]+widthinc,start[1]+heightinc],5)
            count+=1
        change_pointer(reload_rect)
        for event in pg.event.get():
            if event.type == pg.QUIT:
                return True
            if event.type == pg.MOUSEBUTTONDOWN:
                if in_rect(reload_rect,pg.mouse.get_pos()):
                    reset_routine()
                    return
        pg.display.flip()

win_list = [[1,2,3],[4,5,6],[7,8,9],[1,4,7],[2,5,8],[3,6,9],[1,5,9],[3,5,7]]
win = defaultdict(lambda:[])
for i in win_list:
    for j in i:
        win[j].append(i)
```

```
width = 300
height = 400
screen = pg.display.set_mode((width,height))
pg.display.set_caption('Tic Tac Toe')
screen.fill((230,230,230))
black = (0,0,0)
button_bg = (240,240,240)
button_hover_bg = (245,245,245)
button_size = 100
line_width = 1
button_rects = []
create_button_rects(button_rects)
active_button = -1
recent_active_button = -1
X = Image.open('Tic Tac Toe images\X.png')
X = X.resize((button_size-10,button_size-10),Image.ANTIALIAS)
X = convert_to_surface(X)
O = Image.open('Tic Tac Toe images\O.png')
O = O.resize((button_size-10,button_size-10),Image.ANTIALIAS)
O = convert_to_surface(O)
reload = Image.open('Tic Tac Toe images\\reload.png')
reload = reload.resize((button_size//2,button_size//2),Image.ANTIALIAS)
reload = convert_to_surface(reload)
reload_rect = reload.get_rect()
reload_rect.center = (width/2,3*button_size+button_size/2)
screen.blit(reload,reload_rect)
image_list = list(range(9))
initialize_image_list()
draw_canvas()
turn = 0
running = True
```

```
while running:
    mousepos = pg.mouse.get_pos()
    active_button = get_button_index(mousepos)
    if not recent_active_button == active_button and active_button<9:
        pg.draw.rect(screen,button_hover_bg,get_image_rect(button_rects[active_button],line_width))
        if image_list[active_button]!=None:
            screen.blit(image_list[active_button],get_image_rect(button_rects[active_button],10))
    if not recent_active_button == active_button and recent_active_button < 9 and recent_active_button>-1:
        pg.draw.rect(screen,button_bg,get_image_rect(button_rects[recent_active_button],line_width))
        if image_list[recent_active_button]!=None:
            screen.blit(image_list[recent_active_button],get_image_rect(button_rects[recent_active_button],10))
    if active_button<9:
        recent_active_button = active_button
    else :
        recent_active_button = -1
    for event in pg.event.get():
        if event.type == pg.QUIT:
            running = False
        elif event.type == pg.MOUSEBUTTONDOWN:
            if active_button<9 and image_list[active_button]==None:
                button_click(active_button,turn)
                buttons = iswon(active_button+1)
                if buttons!=None:
                    if strike(buttons,turn):
                        running = False
                    turn = 0
                    continue
                turn= turn^1
            elif in_rect(reload_rect,pg.mouse.get_pos()):
                reset_routine()
    change_pointer(reload_rect)
    pg.display.update()
pg.quit()
```

THE END