DESIGN DOCUMENTATION

1. Design

1a. Overview

The program is an interactive snake game. The game is composed of 3 objects: a snake, a monster and food items represented by a set of numbers from 1 to 9. At the beginning of the game is the initial interface. The user can see a short introduction on this screen. Then, the user can click the screen to start the game. What's more, The user is allowed to move the snake with the keyboard using the four directional bottoms (up, down, right and left). Since that, the timer will start timing and the monster will start moving towards the snake's head. The goal of the game is to maneuver the snake within the game area to consume all the food items while avoiding head-on collision with the monster.As each food item is consumed, the snake grows with its body lengthened in size equal to the value of the number being passed.

1b. Data Model

1. snake including the tail

I use the turtle model to draw the head of the snake and its tail. I also use two lists **snake_x** and **snake_y** to store the coordinates of the snake including the tail (It means the coordinates of all the squares of the snake). What's more, I use a integer to represent the number of the tails which are being extended.

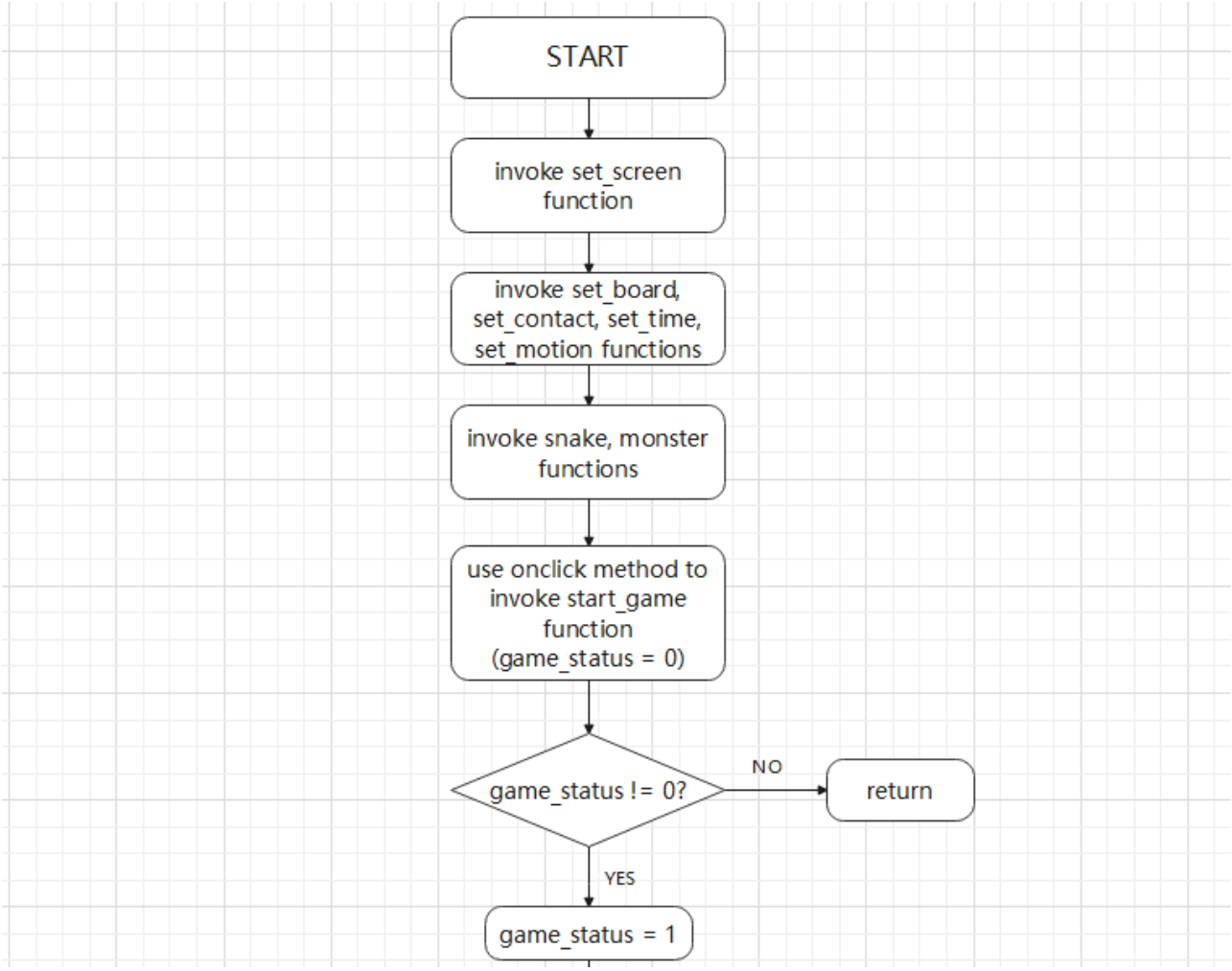
2. food items

I use the turtle model to draw the food items. And then, I use a list called **food_items** to store all the turtles which draw the food items. I also use a list called **eaten** to store the information about whether the food items have been eaten. (The values in this list are Boolean).

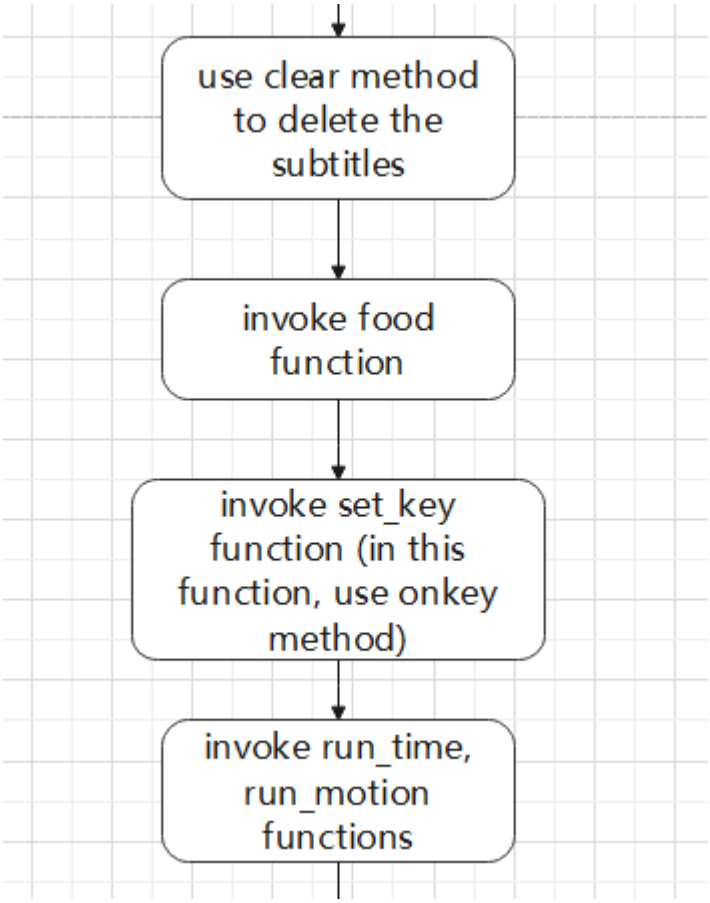
1c. Program Structure

> Reminder: The text is a brief description, the program structure diagram is more detailed and shows the main structure of the program.

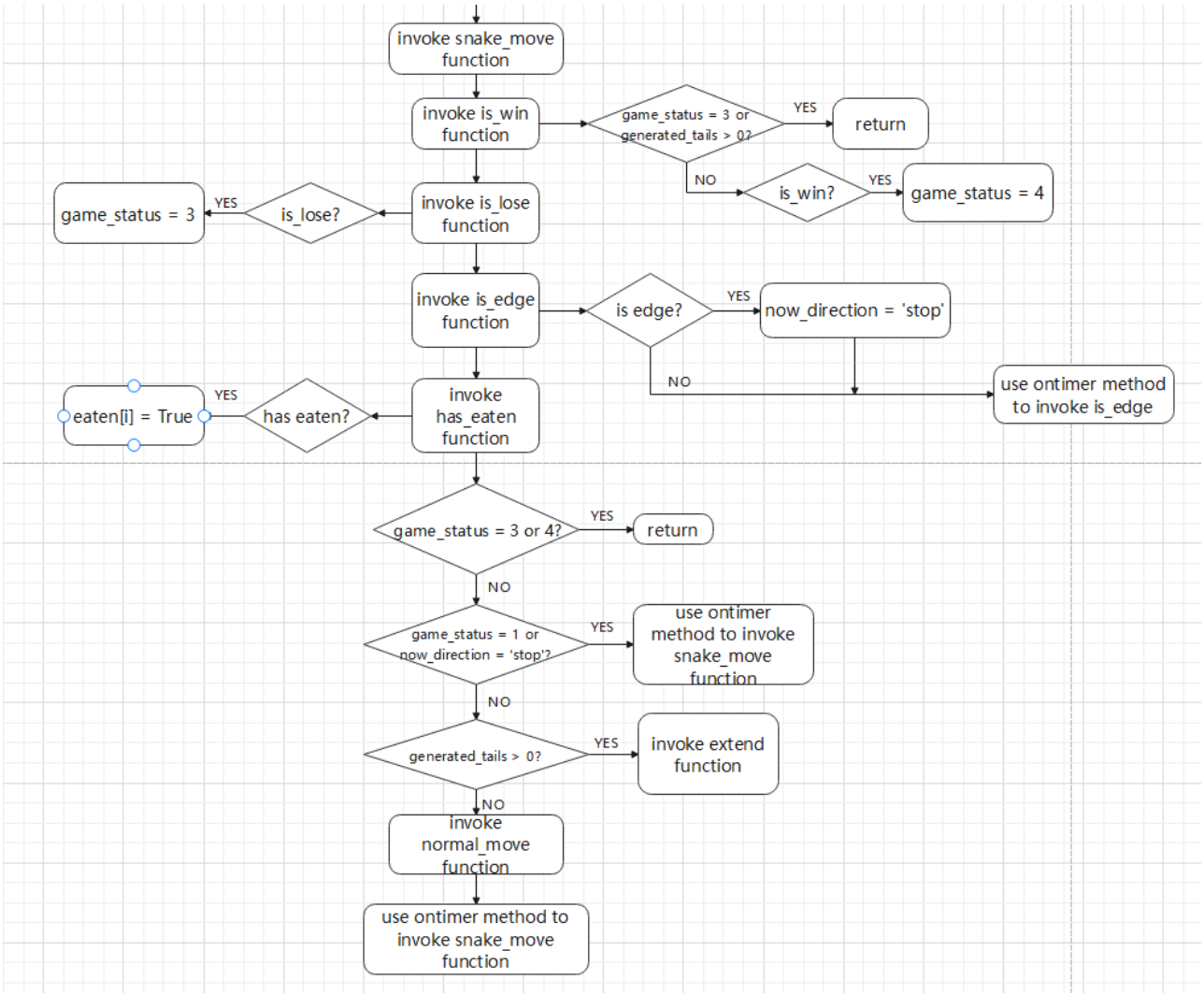
- The first part is to set the initial interface. The following diagram shows the structure of this part.



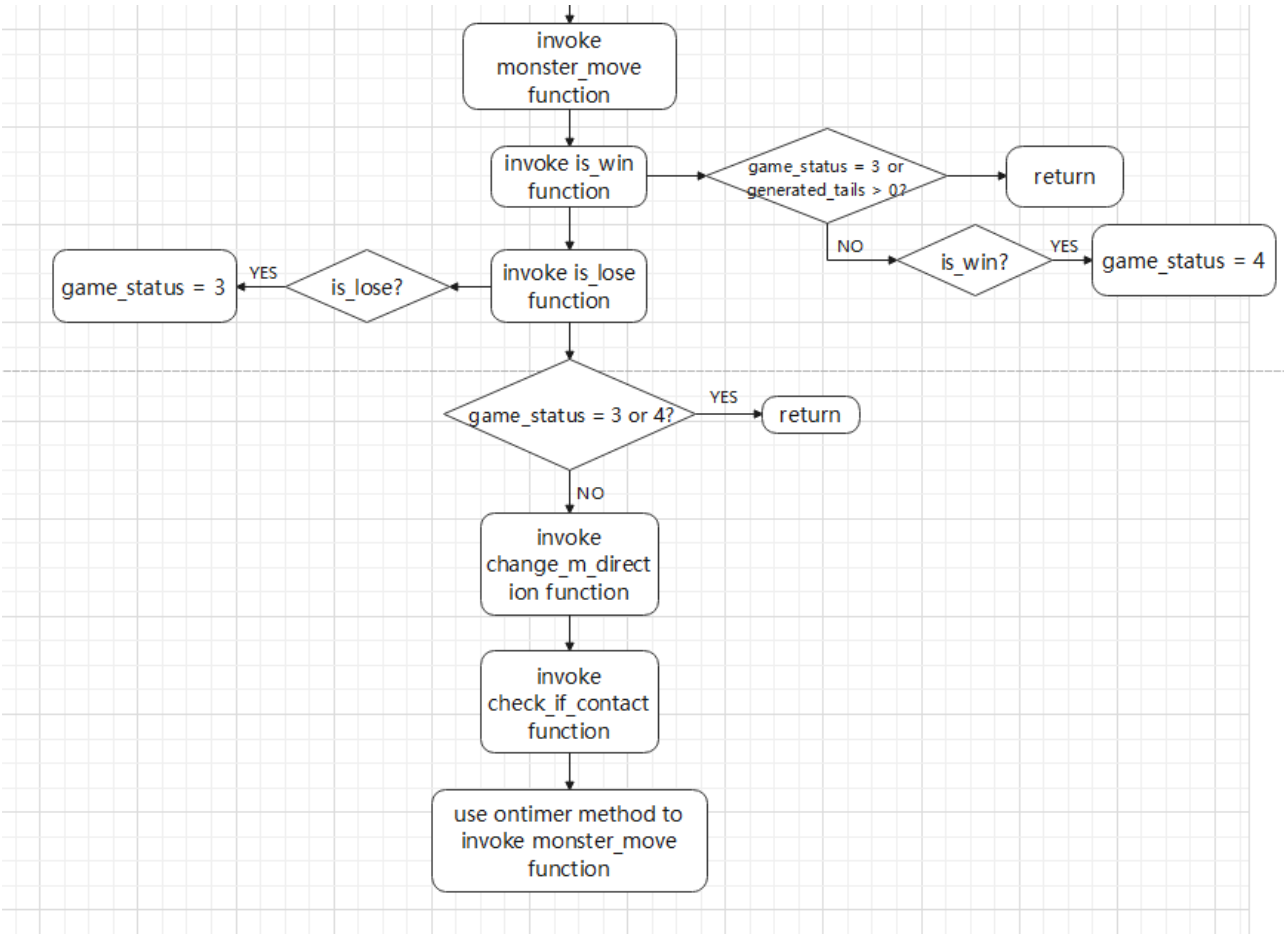
- Then, the user can click the screen to start the game. The structure diagram of this part:



- What's more, the user can maneuver the snake. The structure diagram of this part ;



- Last but not least, after invoke the snake_move function, the structure of code which implement the movement of the monster is like this:



Description: (The bold parts are functions and the italicized parts are variables)
In this program, we first invoke **set_screen** function to set the initial interface of the snake game. In this function, we create a variable *s* which represent the main screen. Then we create another variable *subtitles* to represent the subtitles of the game. And then, we invoke **set_board**, **set_contact**, **set_time** and **set_motion** functions to build the gaming board of the game.

Then, we invoke **snake** and **monster** functions to create two turtle objects to draw the snake and monster.

What's more, I use **onclick** method to allow the user to click the screen to start the game, the **start_game** function will be invoked at this time.

In start_game function, **food**, **set_key**, **run_time**, **run_motion** functions will be invoked first to set food items, keys, update time, motion status at the top of the board respectively.

And then, we invoke the **snake_move** function. In this function, we first invoke **is_win**, **is_lose** function to check whether we win or not. Then we invoke **is_edge** function to check whether the snake touch the edge of the board or not. And we also invoke **has_eaten** function to check whether a food item has been eaten or not. Then, if *game_status* = 1 or *now_direction* = 'stop', we use **ontimer** function to invoke **snake_move** again. Else, we invoke **extend** or **normal_move** functions depends on the value of *generated_tails*. Last, We use **ontimer** to invoke **snake_move** function.

Last, we invoke the **monster_move** function. We also need to check whether we have win the game or not first. If not, we invoke **change_m_direction** function to implement the change of the direction of the monster. Then we use **stamp**, **setheading**, **forward** and **clearstamps** methods to implement the movement of the monster. In the end, we use **ontimer** to invoke **moster_move** again and again.

1d. Processing Logic

i. The logic used to motion the snake and monster.

To move the snake and monster, I mainly use two methods of turtle model. The first is turtle.stamp(), and the other one is turtle.clearstamps(1). The stamp method can copy the turtle at the same position, and then I move the head of the snake (or move the monster) once. Then, I use the clearstamps to clear the earliest of the stamps I have created before. For the snake, it is the end of its tail, for monster, It is the last position of it. And I use ontimer method to repeat the process again and again. Therefore, we can implement the movement of the snake and monster.

For the monster, what's more, is the algorithm to move toward the snake's head. I compare the x distance and y distance between the head of the snake and the monster. Then, if the speed of the monster is larger than that of the snake, it will go through the longer side to shorten the distance, otherwise it will go through the shorter side.

ii. The logic used to expand the snake tail

I use a variable *generated_tails* to store how many squares I need to add (The default value is 5). First, I still use the method stamp to implement the expand of the tail. Then, I move the head of the snake. I also subtract 1 for generated_tails so that the squares will be added until the generated_tails is 0 (it means that the tail is not expanding right now). I put this function in the function **snake_move** and use ontimer method to invoke **snake_move** again and again. Therefore, if generated_tails > 0, the **expand** function will be invoked again and again.

iii. the logic used to detect body contact between the snake and the monster

I define a function **check_if_contact** to detect the body contact between the snake and the monster. When I expand the tail or just normally move the snake, I put the coordinates of the body of the snake into two list *snake_x* and *snake_y*. So I can check for all coordinates in these two list, whether the x or y distance between monster and the body of the snake is larger than 20 or not. If not, the variable *contact* which store the number of contact will add 1. Therefore, I can detect the body contact.

2. Function Specifications

- set_screen

```
def set_screen():
    # Set the size and position of the main screen
    s = turtle.Screen()
    s.setup(660,740)
    s.tracer(0)

    # Set the title of the main screen
    s.title("Snake by Edgar")

    # Set subtitles at the start of the game
    subtitles = turtle.Turtle()
    subtitles.up()
    subtitles.ht()
    subtitles.goto(-230,60)
    subtitles.write(
        '''
        Welcome to Edgar's version of snake...

        You are going to use the 4 arrow keys to move the snake
        around the screen, trying to consume all the food items
        before the monster catches you...

        Click anywhere on the screen to start the game, have fun!
        ''',
        font = ("Times New Roman", 11, "normal")
    )

    set_board()
    set_contact()
    set_time()
    set_motion()

    s.update()

    return s, subtitles
```

The function is used to set the initial interface. The variable s is a Screen object. It is used to set up the main screen of the game. The variable subtitles is a Turtle object which is used to write the short introduction of the game (The subtitles). The return results of the function is s and subtitles.

- set_board

```
def set_board():
    board = turtle.Turtle()
    board.speed(0)
    board.up()
    board.ht()
    board.goto(-250,290)
    board.down()
    for i in range(2):
        board.forward(500)
        board.right(90)
        board.forward(580)
        board.right(90)
    board.up()
    board.goto(-250,210)
    board.down()
    board.forward(500)
    return board
```

The function is used to draw the board (The game area). The variable board is a Turtle object. The return result is board.

- set_contact

```
contact = 0
contact_pen = turtle.Turtle()
def set_contact():
    contact_pen.up()
    contact_pen.ht()
    contact_pen.goto(-200,240)
    contact_pen.write("Contact: %d"%contact, font=("Times New Roman", 14, "bold"))
```

The function is used to write the contact value at the top of the screen. The variable contact is the value (default is 0). The variable contact_pen is a Turtle object.

- set_time

```
time = 0
time_pen = turtle.Turtle()
def set_time():
    time_pen.up()
    time_pen.ht()
    time_pen.goto(-60,240)
    time_pen.write("Time: %ds"%time, font=("Times New Roman", 14, "bold"))
```

The function is used to write the time value at the top of the screen. The variable time is the value (default is 0). The variable time_pen is a Turtle object.

- set_motion

```
motion = "Paused"
motion_pen = turtle.Turtle()
def set_motion():
    motion_pen.up()
    motion_pen.ht()
    motion_pen.goto(80,240)
    motion_pen.write("Motion: %s"%motion, font=("Times New Roman", 14, "bold"))
```

Similar to set_contact and set_time, motion is the value (default is "Paused"). motion_pen is a Turtle object.

- snake

```
def snake():
    snake = turtle.Turtle()
    snake.up()
    snake.goto(0,-40)
    snake.shape("square")
    snake.color("red","red")
    return snake
```

The function is used to draw the initial snake. The variable snake is a Turtle object.

- monster

```
def monster():
    monster = turtle.Turtle()
    monster.up()
    x = random.randint(-48,48)
    while abs(x) <= 16:
        x = random.randint(-48,48)
    y = random.randint(-48,10)
    while abs(y+5) <= 16:
        y = random.randint(-48,10)
    monster.goto(x*5,y*5)
    monster.shape("square")
    monster.color("purple","purple")
    return monster
```

The function is used to draw the initial monster. The variable monster is a Turtle object. The monster have a distance between the initial snake.

- start_game

```
game_status = 0
def start_game(x,y):
    global game_status
    if game_status != 0:
        return
    game_status = 1
    subtitles.clear()
    food()
    set_key()

    fx_screen.update()
    run_time()
    run_motion()

    snake_move()
    monster_move()
```

The function is the core logic structure of the program. In this function, we can achieve the playing function of the game. The variable game_status is a important status I use to represent the gaming status.

- check_food

```
def check_food(x,y):
    for food in food_items:
        if abs(food.xcor()-x) <= 20 and abs(food.ycor()-y) <= 20:
            return False
    return True
```

The function is used to check whether any food near (x,y). This can keep food from getting too close.

- food

```
food_items = []
def food():
    for i in range(1,10):
        food = turtle.Turtle()
        food.speed(0)
        food.up()
        food.ht()
        x = random.randint(-11,11)
        y = random.randint(-13,9)
        while not check_food(x*20,y*20):
            x = random.randint(-11,11)
            y = random.randint(-13,9)
        food.goto(x*20, y*20-5)
        food.write(i, font=("Times New Roman", 10, "normal"))
        food_items.append(food)
```

This function is used to generate all the food items. The variable food is a Turtle object. The variable food_items is a list which store all the food turtles.

- run_time

```
def run_time():
    global time
    if game_status == 1:
        fx_screen.ontimer(run_time, 0)
    elif game_status == 3 or game_status == 4:
        return
    else:
        time_pen.clear()
        set_time()
        time += 1
        fx_screen.ontimer(run_time, 1000)
        fx_screen.update()
```

This function is used as a timer. It can update the gaming time.

- run_motion

```
def run_motion():
    global motion
    if game_status == 1:
        fx_screen.ontimer(run_motion, 0)
    else:
        motion = now_direction.capitalize()
        if now_direction == 'stop':
            motion = 'Paused'
        motion_pen.clear()
        set_motion()
        if game_status == 3 or game_status == 4:
            return
        else:
            fx_screen.ontimer(run_motion, speed//2)
            fx_screen.update()
```

This function is used to update the motion status of the snake and rewrite it at the top of the screen.

- set_key

```
def set_key():
    fx_screen.onkey(stop, "space")
    fx_screen.onkey(right, "Right")
    fx_screen.onkey(left, "Left")
    fx_screen.onkey(up, "Up")
    fx_screen.onkey(down, "Down")
    fx_screen.listen()
```

This function is used to set the four directional keys. The variable fx_screen is the main screen.

- is_win

```
def is_win():
    global game_status
    if game_status == 3 or generated_tails:
        return

    is_win = True
    for i in range(9):
        if eaten[i] == False:
            is_win = False

    if is_win:
        win = turtle.Turtle()
        win.up()
        win.ht()
        win.goto(snake.xcor(), snake.ycor())
        win.color("red")
        win.write("winner!!", font=("Times New Roman", 14, "normal"))
        game_status = 4
```

This function is used to check whether we win or not. The variable is_win is boolean. win is a Turtle object which can write the information if we win.

- is_lose

```
def is_lose():
    global game_status
    if abs(snake.xcor() - monster.xcor()) < 20 and abs(snake.ycor() - monster.ycor()) < 20:
        lose = turtle.Turtle()
        lose.up()
        lose.ht()
        lose.goto(snake.xcor(), snake.ycor())
        lose.color("purple")
        lose.write("Game Over!!", font=("Times New Roman", 14, "normal"))
        game_status = 3
```

This function is used to check whether we lose or not. lose is a Turtle object which can write the information if we lose.

- stop, right, left, up, down

```
def stop():
    global now_direction
    global last_direction
    global game_status
    if now_direction == 'stop' and last_direction != 'null':
        now_direction = last_direction
        last_direction = 'null'
    else:
        last_direction = now_direction
        now_direction = 'stop'

def right():
    global now_direction
    global game_status
    now_direction = 'right'
    game_status = 2

def left():
    global now_direction
    global game_status
    now_direction = 'left'
    game_status = 2

def up():
    global now_direction
    global game_status
    now_direction = 'up'
    game_status = 2

def down():
    global now_direction
    global game_status
    now_direction = 'down'
    game_status = 2
```

This five function is used to change the direction of the snake. The variable now_direction is a string which represents the direction of the snake right now. last_direction represents the direction of the snake last time.

- is_edge

```
def is_edge():
    global now_direction
    if snake.xcor() >= 239 and now_direction == 'right':
        now_direction = 'stop'
    elif snake.xcor() <= -239 and now_direction == 'left':
        now_direction = 'stop'
    elif snake.ycor() >= 199 and now_direction == 'up':
        now_direction = 'stop'
    elif snake.ycor() <= -279 and now_direction == 'down':
        now_direction = 'stop'
    fx_screen.ontimer(is_edge, speed)
```

This function is used to check whether the snake touches the edge of the gaming board right now.

- has_eaten

```
def has_eaten():
    global eaten
    global game_status
    global generated_tails
    if game_status == 3:
        return
    for i in range(9):
        if eaten[i] == True:
            continue
        elif abs(snake.xcor()-food_items[i].xcor()) <= 10 and abs(snake.ycor()-food_items[i].ycor()) <= 10:
            food_items[i].clear()
            generated_tails += i+1
            eaten[i] = True
```

This function is used to check whether the food items have been eaten. The variable eaten is a list which store the information of the result (True or False for every food item).

- extend

```
def extend():
    global generated_tails
    speed = 450
    snake.stamp()
    snake.setheading(angle[now_direction])
    snake.forward(distance[now_direction])
    snake_x.append(snake.xcor())
    snake_y.append(snake.ycor())
    generated_tails -= 1
```

This function is used to implement of the extend of the snake. The variable generated_tails is an integer which represents how many squares should be added.

- normal_move

```
def normal_move():
    global generated_tails
    speed = 300
    snake.stamp()
    snake.setheading(angle[now_direction])
    snake.forward(distance[now_direction])
    snake_x.append(snake.xcor())
    snake_y.append(snake.ycor())
    snake.clearstamps(1)
    snake_x.pop(0)
    snake_y.pop(0)
```

This function is used to implement the normal move of the snake.

- snake_move

```
def snake_move():
    global game_status
    global speed
    global generated_tails
    fx_screen.update()
    is_win()
    is_lose()
    is_edge()
    has_eaten()
    if game_status == 3:
        return
    if game_status == 4:
        return
    if game_status == 1 or now_direction == 'stop':
        fx_screen.ontimer(snake_move,speed)
        return
    snake.color("blue","black")
    if generated_tails:
        extend()
    else:
        normal_move()
    snake.color("red","red")
    fx_screen.update()
    fx_screen.ontimer(snake_move,speed)
```

This function is the core of the movement of the snake. It can invoke extend or normal_move functions again and again if the game is normally playing so that the snake can move.

- is_x_longer

```
def is_x_longer():
    return abs(monster.xcor() - snake.xcor()) >= abs(monster.ycor() - snake.ycor())
```

This function is used to check whether the x distance between the head of the snake and the monster is larger than the y distance. It will be used in the algorithm of the movement of the monster.

- change_m_direction

```
def change_m_direction():
    global m_direction
    if game_status == 1:
        m_direction = 'stop'
        return

    if monster.xcor() < snake.xcor() and monster.ycor() < snake.ycor():
        if is_x_longer():
            if m_speed <= speed:
                m_direction = 'right'
            else:
                m_direction = 'up'
        else:
            if m_speed <= speed:
                m_direction = 'up'
            else:
                m_direction = 'right'

    elif monster.xcor() < snake.xcor() and monster.ycor() > snake.ycor():
        if is_x_longer():
            if m_speed <= speed:
                m_direction = 'right'
            else:
                m_direction = 'down'
        else:
            if m_speed <= speed:
                m_direction = 'down'
            else:
                m_direction = 'right'
```

```
    elif monster.xcor() > snake.xcor() and monster.ycor() < snake.ycor():
        if is_x_longer():
            if m_speed <= speed:
                m_direction = 'left'
            else:
                m_direction = 'up'
        else:
            if m_speed <= speed:
                m_direction = 'up'
            else:
                m_direction = 'left'

    elif monster.xcor() > snake.xcor() and monster.ycor() > snake.ycor():
        if is_x_longer():
            if m_speed <= speed:
                m_direction = 'left'
            else:
                m_direction = 'down'
```

```
    else:
        if m_speed <= speed:
            m_direction = 'down'
        else:
            m_direction = 'left'

elif monster.xcor() == snake.xcor():
    if monster.ycor() < snake.ycor():
        m_direction = 'up'
    else:
        m_direction = 'down'

elif monster.ycor() == snake.ycor():
    if monster.xcor() < snake.xcor():
        m_direction = 'right'
    else:
        m_direction = 'left'
```

This function change the direction of the monster according to the algorithm I have written before

- check_if_contact

```
def check_if_contact():
    global contact
    x = monster.xcor()
    y = monster.ycor()
    is_contact = False
    for i in range(len(snake_x)):
        if abs(x - snake_x[i]) <= 20 and abs(y - snake_y[i]) <= 20:
            is_contact = True
            break
    if is_contact:
        contact += 1

    contact_pen.clear()
    set_contact()
```

This function is used to check whether the body of the snake can the monster contact. The variables x and y is the coordinates of the monster. is_contact is a boolean.

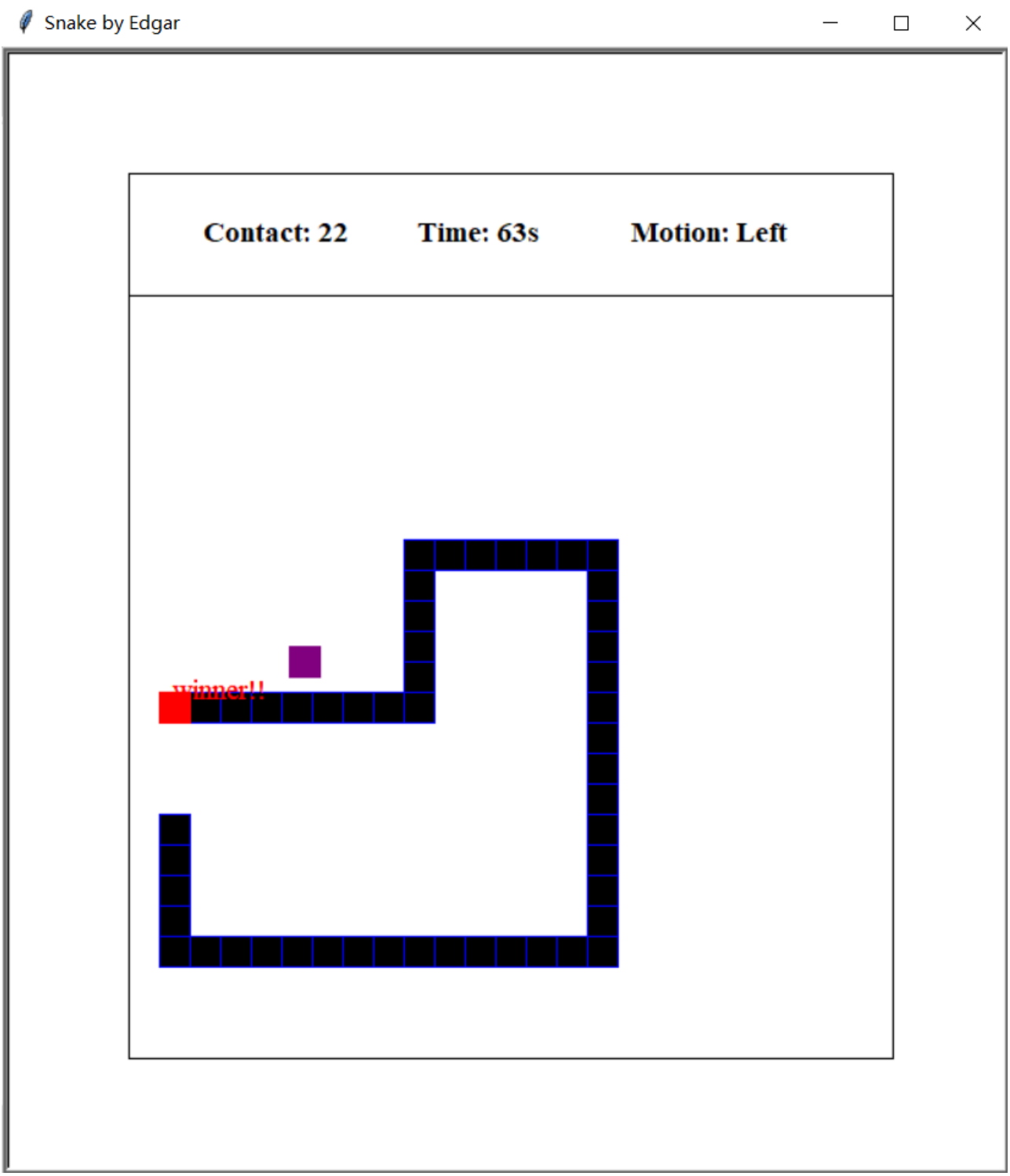
- monster_move

```
def monster_move():
    global game_status
    is_win()
    is_lose()

    if game_status == 3:
        return
    if game_status == 4:
        return

    change_m_direction()
    m_speed = random.randint(260,550)
    monster.stamp()
    monster.setheading(angle[m_direction])
    monster.forward(distance[m_direction])
    monster.clearstamps(1)

    check_if_contact()

    fx_screen.update()
    fx_screen.ontimer(monster_move, m_speed)
```

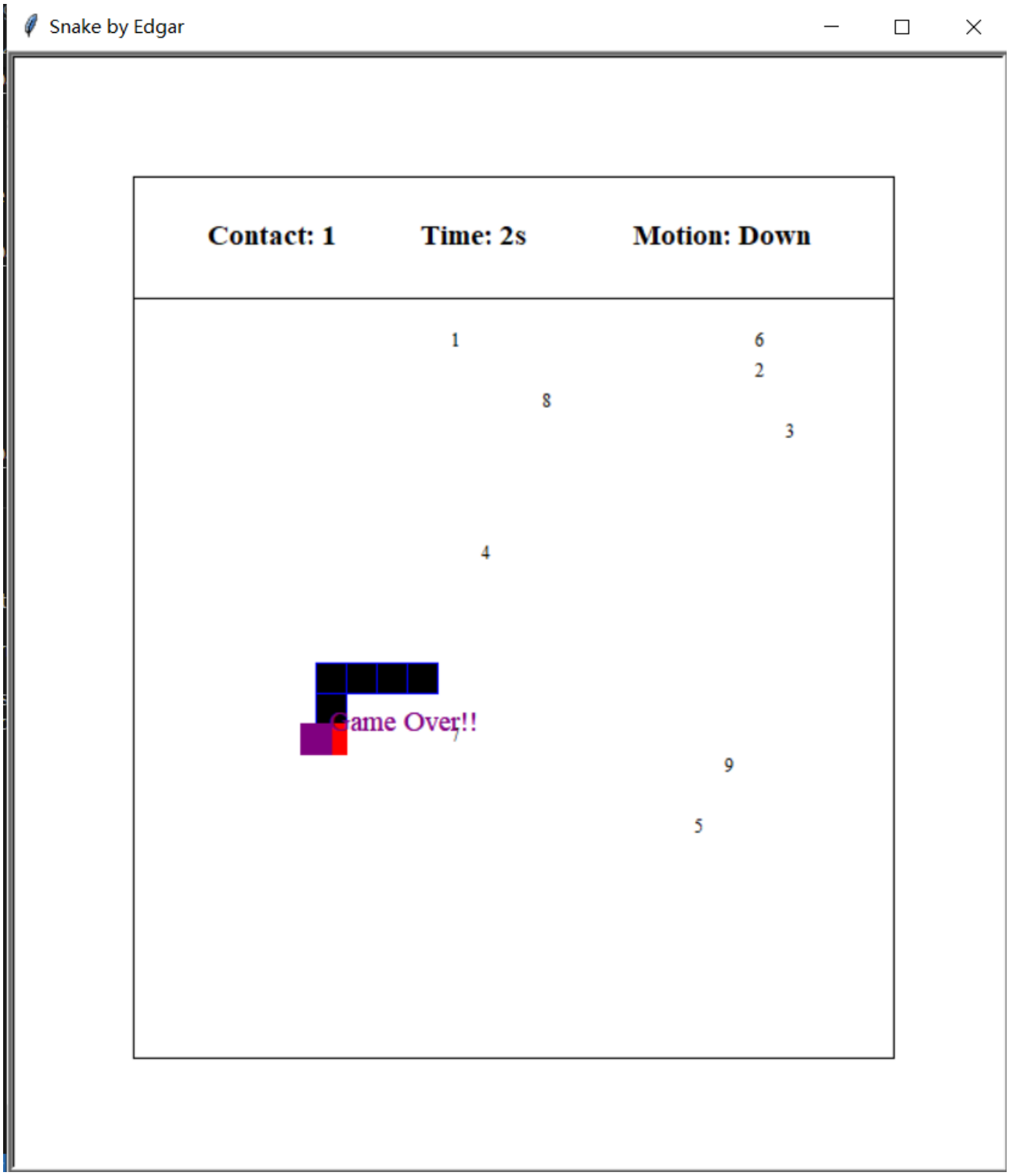
This function is the core to implement the movement of the monster. It uses the ontimer method to move the monster again and again.

3. Output

1. Winner

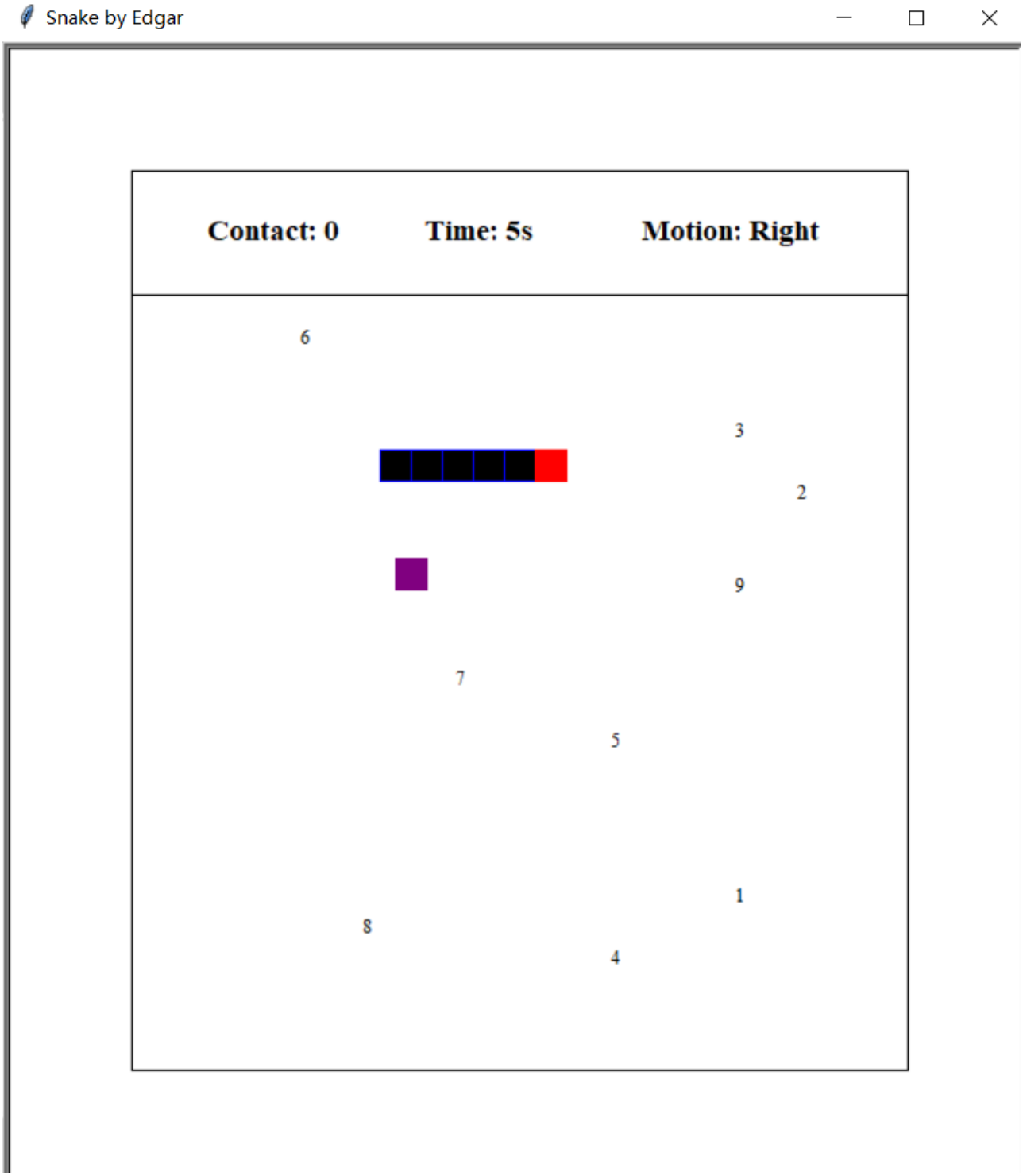


2. Game over

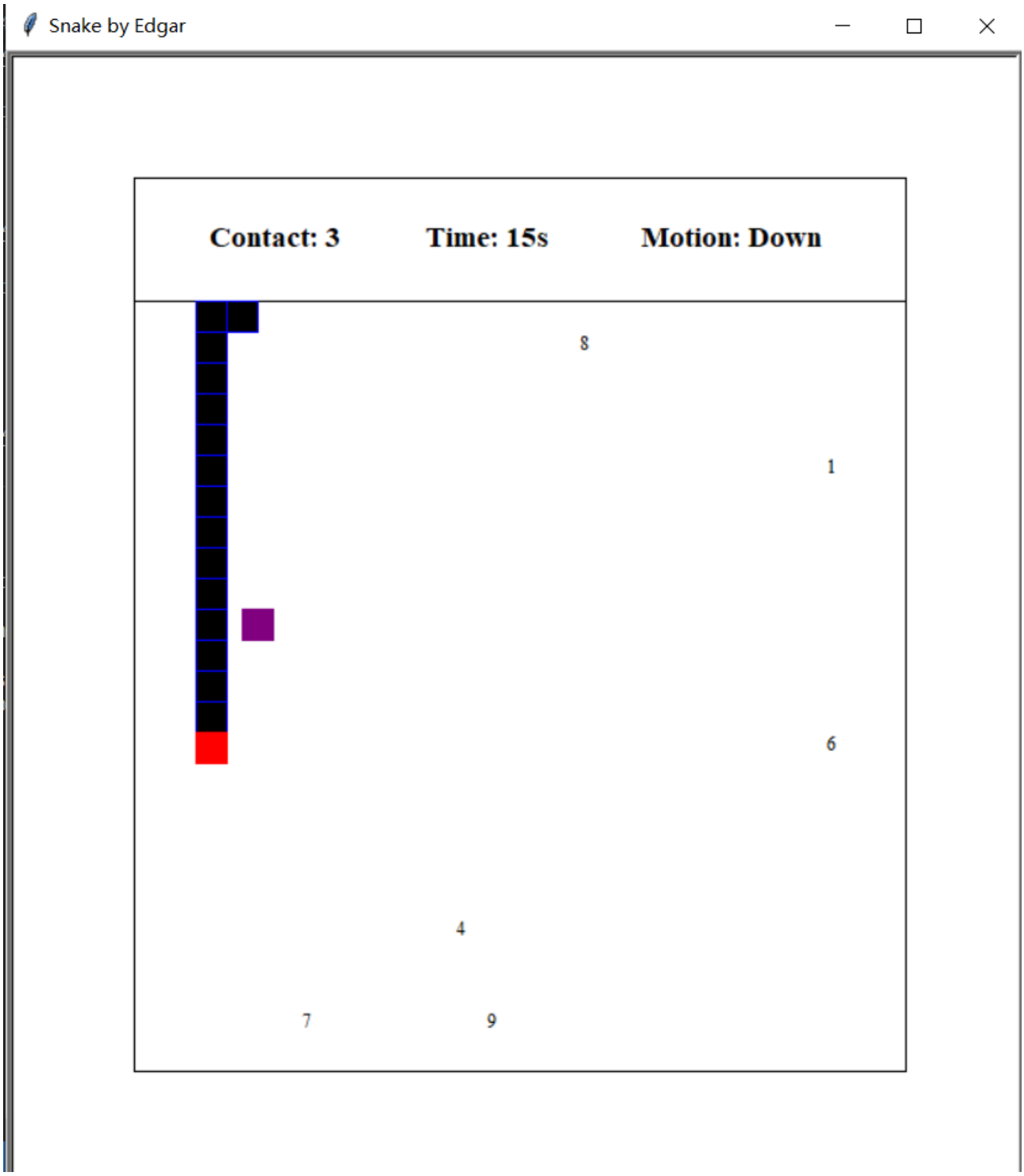


3. 2 others showing various stages of the game

- With 0 food item consumed



- With 3 food items consumed


Snake by Edgar
Contact: 3
Time: 15s
Motion: Down
8
1
6
4
7
9