Design Document

I. *DESIGN*

a. Overview

The game is composed of 3 objects: a snake (pink square head with gold square tail), a monster (red square) and food items (number 1 to 9) represented by a set of numbers from 1 to 9. Players need to use the four arrow keys to control the movement of the snake to eat food and avoid the beast's pursuit. The snake wins by eating all the food before being chased, and loses if it is chased by the beast.

b. Data Model

`BOUNDAYR_POINTS`: list	Store the coordinates of the boundary points
`p_snake_status`: dict	Record the direction of the snake and whether to pause
`DIRECTION`: dict	Indicates the four directions of up, down, left, right
`g_food_location`: dict	Initialize and store coordinates for food
`FOOD_EATEN`: dict	Store coordinates for all food after it has been eaten
`g_tails_length:` int	Record the length of the snake's tail
`g_tails_location`: list	Record the coordinates of the snake's tail
`g_body_contact`: int	Record the body contacts between the snake and the monster

c. Program Structure

First, we create a turtle.Screen() instance (generate game interface) and three major turtle.Turtle() instances(generate snakes, monsters and food). The remaining three auxiliary instances are also generated at the same time to draw the boundaries and inform the game information and status respectively. After entering the `mainloop`, we use `snakeScreen.listen()` and `snakeScreen.onkey()` to monitor player's input. Subsequently, three functions, `move()`, `eat()` and `chase()`, will be executed based on player's input. Keep cycling until winning or losing.

d. <u>**Processing Logic**</u>

i. For snake, we use `snakeScreen.listen()` and `snakeScreen.onkey()` to monitor player's input, thus obtaining a certain direction. Look in the `DIRECTION` for the angle that corresponds to. Use `snake.setheading()` to change direction and move forward by `snake.forward()`. And, we designed the boundary detection to back off when the snake hits the boundary by `snake.backward()`.

For monster, we use `snake.distance(monster)` to go get the distance between the snake and the monster, if distance is smaller than 20, game over. In other cases we use `snake.xcor()` - `monster.xcor()` and `snake.ycor()` - `monster.ycor()` to obtain the horizontal and vertical distances of the monster and the snake, respectively. If `abs(snake.xcor() - monster.xcor())` > `abs(snake.ycor() - monster.ycor())`, monster moves 20 pixels along the x-axis toward the snake, and vice versa. Specifically, we determine the direction of movement by judging the pothe negative and positive of *abs(distance)/distance*.

At the same time, we control the speed of the snake and the beast by coefficient and `random.randint()`.

ii. `g_tails_length` is used to express the total length of the snake's tail (including both grown and ungrown). `g_food_location` uses the coordinates for storing each section of the tail. `ungrown_tails` uses to express tails ungrown. We use `snake.stamp()` copy the snake's head as the tail. And if ungrown_tails > 0, `g_tails_location.append(snake.position())`, indicating that the tail is being generated. If `ungrown_tails = 0`, `g_tails_location.append(snake.position())`, and `del g_tails_location[0]`, add one and delete one to indicate the movement of the snake.

iii. We mentioned in 2 that each section of the tail has coordinates. So we use `monster.distance(single_tail[0], single_tail[1])` to get the distance, if it is less than a certain value, body contact plus one.

II. FUNCTIONAL SPECIFICATIONS

`initTurtle(p_turtle, p_x, p_y, p_speed, p_shape=None, p_color="black")`: Initialize the turtle instance, set coordinates, speed, shape, and color.

`p_turtle, p_x, p_y, p_speed, p_shape, p_color` represent turtle example, horizontal coordinate, vertical coordinate, speed, shape, and color respectively.

`assignFood()`: Draw food (numbers) based on randomly generated food coordinates using the turtle example

`turnUp()`: Modify the state of the snake `p_snake_status`, "Direction" set to "Up", "Motive" set to "active".

`turnDown()`: Modify the state of the snake `p_snake_status`, "Direction" set to "Down", "Motive" set to "active".

`turnLeft()`: Modify the state of the snake `p_snake_status`, "Direction" set to "Left", "Motive" set to "active".

`turnRight()`: Modify the state of the snake `p_snake_status`, “Direction” set to “Right”, “Motive” set to “active”.

`pause()`: Modify the state of the snake `p_snake_status`, if “Motive” is “active”, set it to “Pause”, if “Motive” is “Pause”, set it to “active”.

`displayStatus()`: Return to the snake's movement, up, down, left, right or pause.

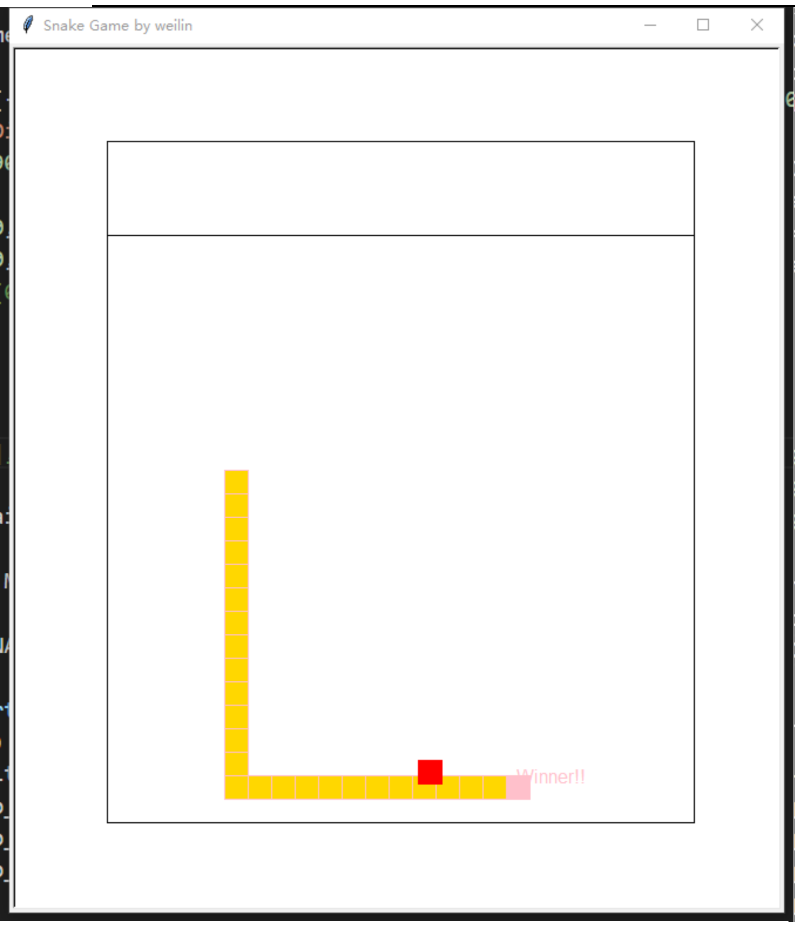
`move()`: Let the snake move in the given direction. And generate tails if there are ungrown tails.

`eat()`: Detect the distance between the snake's head and the food, and if it is less than a certain value, remove the food to increase the length of the tail.

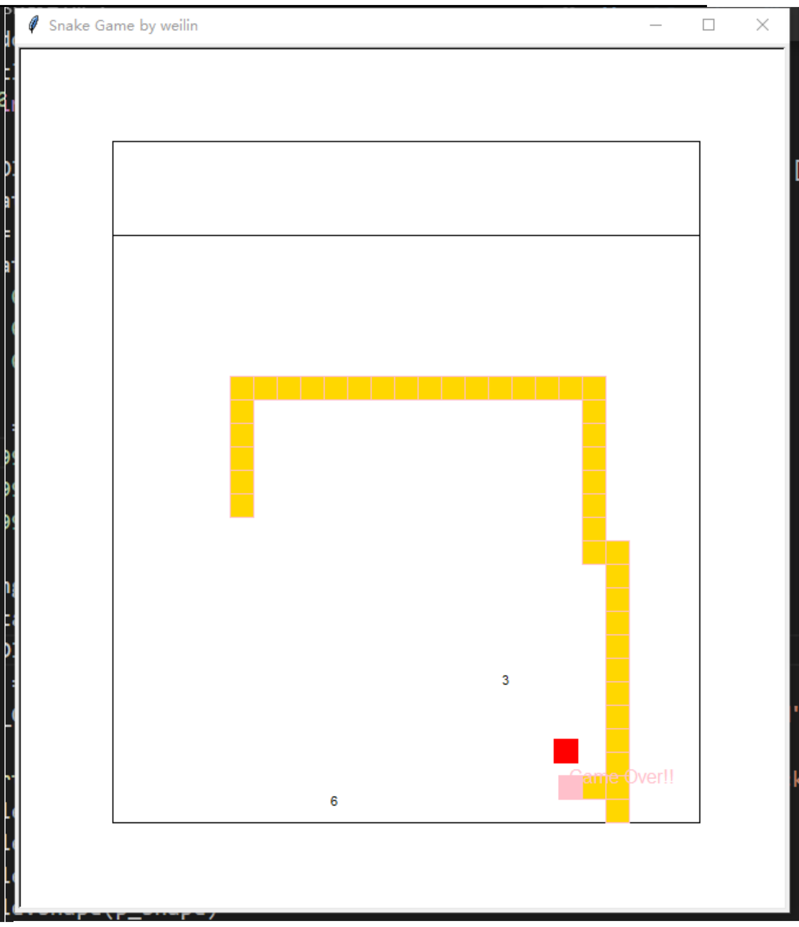
`chase()`: Judge the distance between the monster and the tail and calculate the body contact. Based on the distance between the monster and the snake, let the monster chase the snake. Specifically, we use `snake.xcor()` - `monster.xcor()` and `snake.ycor()` - - `monster.ycor()` to obtain the horizontal and vertical distances of the monster and the snake, respectively. If `abs(snake.xcor() - monster.xcor())` > `abs(snake.ycor() - monster.ycor())`, monster moves 20 pixels along the x-axis toward the snake, and vice versa. For the more, we determine the direction of movement by judging the pothe negative and positive of *abs(distance)/distance*.

`main(p_1, p_2)`: This function is the main function to erase information, display the status, and determine whether the game succeeded or failed.

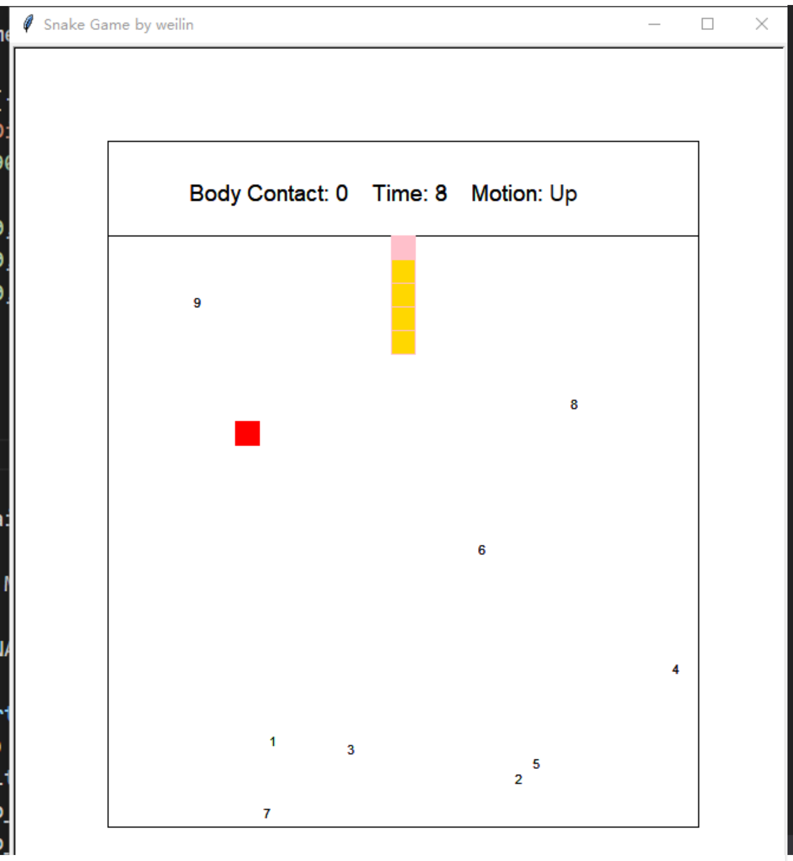
III. SAMPLE OUTPUT



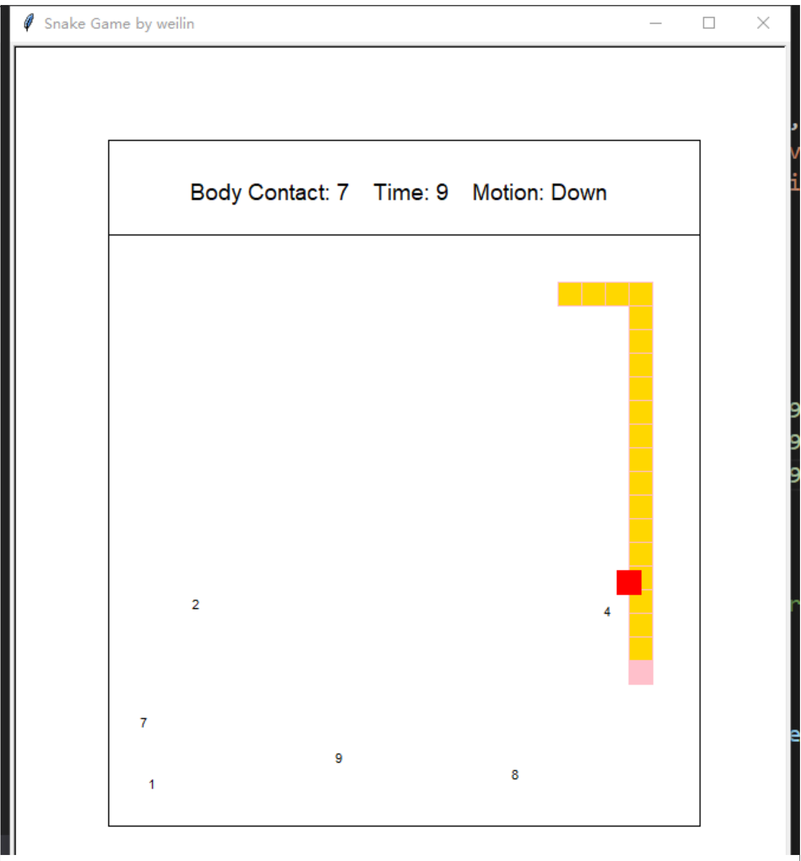
i. Winner



ii. Game Over



iii. 1. With 0 food item consumed



iii. 2. With 3 food items consumed