Design document

1. Design

a. Thought and overall approach

According to the overall requirements, I divided the whole game into five parts.

① The basic set ups, including the screen configuration (size, words, time and other types of displays) and turtle configuration.

② Key configuration, including up, down, left, right, pause and continue.

③ Movement of the snake and monster, where the snake's movement should be related to the key command and the monster's movement should be related to the snake's movement and the head's position.

Every single motion of the snake and the monster should be set in a recursive call with a timer. So that they can move continuously.

④ The extension of snake's tail according to the number of the food it eats.

⑤ Conditional judgement, including whether the snake's head has a collision with the monster, whether the snake's head goes beyond the screen boundary, whether the food were all eaten up and so on.

I use different functions to build up each of these five parts. I use a number of global variables to reset the variables in each function, so that they could be changed when the game precedes. Four functions were used in the end to compile them together. The logic is shown in Figure 1.

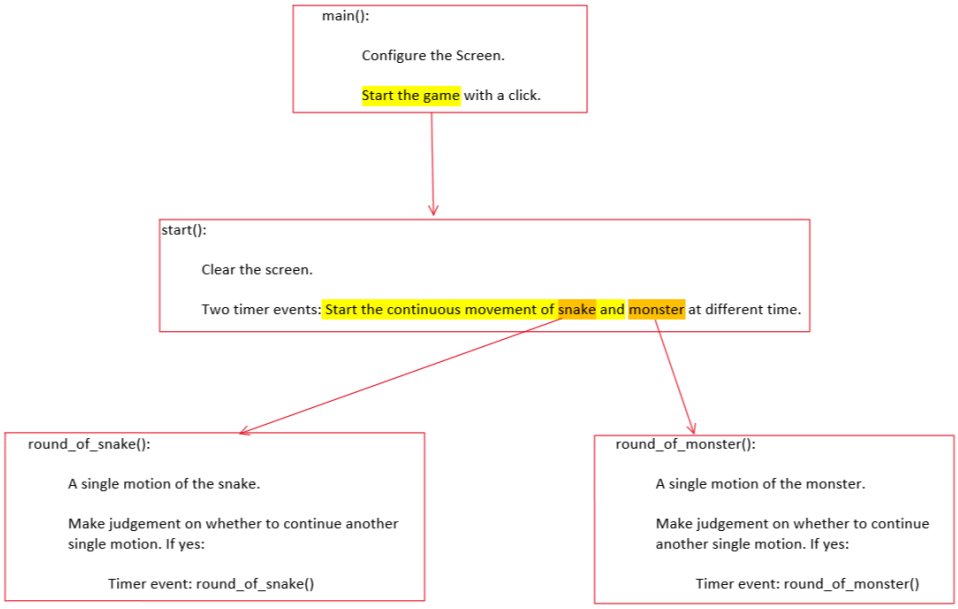

Figure 1

b. Data types

I use the following data types in my program:

- **Numeric:**

Integer: Tracking the index of food, the positions(x and y coordinates) of every turtle (snake's head, each tail section, the monster, the food)

Float: presenting the start time and current time, presenting the angle of the line from the head's position to the monster's position.

- **Boolean:** the return value of the conditional judgement functions, such as "done()" .
- **Sequence type:**

  List: The list consists of tail turtles is used for storing every section of tail. So that tail length can increase by the append method. The list consists of food turtles is used for storing 9 foods. So that food can be moved by the replace a turtle element with a "0".

  Tuple: The position that a turtle object returns is a tuple indicating the x-y coordinate.

c. 1)The motion logic of the snake can be divided into 3 parts.

The first thing is to connect the motion of snake's head to the command from keys.

In part ②, I use "onkey()" to relate the keys to four functions, "up()" "down()" "left()" "right()" and "pause_and_ctn()". These five functions were used to change two global variables, g_direction and g_the_last_direction. g_direction is always changing among five values: "up""down""left""right" and "". g_the_last_direction is only changing among four values: "up""down""left""right", where there is no "". The first global variable is a condition for calling another four motion functions, "move_up()" "move_down()" " move_right()" and "move_left()", which directly trigger a single motion of the head turtle. The reason for setting two separate variables is that, when the snake is paused or makes a collision with the boundary, the g_direction will be set to a "", which means the "if" condition of the four motion functions failed, so that the head can not move. But at the same time the last direction will be saved in g_the_last_direction. Thus, once a continue command is made, g_direction will change back to the value saved in g_the_last direction, and the four motion functions can work again.

The second thing is to conduct a single movement of the snake, including the motion of the head part and the tail part.

In part ③, I set a turtle for the head. Every single motion of the head follows key command and was set in a function called "move_of_head()". I also set a turtle for every section of the whole tail and store them all in a list name "g_tail". The logic for a single motion of the whole tail is that, the first section moves to the last position of the head, and every other section moves to the last position of the section before it. I set the motion of the whole tail in a function called "move_of_tail()".

The last thing is to join every single movement into a continuous loop.

The recursive call is used in the "round_of_snake()" function, meaning that the snake will repeat the single motion again and again with time slot in between. The time slot is designed to be larger when the snake is extending it's tail.

2) The motion logic for monster can be divided into 3 parts.

The first thing is to determine the direction of the monster's next motion.

I divide 360° into 4 equal parts, and use "toward()" to get the angle of the line between the

head and the monster. Figure 2 shows how the angle determines the direction.

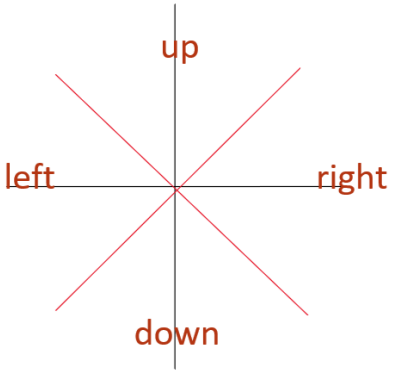

Figure 2

The second thing is to conduct a single movement of the monster. This is straightforward.

The last thing is to join every single movement into a continuous loop.

Same approach is used as the third step in snake's movement. One difference is that, the time slot set between every single movement is randomly larger or smaller than the one set for snake. For the range of fluctuation, I set it be 100ms after several trials.

d. The extension of tail consists of 2 parts. The first one is to get the number that was collided.

Since I store all the food in a list, I identify the number a food representing according to their index. I use a global variable, g_number, to indicate this number.

I use another global variable, g_num_of_waiting_tails, to indicate the number of new sections that are waiting to be appended to the original tail. Its value will increase by the value of the first variable, g_number.

The second step is to append one section to the tail in each motion, until the number of waiting sections comes to 0.

If g_num_of_waiting_tails is not zero, after every motion of the original tail, a new turtle will be created as a new tail section and will be appended to the original tail. (And in the next motion of the snake, this section will be seen as a part of the tail.)

The advantage of separating the process of changing the number of waiting sections and appending a new turtle is that, we can make sure that at the moment the snake's head collides with the food, the food is cleared, the number is added, but the tail is still at it's original length. It was not until the next moment, when the snake moves forward, that the length of the tail increases. (As figure 3 shows)

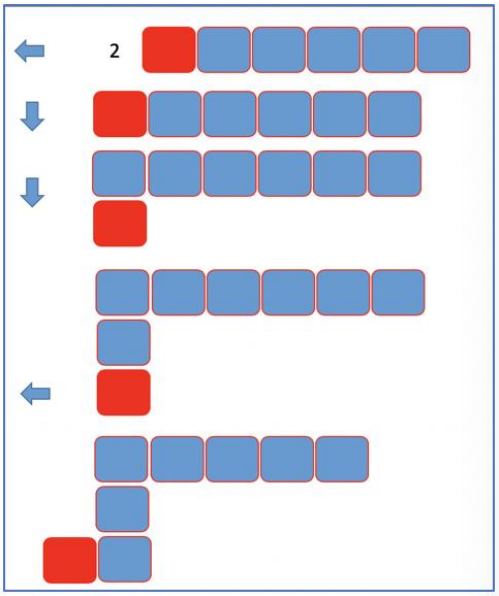

Figure 3

e. I set a global variable, g_collision_time, to indicate the times that there is a tail-monster collision.
Since each section of the tail is a turtle and they were all in the tail list, I use a “for” loop to check every element in the tail list after each motion of the snake and the monster. For any tail section, if the distance between it and the monster >20, the g_collision time will increase by 1.

2. Functions

- **configure_screen():** Set up the screen’s title, size, color and show the instruction.
- **configure_turtle():** Create a turtle with the given parameter: shape, pencolor, fillcolor, x and y. The turtle was set to be penup and go to the position the (x,y).
- **instruction():** Show the instruction on the screen.
- **monster_start_point():** Create a random position for the monster to start.
- **configure_food():** Create 9 turtles as food. Let them go to 9 random position within the boundary on the screen. Save them in a list.
- **configure_keys():** A configuration of five keys. Each key is connect to aanother function.
- **moveUp() / moveDown() / moveLeft() / moveRight():** Move the head directly according to the g_direction. The distance is given by the parameter d = 20.
- **Up()/Down()/Left()/Right()/pause_and_ctn():** Change the value of global variables g_direction and g_the_last_diection, as shown in 1, c(1).
- **move_of_head(direction):** The global variable g_direction will be passed as the parameter. Based on the parameter, this function will call one of the motion functions to conduct a single motion of the head.
- **move_of_tails(direction):** The global variable g_direction will be passed as the parameter. If the parameter is not “”, all tail sections will conduct a motion.

- **move_of_monster():** The angle of line connecting monster and head is passed as the parameter. The function conducts a single motion of the monster based on the parameter. See details in 1. c(2).
- **Food_Collision():** Check if the head collides with a food. If it does, retrieve the number of food and change the value of g_num_of_food. Clear the food on the canvas and replace the turtle in the list with a 0, so that it's no longer a turtle but all other indexes will not be changed.
- **add_new_tails():** Increase g_num_of_waiting_tail by the value of g_num_of_food.
- **append_a_tail_to_body():** If the snake is still moving and there's are tail sections waiting to be appended, create a new turtle and put it at the right position as a new tail section. Also, append it to the list g_tail so that it can move as a part of the whole tale next time.
- **BorderCollision ():** If the head collides with the border, set the global variable, g_direction, into "".
- **count_tail_collisions():** when a tail-monster collision occurs, add 1 to the g_collision_times.
- **round_of_snake():** Repeat the single motion of the snake using a recursive call. All complex cases are considered in this function, including the snake is going through its extension, the head collides with a food, the snake is paused or continued.
- **round_of_monster():** Repeat the single motion of the monster using a recursive call. Update the passed time and the tail-monster collision time by changing the value of g_current_time and call another function "count_tail_collisions()"
- **done():** Check if the game comes to an end. If the monster collides with head, the gamer loses. If the snake eat up all food, the gamer wins.
- **start():** Clear the Screen. Display the time and collision time in the title. Start the first motion of snake and monster at different times.
- **main():** Set up screen, head monster, instruction before user's click.

3. Output

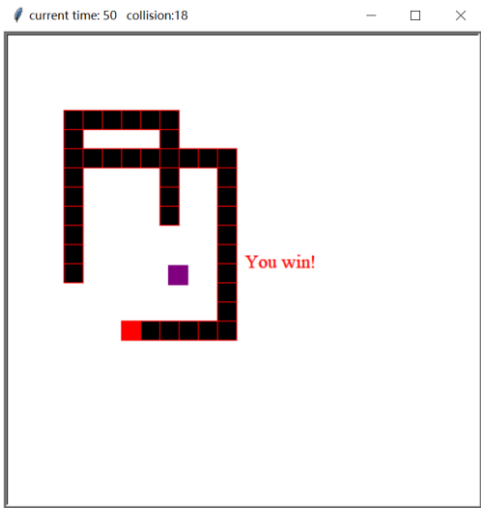

Figure 4. Winner

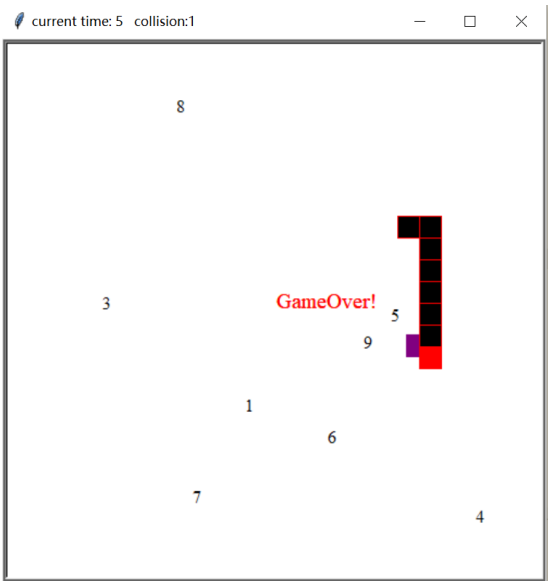

Figure 5. Game Over

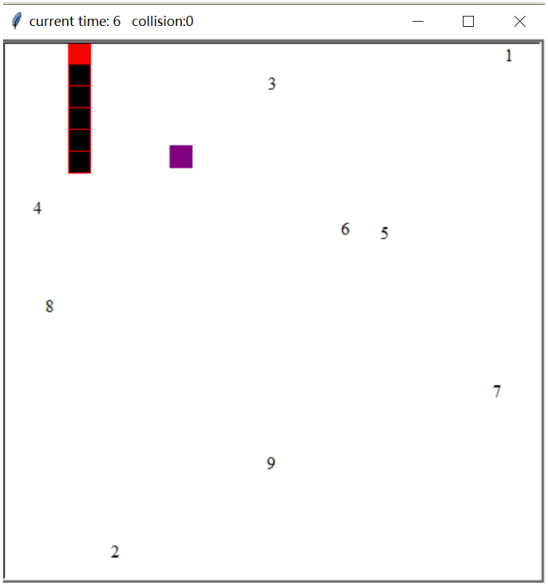

Figure 6. With 0 food consumed

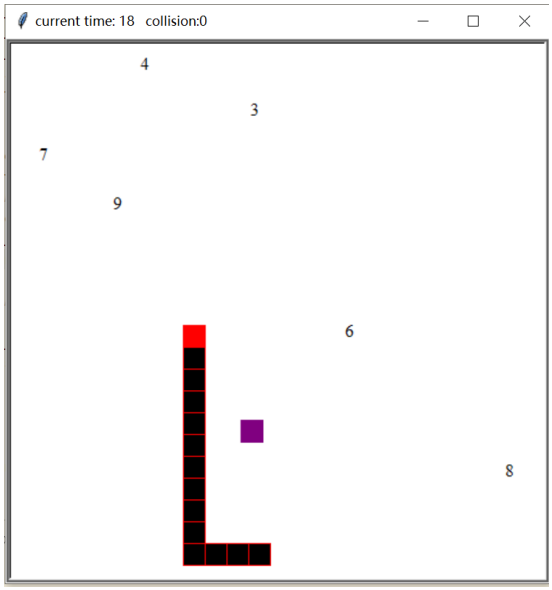

Figure 7. With 3 food consumed