Design document of the Gluttonous Snake

1. Design
2. Description of my thoughts and overall approach

First. The sub components of my logic

1. Show the original status of the game

Write a brief introduction of the game. Show the snake and the monster. Then clear the introduction and show 9 numbers.

1. Design the moving direction of the snake

Guarantee each time the user presses the direction button, the snake can move to a proper direction.

1. Design the body length of the snake

Calculate how long should the snake be in order to decide whether the snake need to clear the earliest stamp of the body.

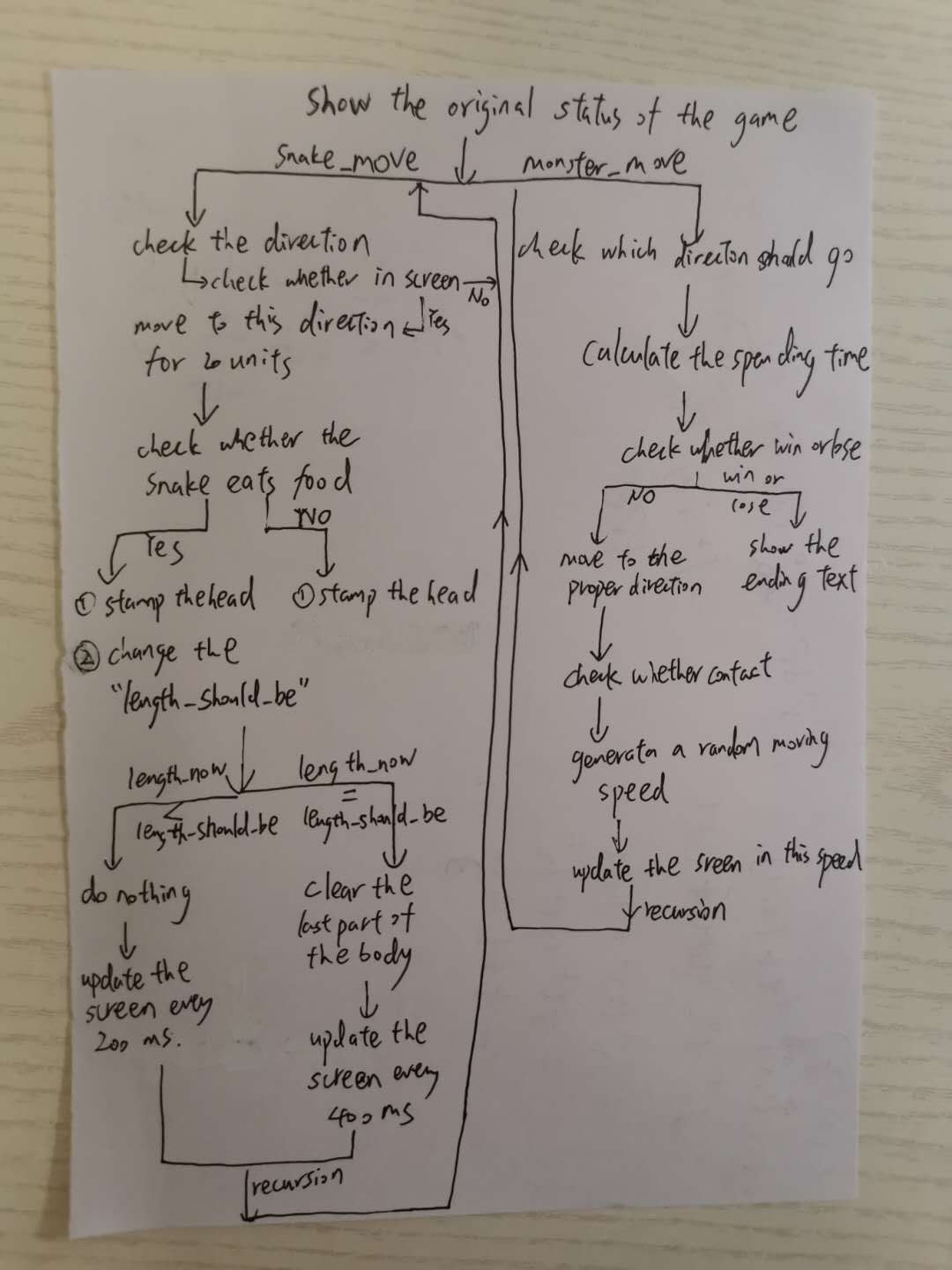
1. Design the motion of the monster and the snake

1. Check whether the snake eats food and change the status of the food
2. Check contact times, check whether the user wins or loses, check whether the snake’s head is in the screen.

Second. the connection of these sub components

There should be a function to control how does the snake move. In this function, “Design the moving direction of the snake” should be executed first to determine which direction to move. Then use “Design the body length of the snake” to control the length of the snake: extend 20 unit, and use another function to determine whether the last part of the snake should be canceled. To calculate how long of the snake should be, “Check whether the snake eats food and change the status of the food” should be implemented.

There also should be a function to control how does the monster move. In this function, “Design the motion of the monster” should be implemented first. The “Check contact times, check whether the user win or lose” function should also be put into this function to determine the game status.



1. Data types
2. food\_x, food\_y, food\_position\_list

Put the position of x-coordinate, y-coordinate, and the complete position into these three lists.

1. food\_list

Store the 9 turtle that represent the 9 numbers in the food\_list.

1. snake\_pos

Every time when the snake head changes its position , the program will store the position of the snake head into the snake\_pos.

1. stamp\_id

Each time the snake head stamps itself, the program will put the stamp id into this list.

1. contact

Each time the snake contacts with the monster, the number of contact times will plus one.

(6)time1, time2, spend\_time

Time1 is to record the starting time of the game. Time2 is to record the current time of the game. Spend\_time is to record how much time have the user used to play the game.

1. direction

Each time the user press the direction button the direction will change to a corresponding angle.

1. length\_now

The length of the snake which will be updated each time the screen updates

1. length\_should\_be

Each time the screen updates, the program will check whether the snake eats food. If it eats, then length\_should\_be will plus the food number. If no, the length should\_be will not change. Originally, length\_should\_be is 5.

1. diff\_x, diff\_y

These data represent the difference between the position of the snake and the monster in x, y coordinate respectively.

1. speed

Speed is to control the update interval of the monster.

1. pause\_pos

Put all the positions that the snake once paused on into this list.

1. direction\_list

Put all the directions that the snake once heading to into this list.

1. start\_times

Originally, start\_times equals to 0. If the start function runs, start\_times will become 1. Thus if the user occasionally click the mouse, the game won’t go back to the starting status.

`

(13) foods, monster, snake

I use turtle() to represent these objects.

(14)Screen

I use screen() to represent the screen.

1. The motion logic for both snake and monster
2. Snake

Set direction as a global variable. Use four directions: right, up, left, down to represent 4 angles of the heading: 0, 90, 180, 270 respectively. Each time the user press the direction button, the direction will change based on the heading angle. If the user doesn’t press the direction button, the direction will remain the same. Each time the screen updates, the program will check the heading direction and the snake will move 20 units to that direction. In particular, if the user press the space button, then the direction will become 360, which means the screen will update directly without letting the snake move for 20 unit first. Each time before the snake move, it should stamp the head to guarantee the snake won’t seems as being cut into two parts.

If the snake move to the right, up, left, down margin of the screen, it can only turn to some particular directions respectively.

The screen will update every 200 millisecond and 400 millisecond when the snake doesn’t eat food or eat food respectively.

1. Monster

If the position difference between the y-axis is larger than the difference in the x-axis and the monster’s position is higher than the snake, then the monster should move down; if the monster’s position is lower than the snake, then the monster should move up. If the position difference between the y-axis is smaller than the difference in the x-axis and the monster’s position is on the left to the snake, then the monster should move right; if the monster is on the right to the snake, then the monster should move left.

In order to let the monster move in a variable and reasonable speed, the screen will update in a speed that is generated from a proper interval randomly.

1. The expansion logic for the snake tail

Whatever the snake eats food or not, it will extend 20 units length toward the newly updated direction. The only difference between eating food or not is whether the snake need to cancel the last part of its body (or called the most earliest stamp of the head). If the snake eats food, the “length\_should\_be” of the snake need to plus the food number. Then compare the “length\_now” with the “length\_should\_be”. If “length\_now” less than “length\_should\_be”, the snake only need to move forward. If “length\_now” equals to “length\_should\_be”, the snake not only need to move forward, but also need to clear the earliest stamp of its body.

1. The body contact logic between the monster and the snake

Each time the snake head move to a new place, I put the coordinate of the head into a list. If the tail need to be canceled, then the tail’s coordinate (the last coordinate in the list) should be removed from the list. So the list is a collection of the position of all the components of the snake. Each time the screen updates, the program will compare the position of every part of the snake (that is all of the position in the list) and the monster. If there exists one position in the list that the position difference between x-axis and y-axis of this position and the monster are both less than 20, than the snake contacts the monster.

1. Functions
2. produce\_numbers()

Randomly generates numbers in range (-240,240). These numbers are the components of the coordinates of the 9 food.

1. food\_turtle\_write\_numbers()

Create 9 food turtles. Let these turtles write the corresponding numbers at a random position.

1. open\_text()

Write a brief introduction of the game. Show the snake in red square at the center of the screen. Show the monster in purple square at the left corner of the screen.

1. start()

Clear the introduction text. Generate 9 foods existing in random places.

1. right(), up(), left(), down(), pause()

Change the moving direction of the snake head. If the user press the space button, then the direction will become 360.

1. extend()

Stamp the snake head. Move the head to a proper direction for 20 units. Change the head into the red colour.

1. check\_eat()

Check whether the snake eat the food. If true, than the length\_should\_be need to add the number of the food, and this food turtle should use clear method to remove itself.

1. check\_in\_screen()

Check whether the snake head is inside the screen boundary.

1. check\_win()

Check whether the snake has consumed all of the food. If yes, show the ending text.

1. check\_contact()

Check whether the snake contacts with the monster. If yes, the contact times should plus one.

1. snake\_move\_part()

Control each 20 units of the move of the snake. If the snake is eating the food, then just move the snake head forward for 20 units and update in a slower speed. If the snake isn’t eating food, then extend the body for 20 units and cancel the last part of the body in a faster speed.

` (12) snake\_move()

If the snake is in the screen range, and the user doesn’t press the space, then directly run the snake\_move\_part() function. If the user press the space button, then directly update the screen without moving the snake. If the snake head is out of the range then only some particular directions can the snake move to, which means only in these directions can the the program runs snake\_move\_part() function.

(13)monster\_move()

Move the monster to a reasonable direction. Calculate the spending time. Calculate the contact time. Change the title. Check whether the user wins or loses.

(14)bind()

Bind the direction buttons with the corresponding functions (up(), down(), left(), right()). Bind the “space” to the pause() function. Bind the mouseclick to the start() function. And use listen() to interact with the user.

(15)main()

Put the open\_text(), snake\_move(), monster\_move(), Bind() and turtle.done() into this main function. Finally only need to run this function.

1. Output

