1. Design

a. Overview

My design mainly includes using turtle class to describe the overall setting, the snake, the monster and the food.

The snake is moving with decent logic (e.g. bounded with a square area, tails lengthened according to certain requirements). Foods are consumed as long as the snake’s head cross it. Monster is designed intelligently to approach the snake every step. The game is over as long as the snake’s head encounters the monster, or all the foods are consumed.

b. Data Model

Snake’s head is in a tuple: “snakeHeadPos”, while its tails are described in a list, “snakeTailPos”.

Foods are described in a list, “foodPos”.

c. Program Structure

My program first initializes all the components, like the border, the monster, and the snake head, shown in the function “xxxxRender”.

Once the user clicks, the program starts looping. “snakeRefresher” and “monsterRefresher” are called alternately to refresh the movement of snake and monster. “snakeRefresher” is further composed of “attemptMove” (try to move one step for the snake), and “drawSnakeDyn” (render the snake and remaining foods after each step). “statusCheck” is included in “snakeRefresher” to check whether the game is over or not, and render the content of status area each step.

d. Processing Logic

The monster is motioned according to the distance from the snake’s head. And I compare the x distance and y distance, and choose the longer distance coordinate to shorten the distance. The snake is motioned according to the last key pressed. If not pressed, keep the current motion status.

If the snake does not eat any food, or after its tail is fully extended, I remove the last part of the tail, the replace the old head with a new tail element. Head is updated according to the current motion. When the snake’s tail is extending (i.e. not fully extended), I did all the things same except that the last tail element is not removed.

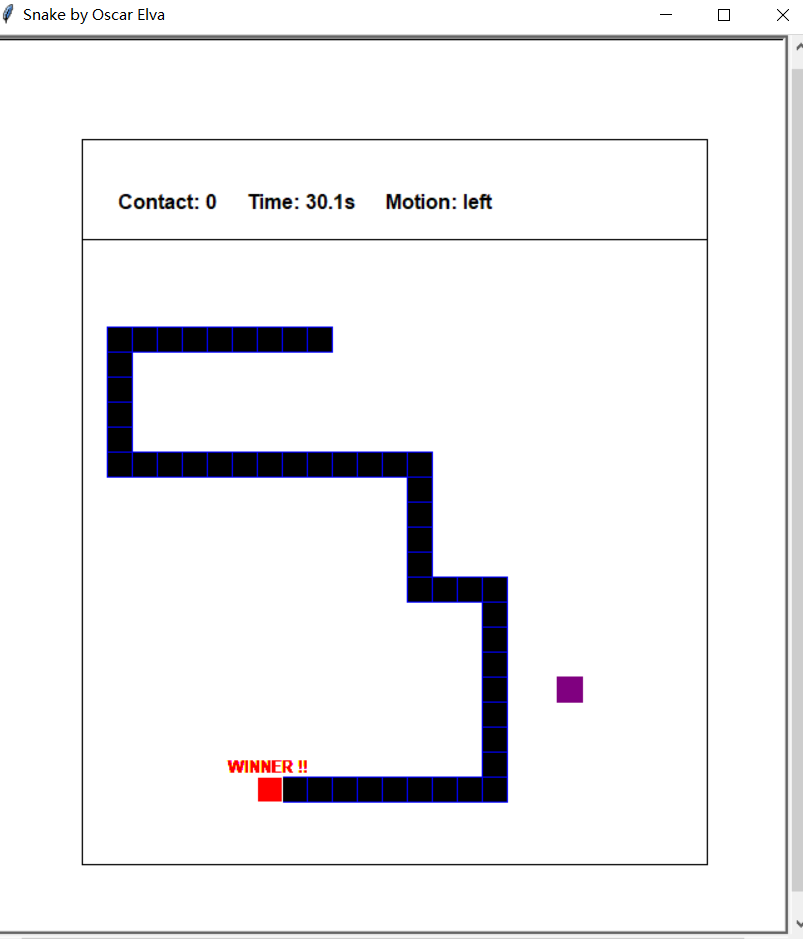
Body contact between these two creatures is detected using a for loop. In the for loop, I compare whether each segment of the tails is overlapped with the monster.

2. Function Specifications

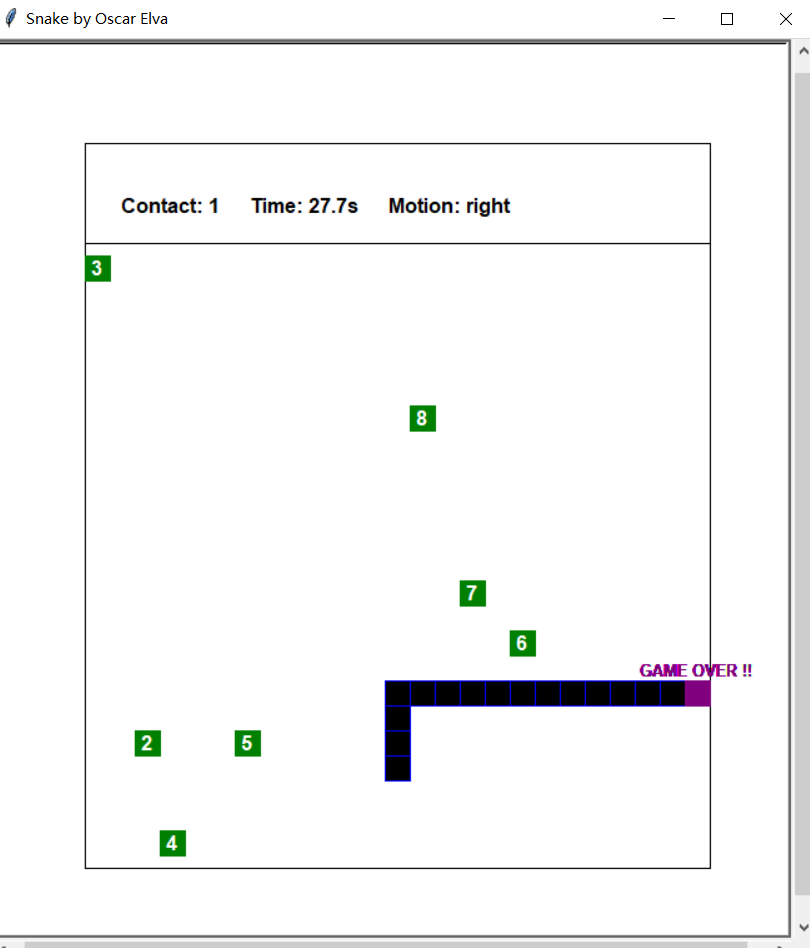
* getRoundPos(myobj, xy): x/y/xy coordinate of the rounded position for the myobj.
* titleRender(): render the initial introduction.
* myCounterRender(): render the initial data in the status area.
* turnUp(obj), turnDown(obj), turnLeft(obj), turnRight(obj): direction changed for the snake.
* attemptMove(): set the next position of the snake’s head.
* Pause\_unpause(): change to the reversed state of isPaused.
* contactCheck(pos, hazard): If none is returned there is no collision, otherwise the coordinates are returned, i is returned for indexing the tuple in the list.
* statusCheck(): check whether the game is over (win/lose), and update the data in the status area.
* snakeRender(): render the initial snake head.
* monsterRender(): render the initial monster.
* foodRender(): render the initial foods, with numbers marked.
* drawSnakeDyn(): update and draw the snake after each step.
* snakeRefresher(): bound attemptMove() and drawSnakeDyn(), and also check status.
* monsterRefresher(): update the monster each step.
* clickStart(a, b): loop after the game starts.

3. Output

I). winner

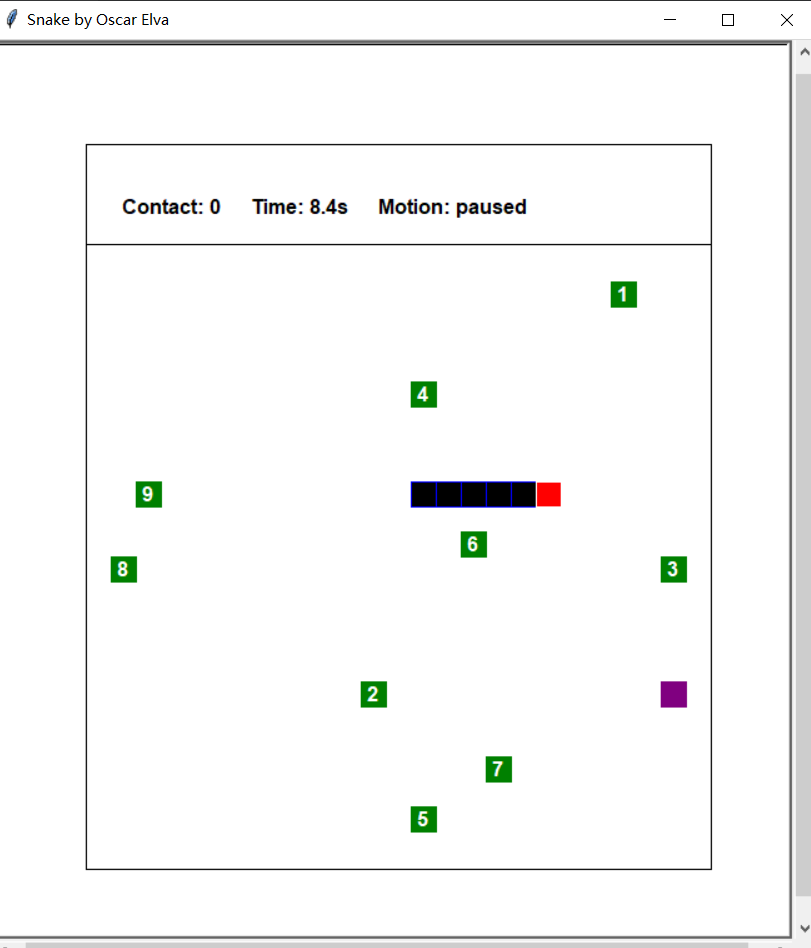


Ii) game over



Iii)

Zero food consumed:



Three foods consumed:

