Design

Overview:

This program can create a 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. The snake is represented by a sequence of squares where its head and its body are displayed in red and black colors respectively, while the monster by a purple square. The foods are represented by digits from 1 to 9. When the snake eats a digital food, it increases the length of the number corresponding to the food. The monster's initial position is random. The monster will always chase the snake head. The goal of user is to control the snake to eat all the digital food and fully extend the snake before the snake is caught by the monster. When user finishes the goal, it will print "Winner!!" on the screen. When the snake head is caught by the monster, it will print "Game over!!" on the screen. The top of the game interface will show the contact times with the monster, the game time and the motion direction of the snake. The snake and monster can only move within a designated area.

The user can click anywhere in the game window to start game. The user can use up, down, left and right arrow keys on the keyboard to control snake's move. And can use space key on the keyboard to pause the snake's move. But the monster will always trace the snake head after game starts. The snake's speed is fixed initially, but it will be slower while the snake being extended, after being extended, it will restore its speed. The monster's speed is random, it will faster or slower than snake's speed.

Data Model:

1. **s, f**:

   Turtle objects. s is used to draw snake, f is used to draw food items
2. **snake, foods**:

   list objects. Every element in snake is also a list, and the inside list contains two elements, they represent every x and y positions of snake's head or body. Every element in foods is also a list, and the inside list contains two elements, they represent every x and y positions of digital food. And the (index in foods list+1) represent the number.

Program Structure:

1. Define 5 constant variables to represent up, down, left, right arrow and space keys on the keyboard.
   KEY_UP, KEY_DOWN, KEY_LEFT, KEY_RIGHT, KEY_SPACE
2. Define a function to change snake's direction.
   change_direction(x, y)
3. Define 5 functions to control snake's motion.
   up() down() left() right() paused()
4. Define a function to bind snake movements to keyboard events.
   control()
5. Define a function to create the main screen of the game.
   create_screen()
6. Define a function to draw status area.

create_status_area()

7. Define 3 functions to draw initial status.
   print_contact()   print_time()   print_motion()
8. Define 3 functions to update status when snake and monster moving.
   update_contact()   update_time()   update_motion()
9. Define a function to draw motion area.
   create_motion_area()
10. Define a function to print introduction of the game on the initial screen,
    print_introduction()
11. Define a function to initially draw digital food items.
    initilize_food_items()
12. Define a function to update food items when one food being eaten.
    update_foods()
13. Define a function to judge whether the snake collide the walls.
    collide_wall()
14. Define a function to draw snake.
    draw_snake()
15. Define a function to make snake move on the screen.
    snake_move()
16. Define a function to initialize monster.
    initilize_monster()
17. Define a function to make monster trace snake.
    monster_move()
18. Define 2 functions to judge whether win the game or game over.
    judge_win()   judge_over()
19. Define a function to integrate what will happen after a mouse click.
    start_game(x, y)
20. Determine the initial value of snake, snake_speed, aim, contact, time, motion.
21. Use create_screen() to create screen g_screen. And turn off automatic refresh: g_screen.tracer(0)
22. Create t turtle object to print Time status on the screen later, create m turtle object to print Motion status on the screen later, create c turtle object to print Contact status on the screen later, create f turtle object to print digital food items on the screen later, create s turtle object to draw snake when it moves on the screen later, create monster turtle object to draw monster when it moves on the screen later. Use initilize_monster() to initially draw monster.
23. Use create_status_area() create_motion_area() print_contact() print_time() print_motion() print_introduction() to draw initial interface.
24. Use onclick() to bond start_game() function to mouse click event.
    g_screen.onclick(start_game)

Processing Logic

snake is a list, and every element in snake is also a list. The sublist contains coordinates of snake head and snake body, and the length of snake list is the total length of the snake's body and head. The last sublist is the snake head’s coordinate, and so forth from the back. If the monster hasn't caught snake head and Motion status isn't Paused, make the coordinates of snake head increase or decrease in the corresponding direction and pop-up the first sublist in the snake list. Then, if the snake head's coordinate is the same as one of the digital foods, add a corresponding number of coordinates to the top of the list. Then, clear old snake that was on the screen, draw new snake according to new snake list. Run through the snake list, let s turtle object go to corresponding coordinates and make stamps, when traversed to the last element of snake, make the stamp of the snake head. Then, judge whether all foods are eaten and whether snake is fully extended or not. When snake is being extended, increases snake refresh time, which slows the snake speed.

monster is a turtle object. First, judge whether user win or not. If True, stop monster move. If False, obtain the current positions of monster and snake head. If snake head is in the upper right corner of the monster, and if the abscissa difference between the snake head and the monster is bigger than the ordinate difference, the monster moves right, otherwise moves up. If snake head is in the right of the monster, the monster moves right. If snake head is in the lower right corner of the monster, and if the abscissa difference between the snake head and the monster is bigger than the ordinate difference, the monster moves right, otherwise moves down. If snake head is in the upper of the monster, the monster moves up. If snake head is in the lower of the monster, the monster moves down. If snake head is in the upper left corner of the monster, and if the abscissa difference between the snake head and the monster is bigger than the ordinate difference, the monster moves left, otherwise moves up. If snake head is in the left of the monster, the monster moves left. If snake head is in the lower left corner of the monster, and if the abscissa difference between the snake head and the monster is bigger than the ordinate difference, the monster moves left, otherwise moves down.

Run through snake, get snake body's position and monster's position, when the distance between them is less than 20, contact will plus one. Because every square of snake body is a square with sides of 20, and monster's sides is also 20. Each time they move, they only move 20. And when random the initial position of monster, I set the step is 20. So, when they overlap, the distance between them must be less than 20, when they don't overlap, the distance between them must be bigger than 20.

Function Specifications

1. up() down() left() right()
   No parameter, no return. They are used to change elements in aim list, which is used to change move direction of snake. They also change Motion status into correspond direction.
2. paused()
   No parameter, no return. If Motion status is not Paused, it changes that into Paused and change elements in aim list to make snake stop. If Motion status is Paused, it changes that into the status before Paused and change elements in aim list to make snake move to correspond direction.
3. control()

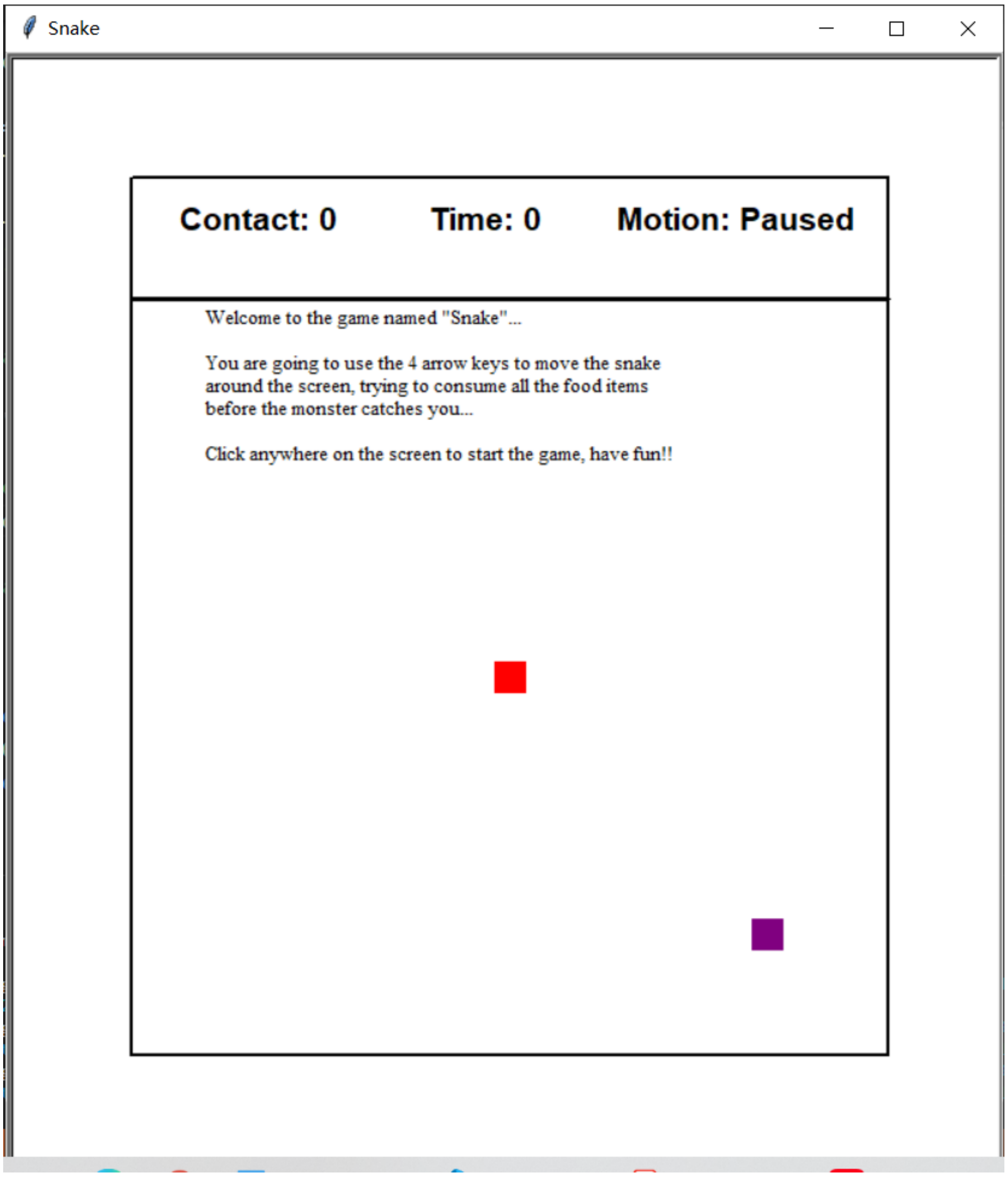
No parameter, no return. It uses Screen,onkey() to bind up(), down(), left(), right(), paused() functions with arrow keys and space key on the keyboard.

4. create_screen()
   No parameter, no return. Create the main screen of the game by using ways in turtle.
5. create_status_area()
   No parameter, no return. Draw the area which is used to show status.
6. print_contact()   print_time()   print_motion()
   No parameter, no return. Show Contact, Time and Motion status on the screen.
7. update_contact()   update_time()   update_motion()
   No parameter, no return. When snake and monster move, update the Contact, Time and Motion status on the screen according to their behaviors.
8. create_motion_area()
   No parameter, no return. Draw the area which is used for snake and monster move.
9. print_introduction()
   No parameter, no return. Print game introduction on the screen initially.
10. initilize_food_items()
    No parameter, no return. Randomly place all digital foods initial positions. They won't overlap. And it also creates a list: foods to store all digital foods' positions. The position is in the center of the digit.
11. update_foods()
    No parameter, no return. When one food is eaten by snake, refresh it be disappeared on the screen and draw the food that hasn't been eaten yet.
12. collide_wall()
    No parameter, return True or False. Determine whether the snake has hit the wall or not. If yes, return True. If no, return False.
13. draw_snake()
    No parameter, no return. According to the position in snake list, draw snake on the screen.\
14. snake_move()
    No parameter, no return. Refresh the move of the snake. Determine what the snake needs to do based on what happens after it moves. If game over, stop snake. If Motion is not Paused and snake doesn't collide wall, if snake eats a food, extend its length. When snake is being extended, slow its speed. Repeat snake_move() until win or game over.
15. initilize_monster()
    No parameter, no return. Randomly place the monster's initial position and draw it on the screen. The monster can't get too close to the snake at first.
16. monster_move()
    No parameter, no return. Refresh the move of the monster. According to the relative positions of monster and snake head, make the monster trace snake head. Repeat monster_move() until win or game over.
17. judge_win()

No parameter, return True or False. Judge whether the snake consumes all foods and its body is fully extended or not. If so, return True, otherwise, return False.

18. judge_over()
    No parameter, return True or False. Judge whether the monster catch up the snake head or not. If so, return True, otherwise, return False.
19. start_game(x, y)
    2 parameters, no return. But the 2 parameters is useless, just for satisfying Screen.onclick(). This function will be executed after user click the screen. It is used to start the game.

Output

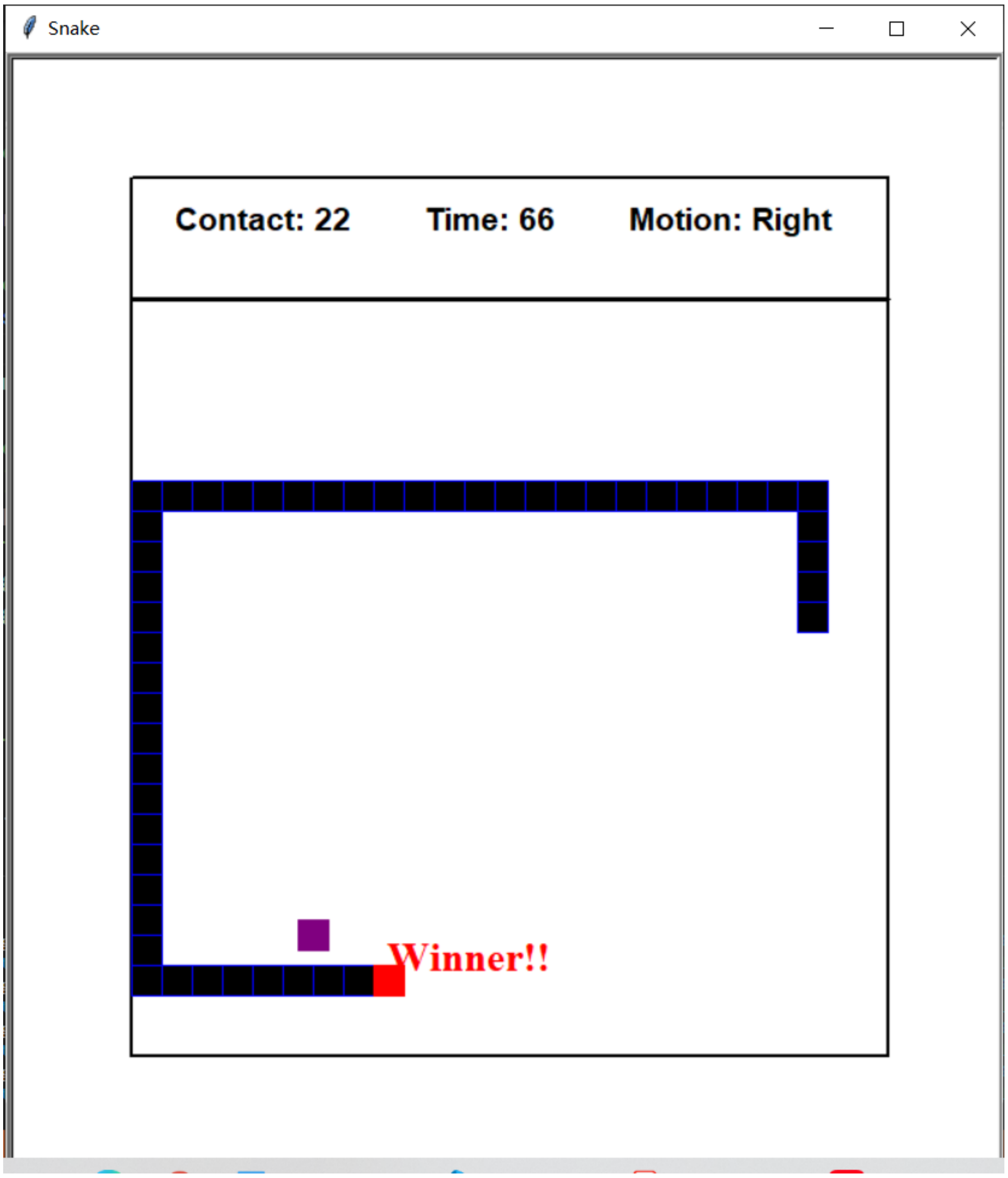


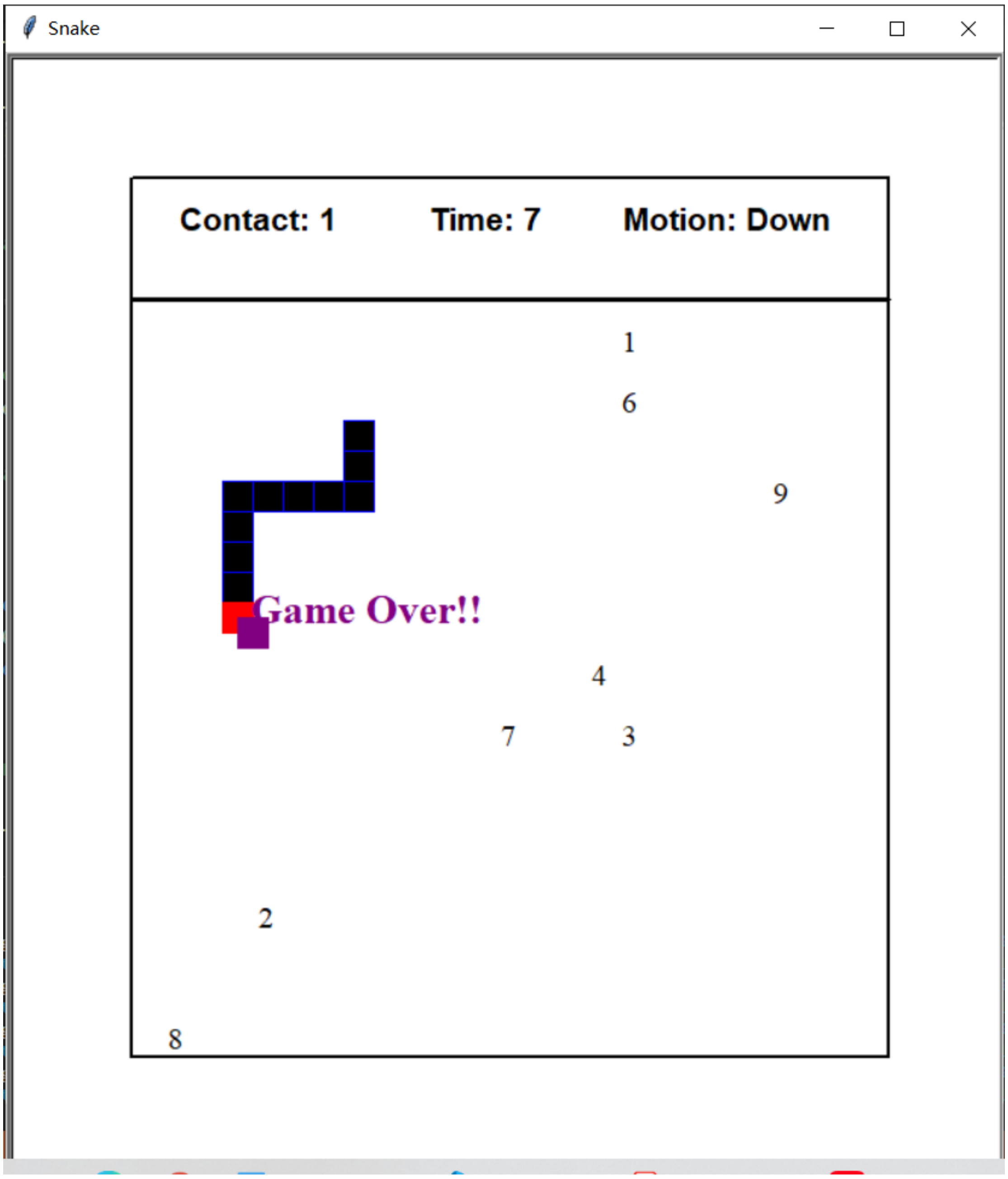
Contact: 0 Time: 0 Motion: Paused

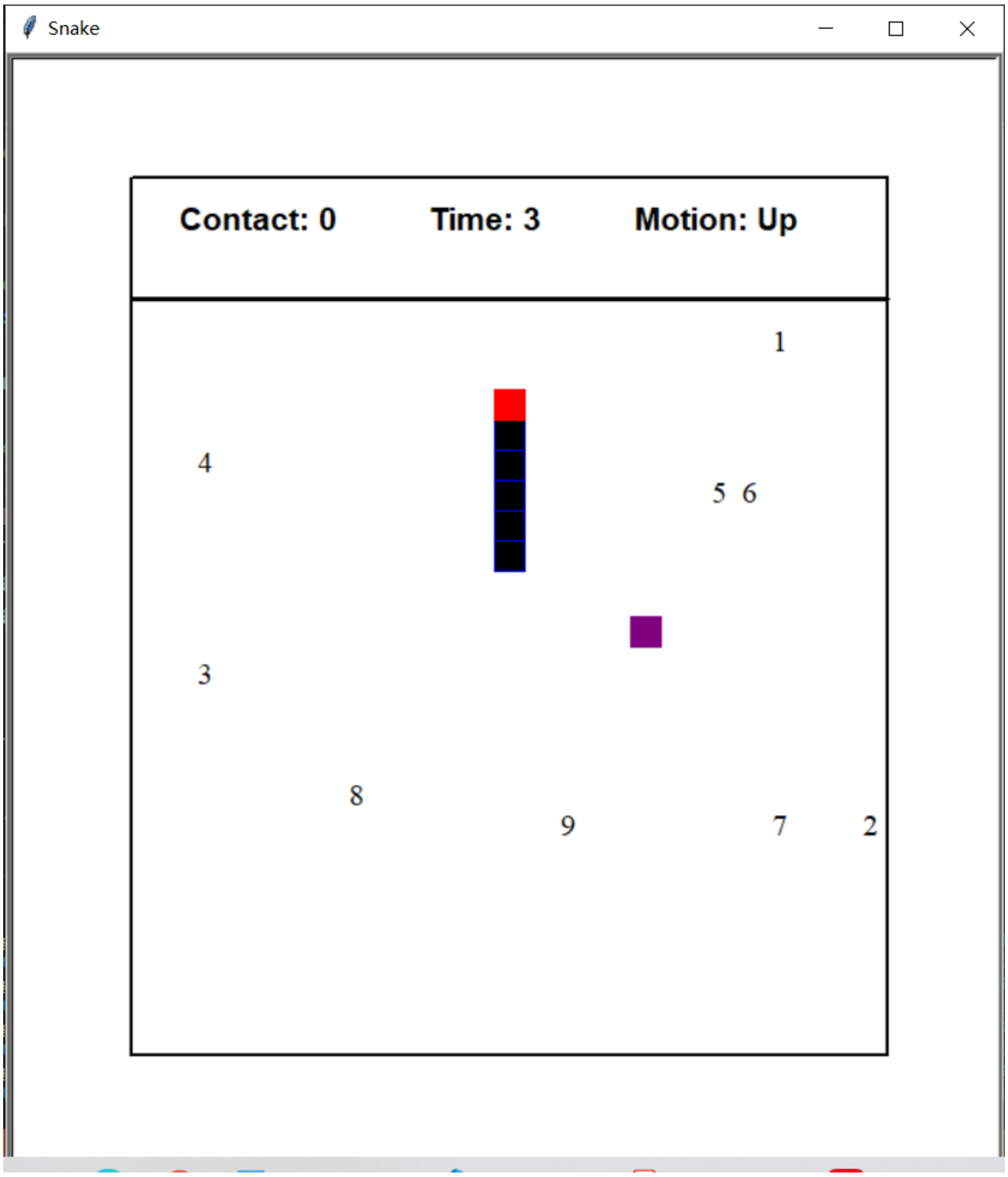
Welcome to the game named "Snake"...

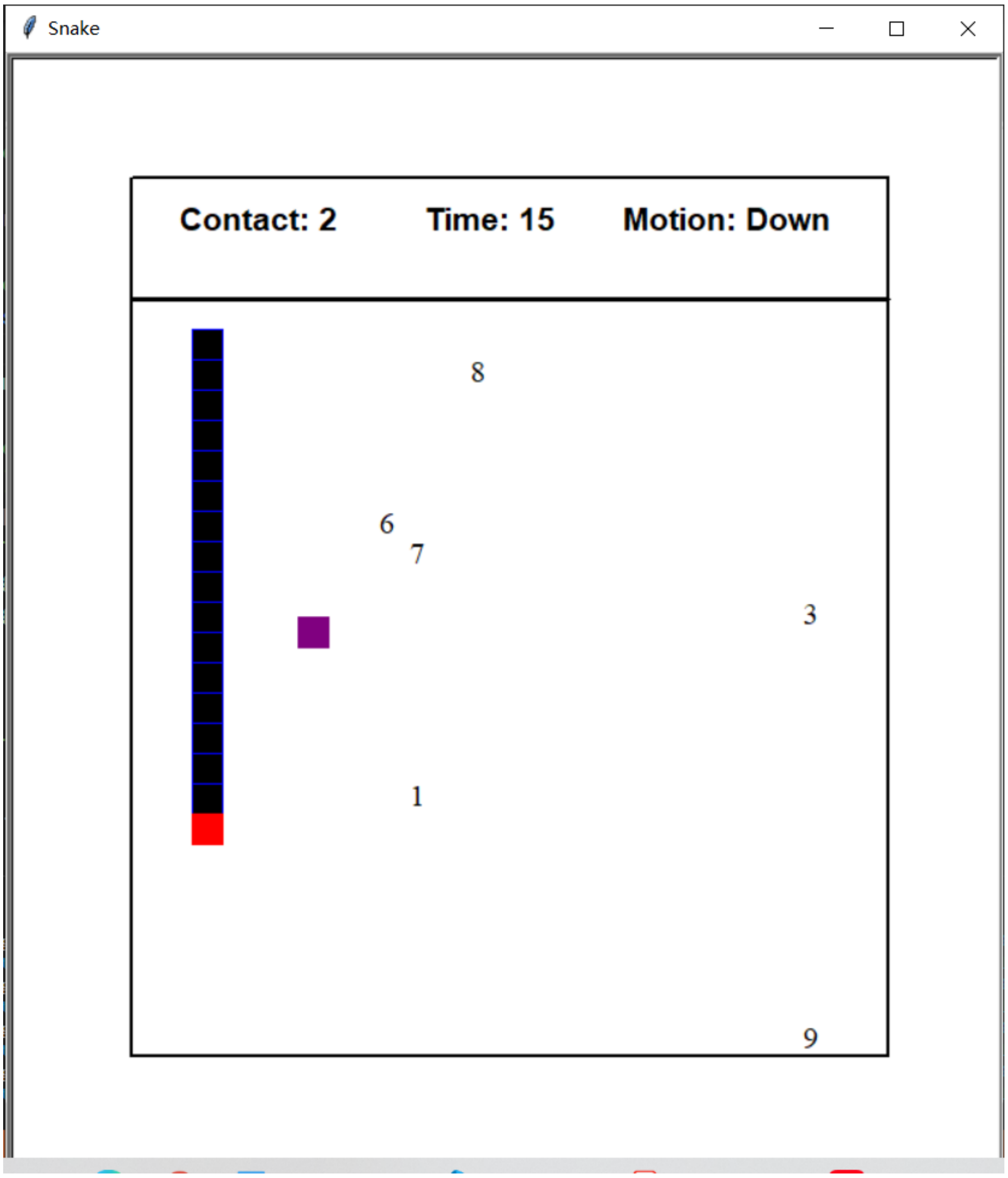
You are going to use the 4 arrow keys to move the snake around the screen, trying to consume all the food items before the monster catches you...

Click anywhere on the screen to start the game, have fun!!


Snake
Contact: 22
Time: 66
Motion: Right
Winner!!


Snake
Contact: 1
Time: 7
Motion: Down
1
6
9
Game Over!!
4
7
3
2
8


Snake
Contact: 0
Time: 3
Motion: Up
1
4
5
6
3
8
9
7
2


Snake

Contact: 2 Time: 15 Motion: Down