Design Documentation

1. Design

a. overview

This snake game has 3 objects: a snake (its body extends after eating food), a monster, 9 food items represented by numbers from 1 to 9. The head of snake is in red and its body is in black. The monster is in purple. After clicking anywhere of the game window, the game starts and the initial motion of the snake is **Right**. The initial body length of snake is 5. User can use four arrow keys (Up, Down, Left and Right) to change the direction of the snake head and make the snake consume food items as many as possible.

b. data model

In my program, I use three lists to store the positions of food items, the current position of snake body and the current ids of stamps respectively.

A variable called *state* has two values: *run* and *hit_boundary*. If it is *hit_boundary*, the snake will stop until you choose a proper direction for snake to go.

A variable called *motion* has five values: *Paused, Up, Down, Left, Right* which is used to be shown in the status bar.

Variables like *g_snake, g_monster, g_introduction, g_statusbar, g_notice* are instances of turtle in the game. *g_introduction* is used to show the welcome and the basic description of the game before the game starts. *g_notice* is used to notice you that the snake has hit the boundary.

There are a few other variables in the program but they will not be shown here because they are not that important and their names show what they really mean.

c. Program Structure

To implement the snake game by using turtle, I break it down into many functions. First, I need **configureScreen()**, **configureBoundary()** and **configureFood()** to initialize the screen, the boundary and food items. I need **configureTurtle()** to generate a monster and a snake head. Then, I need **configureKey()** to bind four arrow keys to four functions to control snake's moving. When the snake is moving (Calling **moveSnake()** makes the snake move), **moveMonster()** let the monster run after the snake.

Each movement of the snake, **checkForDeath(), hitBoundary(), checkForFoods()** will be called to see if the monster catches the snake, if the snake hit the boundary and if the snake eats the food items.

Each movement of the monster, **collapse()** will be called to see if it contacts snake body.

moveSnake() is the most important function in this program. It contains so many functions and it will be illustrated in details later.

If it eats a food item, it will extend its body (add the same number of squares as the number of the food item shows). a key point is how to make the snake extend its body and move its body. This can be solved by using a set of methods called **stamp()** and **clearstamp()**.

d. Processing Logic

(1) Users reads the welcome and the basic description of the snake game. Click the screen to start the game.

(2) The snake moves in Right direction initially. Each move it will check if the monster catches it, if the snake hit the boundary and if the snake eats the food items.

If the snake head is caught, the game ends.

If the snake hit the boundary, the snake stops and wait for user to press keys to change the direction.

If the snake eats a food item, it will extend its body (add the same number of squares as the number of the food item shows)

All these processes are done in **moveSnake()** function.

(3) Users press arrow keys or space to change the direction of snake or pause the game.

(4) If users press space, the game pauses and if users press space again, the game continues.

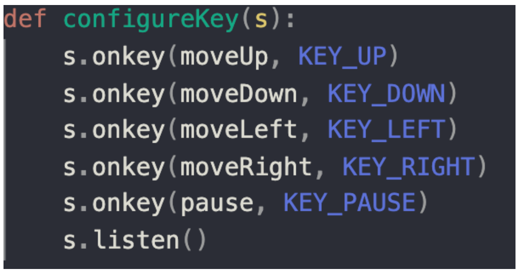
(5) If users press one of the arrow keys, the direction of snake head changes and the snake moves in the corresponding direction.

(6) If 9 food items are consumed, the game ends and users win.

2. Function Specifications

configureScreen(), **configureBoundary()**, **configureFood()**, **configureTurtle()**, **configureStatusBar()**, **configureIntroduction()** are very easily implemented function, just to initialize turtle to draw or write something we want.

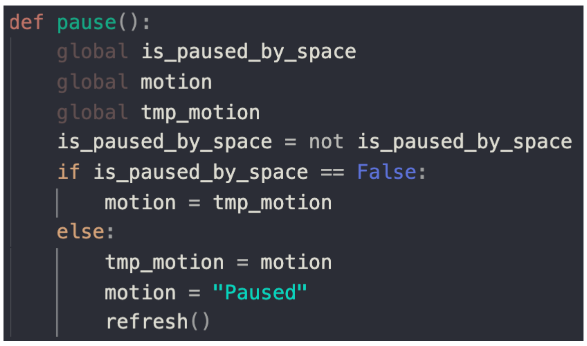
configureKey(): easy function, bind functions to some keys.



```
def configureKey(s):
    s.onkey(moveUp, KEY_UP)
    s.onkey(moveDown, KEY_DOWN)
    s.onkey(moveLeft, KEY_LEFT)
    s.onkey(moveRight, KEY_RIGHT)
    s.onkey(pause, KEY_PAUSE)
    s.listen()
```

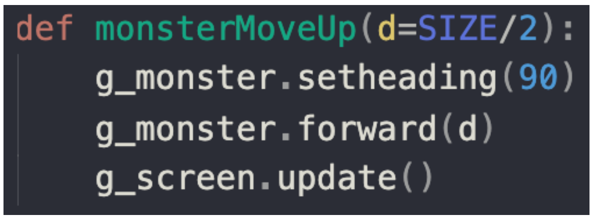
checkForFoods(): check the distance between the snake head and 9 food items. If the distance between the head and one item is less than 20, it shows that the snake eats the item.

pause(): change the value of *is_paused_by_space* when space key is pressed



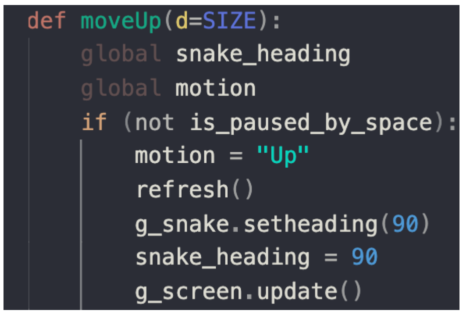
```
def pause():
    global is_paused_by_space
    global motion
    global tmp_motion
    is_paused_by_space = not is_paused_by_space
    if is_paused_by_space == False:
        motion = tmp_motion
    else:
        tmp_motion = motion
        motion = "Paused"
        refresh()
```

monsterMoveUp(), monsterMoveDown(), monsterMoveLeft(), monsterMoveRight() are very similarly implemented functions. The only difference is the argument value of **setheading()** method.



```
def monsterMoveUp(d=SIZE/2):
    g_monster.setheading(90)
    g_monster.forward(d)
    g_screen.update()
```

moveUp(), moveDown(), moveLeft(), moveRight() are very similarly implemented functions. The only difference is the argument value of **setheading()** method.



```
def moveUp(d=SIZE):
    global snake_heading
    global motion
    if (not is_paused_by_space):
        motion = "Up"
        refresh()
        g_snake.setheading(90)
        snake_heading = 90
        g_screen.update()
```

refresh(): refresh the status bar.

printMessage(str): print any message you want on the screen.

checkForDeath(): check the distance between the snake head and the monster. If the distance is less than 10, it means that the monster catches the snake and the game ends.

collapse(): check the distance between two points.

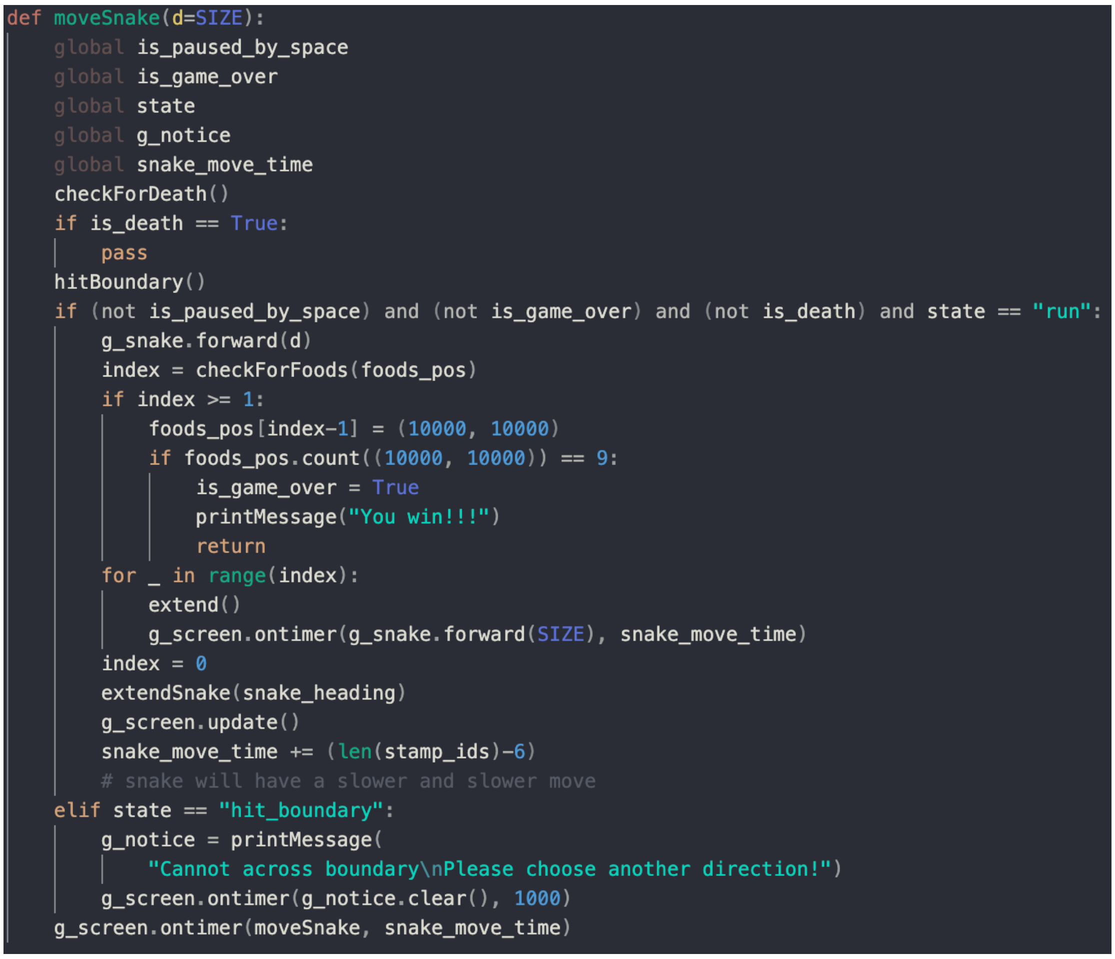
hitBoundary(): check it the snake head hit the boundary.

extendSnake(): "refresh" the snake body. Delete the oldest stamp and make a new stamp at the current position, which seems like the snake is moving if the snake head moves forwards.

moveMonster(): similar to **moveSnake()** but less difficult to understand.

onClick(): call other functions to start the game when users click the screen.

moveSnake():

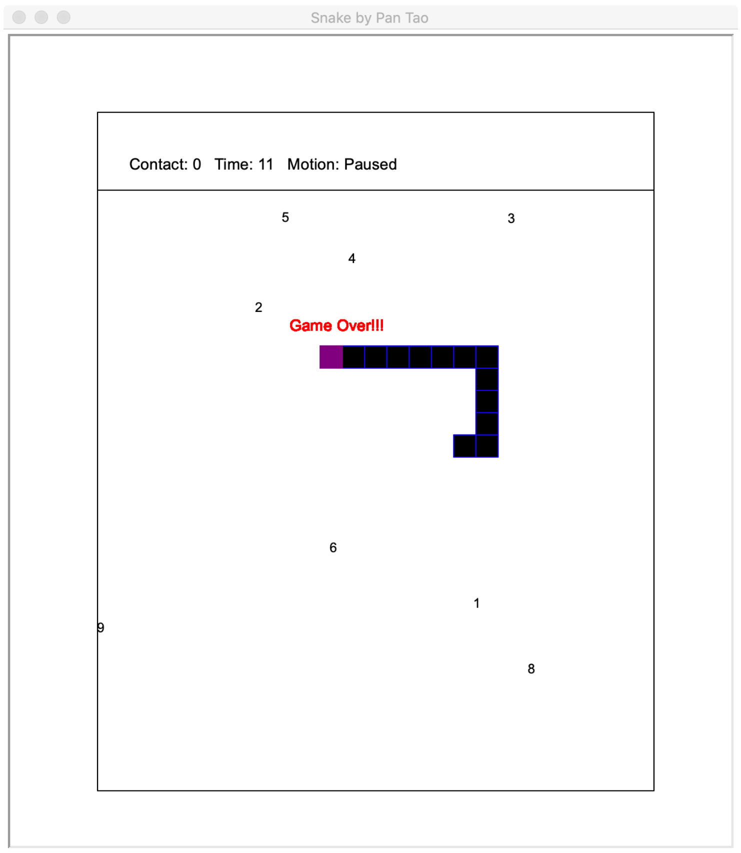


```python
def moveSnake(d=SIZE):
    global is_paused_by_space
    global is_game_over
    global state
    global g_notice
    global snake_move_time
    checkForDeath()
    if is_death == True:
        pass
    hitBoundary()
    if (not is_paused_by_space) and (not is_game_over) and (not is_death) and state == "run":
        g_snake.forward(d)
        index = checkForFoods(foods_pos)
        if index >= 1:
            foods_pos[index-1] = (10000, 10000)
            if foods_pos.count((10000, 10000)) == 9:
                is_game_over = True
                printMessage("You win!!!")
                return
        for _ in range(index):
            extend()
            g_screen.ontimer(g_snake.forward(SIZE), snake_move_time)
        index = 0
        extendSnake(snake_heading)
        g_screen.update()
        snake_move_time += (len(stamp_ids)-6)
        # snake will have a slower and slower move
    elif state == "hit_boundary":
        g_notice = printMessage(
            "Cannot across boundary\nPlease choose another direction!")
        g_screen.ontimer(g_notice.clear(), 1000)
    g_screen.ontimer(moveSnake, snake_move_time)
```

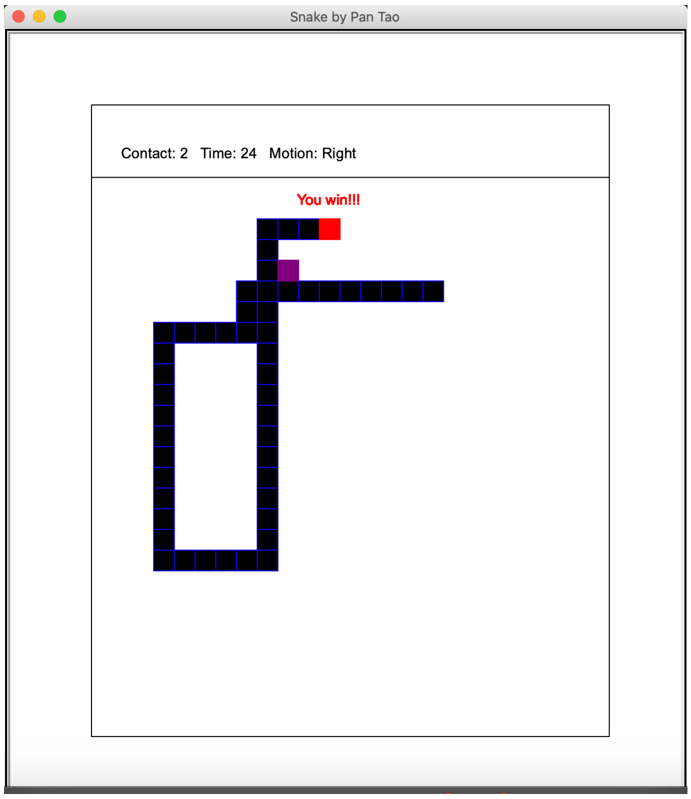
Each time when calling **moveSnake()**, first call checkForDeath() and hitBoundary() to see if the game ends. If not, the snake move forward (the direction is controlled by moveXXXX()). After moving forward, check if the snake eat food items. If it eats, check if it eats all items. If it eats all, users win. If not, extend its body. If the head hit boundary, change state and call **moveSnake()** next to see if user choose a proper direction.

3. Output

i. Game Over

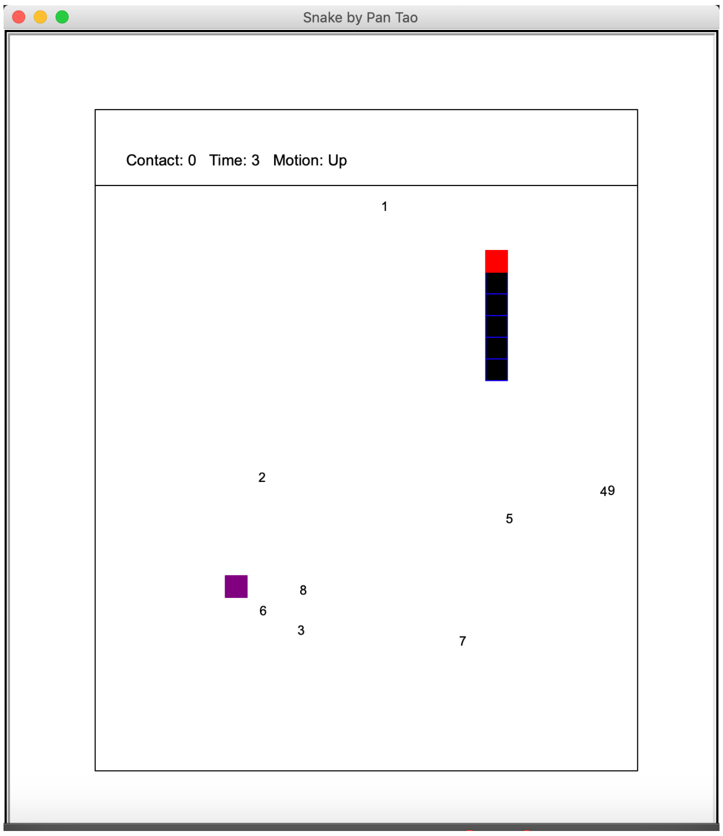


ii. Winner

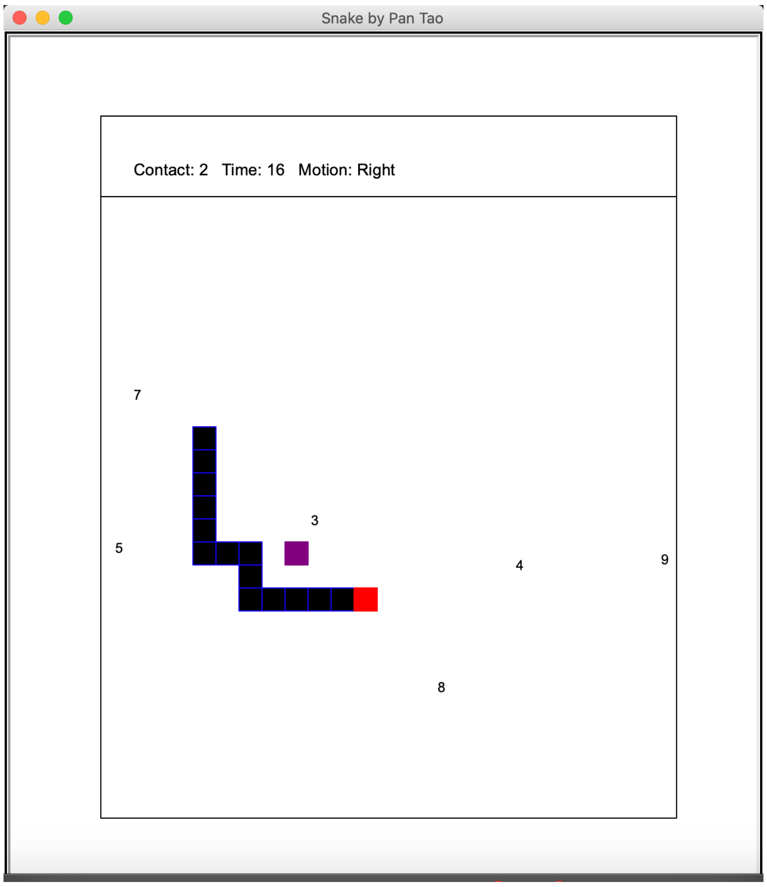


iii. 2 others showing various stages of the game:

1. With 0 food item consumed



2. With 3 food items consumed


Snake by Pan Tao
Contact: 2   Time: 16   Motion: Right
7
3
5
4
9
8