Design Documentation

120090272 Feng Chuqiao SDS Assignment 2 --- Snake

1. Design

a. Overview

The program is designed to develop a snake game for users. The users will be allowed to use for arrows on keyboard to control the direction of snake moving. 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 pink and black colors respectively, while the monster is a blue square. The food items will be consumed by the snake.

The goal of the game is to move the snake within the game area, trying to consume all the food items while avoiding head-on collision with the monster. As each food item is consumed, the snake grows with its body lengthened in size equal to the value of the number being passed. While directing the movement of the snake users should avoid contact with the monster. Furthermore the monster is also programmed to be motioned in the direction towards the head of the snake at a variable speed.

b. Data Model

First, the program generate several turtles. And import turtle as 'tt', 'monster', 'food' to represent the objects of snake, monster and food items respectively for easier control of the main objects. Also, set or use turtle to set the following data structures to denote the characteristics of all objects that needed.

i. Dictionary:

1. **pos_food**: Use a dictionary to store the position of the food numbers in relate to its number. So that in the program, the food items can be easily operated with their numbers.

```
pos_food = {}
pos_food[number] = (x, y)
del pos_food[num] # food consumption
# keys of dictionary are numbers with positions as their values
```

2. **direction**: Use a dictionary to store the the keys pressed and their values of angle. That the program can easily call the direction with the keyboard and control the angel and the direction of the turtle objects.

```
direction = {K_UP: 90, K_DOWN: 270, K_LEFT: 180, K_RIGHT: 0}
# dictionary stores the keypress and their values of angle
```

ii. List:

1. **L_snake**: Append the current position of head of snake when moving, which represents the following tail's position.

```
L_snake = []
L_snake.append(pos_head) # append the position of body into a list
```

iii. String:

1. **g_keypressed**: The string g_keypressed denotes the current moving status of moving, and changed as the users pressing the arrow key, also it denotes whether the snake is paused or moving.

```
g_keypressed = ''
g_keypressed = key # show the current moving status and direction
if gamePause == True: g_keypressed = 'Pause' # show when paused
```

2. **intro**: Show the introduction before starting the game, inform the users how to control and play, how to win the game and how would they fail the game.

```
intro = '''Welcome to 120090272's version of 'Snaker Game'!
... (omit the details of intro here)
'''
```

iv. Variable:

1. **Time**: Show the total time after the game started, in unit of seconds as integer, using ontimer to control the increasing and update of time.

```
Time = 0 # before start show the time as 0
def time_update():
    ...
    Time += 1 # show in integer
    s.ontimer(time_update, 1000) # 1 second per update
```

2. **cnt_contact**: Calculate the contact of times that the monster contact with the snake body, each time the monster contact with the snake body in second unit, the contact counter will plus one. And it will be showed in the status area.

```
cnt_contact = 0
if distance(pos, b) < 20:
    cnt_contact += 1
```

3. **g_length**: Keep the current length of snake. After consuming the food items, the g_length will add up the corresponding number. Also, the g_length will have a initial value of 5.

```
g_length = 5
def food_consume(pos_1, pos_2, num):
    ...
    g_length += num
```

v. Tuple:

1. **pos_head**: The tuple indicates the coordinate of snake head. It is global for representing the position of snake.

```
pos_head = (0, 0)
pos_head = tt.heading()
```

2. **pos_monster**: The tuple indicates the coordinate of monster. It is global for representing the position of monster.

```
pos_monster = (0, 0)
pos_monster = monster.heading()
```

vi. Boolean:

1. **gameStart, gamePause, gameOver**: The boolean types are used for control the state of game. Such as whether the game is paused, started, over, win.

```
gameStart = False # means the game hasn't been started
gamePause = False # means the game is paused
gameOver = False # means the game is over(fail or win)
```

c. Program Structure

1. Initialize the screen

`generate_turtle()` is used first to generate several turtles and make them all penup and speed(0). The turtles are `tt` to represent the snake head, `monster` to represent the monster, `food` as the food items, `g_time` as the time keeper, `g_status` as the status keeper, `g_contact` as the contact keeper, `g_intro` to write the introduction, `upper_a` as the upper status area, `motion_a` as the motion area.

`init_state()` set the color, shape, size, position of the turtles in need. Also, write the information in the places that they should be.

2. Set the movement of snake

`snake_update()` is defined which contains both the forward move and extension ability of snake body. The function will judge whether the game win or game over to make sure whether to start the snake's move. Also the function will realize the extension of snake's body by the limit of `g_length` and `L_snake`. Also use the `ontimer` to control the speed of snake. When the snake eat the food and extend, the speed of snake will be a little slower than normal. Also the function ensures that the snake move in the motion area. And if crush on the boundary, the snake will stop at and wait until users change the direction of moving.

3. Set the movement of monster

`monster_place()` before the game start, the program should randomly put the monster on the screen first with the function. This function will put the monster a little far from the initial position of the snake to make sure that the users won't fail the game easily.

`monster_dir` is used to set the direction of monster. The function make sure that the monster will move in the shortest way to approach the snake. Also, the function can limit the monster move in the motion area.

`monster_update()` is defined to move the monster forward with a random speed. The random speed sometimes slower than the snake while sometime a little faster than the snack especially when the snake is extending.

4. Set the way of food consumption

`get_number()` before the game start. The numbers 1 to 9 of food items will be placed in the random place on the screen. The function make sure that the food items will have a distance with each other, and they will have a distance with the monster and the snake. Also, make sure that the snake can eat the food head on head.

`food_consume()` defines the way of consuming food items. Get the position of food item and the position of snake. If the distance of snake and food is smaller than 20, which means they contact with each other on the screen, the snake eat the food number, and the snake's body will extend the same length as the number on food.

5. Judge the game status

`game_start()` to start the game if the function is called.

`game_over()` to judge whether the game is over. If the game over, it stops the program.

`game_win()` to judge whether the game is win. If the game win, it stops the program.

`game_pause()` to pause the game process when needed. Also restart the game when needed.

6. Update the status of time, contact and status in the upper area

`time_update()` The function update the time and show it in upper status area with an interval of 1 second, each second pass, the number of time will change in the upper status area. After the game start, the time will increase until game over or game win.

`status_update()` The function update the status of snake and show it in the upper status area when the state changes or when the key pressed.

`contact_update()` The function update the contact of times that the monster contact with the snake per unit second.

7. Set the onclick and keypress events to start the game loop.

`game_on()` The game on function will be invoke when click on the screen, it clean the introduction words on the screen, and make the game start, put the numbers of food items on screen. Then jump into the game loop

`game_loop()` If the game is start and before the game over or win, the game can be played by the functions above.

`main()` The main process of the program. Control the screen by keyboard, start the game, and update the status.

8. Some other tool functions used for the game

`is_bounded(Object)` The object here should be a turtle, the function is used to judge whether the object is in the required bounded range.

`distance(a, b)` The function is used to calculate the distance of two tuples a and b.

`arrow_key(key)` The function is used for control of keyboard and direction of snake move.

d. Processing Logic

i. The logic used to motion the snake and monster.

Use the `turtle.forward()` function to move forward the turtles of snake and monster. Use arrow key on the keyboard to control the direction of snake, set up an algorithm to control and calculate the direction of monster. Use the `turtle.setheading()` function and relate the directions with angles to control the turtles' moving condition.

ii. The logic used to expand the snake tail.

Use `turtle.stamp()` to stamp the path of the snake's move. Also, use the `turtle.stampItems` to calculate and use the function `turtle.clearstamp()` limit the snake's length with `g_length`. What's more, judge the `number` of food items and add it up on `g_length` to control the increase of snake.

iii. The logic used to detect body contact between the snake and the monster.

A tool function called `distance(a, b)` is designed to calculate the tuples of two objects. We know the defaulted size of the snake square is 20*20, and each time they move forward 20 as a step. So if the distance is smaller than 20, two bodies contact with each other.

2. Function Specification

(Notice that the design ideas are in the code block)

(The usage of all my own defined functions' details are in the 'Program Structure' part)

1. `generate_turtle()` :

```
def generate_turtle(): # use the function to generate turtle
    turt = Turtle()
    turt.penup()
    turt.speed(0)
    turt.hideturtle()
    return turt # return the turtles that are generated
```

2. `init_state()`: Just some simple process to generate and set the status of turtles, so here omit it in the document.

3. `snake_update()` :

```
def snake_update():
    global L_snake, pos_head, pos_food, gamePause, gameOver, gameStart,
g_keypressed, g_length
    t = 300
    game_over() # judge whether the game over
    game_win() # judge whether the game win
    if gameStart == True and is_bounded(tt) == True and gameOver == False and
gamePause == False:
        tt.color('blue', 'black') # change the turtle color the same as body
        tt.stamp() # make a stamp of turtle as body
        pos_head = tt.position() # note the position of turtle now
        tt.forward(20) # move forward the snake
        tt.color('blue', 'pink') # change the turtle color back the same as head
        L_snake.append(pos_head) # append the position of body into a list
        if len(L_snake) > g_length:
            L_snake.pop(0) # keep the length of snake in control
        if len(tt.stampItems) > g_length:
```

```
            tt.clearstamps(1)
        elif len(tt.stampItems) < g_length:
            t = 350 # slower the speed of snake when it is expanding
        for number, position in list(pos_food.items()): # consume the food when
needed
            food_consume(tt.position(), position, number)
    status_update() # update the state of current snake
    s.ontimer(snake_update, t) # set an ontimer to control the speed of snake
    s.update() # update the screen
```

4. `monster_place()`:

```
def monster_place(): # randomly place the monster and make sure their positions
can
    global pos_monster
    monster.color('skyblue')
    monster.showturtle()
    while True:
        pos_mons = (random.randint(-240, 240), random.randint(-280, 200)) #
randomly choose number as coordinates in range
        # make sure that the monster can contact with snake in corner and has an
initial distance with it
        if pos_mons[0] % 20 == 10 and pos_mons[1] % 20 == 10 and
distance(pos_mons, pos_head) > 80:
            pos_monster = pos_mons
            monster.setpos(pos_monster)
            break
```

5. `monster_dir`:

```
def monster_dir():  # set the direction of monster
    dx, dy = pos_head[0] - pos_monster[0], pos_head[1] - pos_monster[1]
    if dx > 0:
        if dx > abs(dy): monster.setheading(0)  # go right
        elif dy > 0: monster.setheading(90) # go up
        else: monster.setheading(270)
    elif dx < 0:
        if abs(dx) > dy: monster.setheading(180) # go left
        elif dy > 0: monster.setheading(90) # go up
        else: monster.setheading(270)
    next_step = generate_turtle()
    next_step.goto(pos_monster[0]-20, pos_monster[1])
    if is_bounded(next_step) == False:
        monster.setheading(tt.heading())
```

6. `monster_update()`:

```
def monster_update(): # function to control and update the state of monster
    global pos_monster
    t = random.randint(330, 400)
    if gameStart == True and gameOver == False:
            pos_monster = monster.position() # get the position of monster
            monster_dir() # set the direction of monster
            monster.forward(20) # move the monster forward
            s.update() # update the screen
            s.ontimer(monster_update, t) # set a ontimer to control the speed of
monster
```

7. `get_number()`:

```
def get_number(): # get the food items in number on the screen
    global pos_food, food, number
    radius = 60 # make sure that the food has distance with each other
    deltas = set()
    for x in range(-radius, radius+1):
        for y in range(-radius, radius+1):
            if x*x + y*y <= radius*radius:
                deltas.add((x, y))
    excluded = set()
    number = 0
    while number <= 8:
        x, y = random.randint(-230, 230), random.randint(-260, 180) # randomly
choose number to denote the coordinates of position
        if x % 20 == 0 and y % 20 == 10 and (x, y) not in excluded: # make sure
the snake can eat the food in directly direction
            pos_food[number] = (x, y)
            number += 1
            excluded.update((x+dx, y+dy) for (dx, dy) in deltas)
            food.goto(x, y) # put the food in the certain place
            food.write(number, False, 'center', font=('Arial', 12, 'normal')) #
write the number on food
        else:
            continue
```

8. `food_consume()`:

```
def food_consume(pos_1, pos_2, num): # define the way of consuming food items
    global g_length, pos_food, food
    if distance(pos_1, pos_2) < 20: # eat when the distance of food and snake
head is smaller than 20
        del pos_food[num] # delete the number in dictionary
        food.clear() # clear the food item that has been consumed
        for value, position in list(pos_food.items()): # write the rest of food
items
            food.goto(position)
            food.write(value+1, False, 'center', font=('Arial', 12, 'normal'))
            s.update()
        g_length += num # increse the length of snake when eat food
```

9. `game_start()`:

```
def game_start(): # start the game
    global gameStart # global the state the gameStart
    if pos_food != {}:
        arrow_key('Right')
        gameStart = True
```

10. `game_over()` :

```
def game_over(): # judge whether the game is over
    global gamePause, gameOver
    if distance(pos_head, pos_monster) < 20 and gamePause == False and gameStart
== True: # if distance of head and monster smaller than 20
        gamePause = True
        gameOver = True
        tt.write('Game Over!!', False, 'right', font=('Arial', 12, 'normal')) #
write the information of game over
        s.update()
```

11. `game_win()` :

```
def game_win(): # judge whether the game is win
    global gamePause, gameOver
    if pos_food == {}: # if all food items are consumed, the game win
        gamePause = True
        gameOver = True
        tt.write('Winner!!!', False, 'right', font=('Arial', 12, 'normal')) #
write the information of game win
```

12. `game_pause()` :

```
def game_pause(): # pause the game process when needed
    global gamePause, g_keypressed
    gamePause = not gamePause
    if gamePause == True:
        g_keypressed = 'Pause'
    elif gamePause == False: # use arrow key to restart the game after being
paused
        a = int(tt.heading())
        b = list(direction.keys())[list(direction.values()).index(a)]
        arrow_key(b)
```

13. `time_update()` :

```
def time_update(): # update the time and show it in upper status area with an
interval of 1 second
    global Time, g_time
    if gameStart == True and gameOver == False:
        g_time.clear()
        g_time.write('Time:{}'.format(Time), False, 'center', font=('Arial', 14,
'normal'))
        s.update()
        Time += 1 # time plus 1 when 1 second after
    s.ontimer(time_update, 1000)
```

14. `status_update()` :

```
def status_update(): # update the status of snake and show it when the state
changes
    global gamePause, state, gameStart, g_keypressed
    g_status.clear()
    g_status.write('Status:{}'.format(g_keypressed), False, 'center', font=
('Arial', 14, 'normal'))
    s.update()
```

15. `contact_update()` :

```
def contact_update(): # update the contact of times that the monster contact
with the snake body
    global cnt_contact
    b = monster.position() # get the tuple of position of monster
    if gamePause == False and gameStart == True:
        for pos in L_snake:
            if distance(pos, b) < 20:
                cnt_contact += 1
        g_contact.clear()
        g_contact.write('Contact:{}'.format(cnt_contact), False, 'center', font=
('Arial', 14, 'normal'))
        s.update()
    s.ontimer(contact_update, 1000) # update by the time of 1 second
```

16. `game_on()` :

```
def game_on(x, y): # click to invoke the game_on function to start the game
    global g_intro, pos_food, gameStart
    s.onscreenclick(None) # close the onclick of screen
    s.tracer(0)
    g_intro.clear()
    get_number() # get numbers of food items on the screen
    game_start() # start the game
    game_loop()
```

17. `game_loop()` :

```
def game_loop(): # define the loop of the game
    if gameStart == True and gameOver == False:
        snake_update()
        monster_update()
```

18. `main()` :

```
def main(): # main process of the code
    s.listen() # ask the screen to listen
    s.onkey(partial(arrow_key, 'Up'), 'Up') # onkey arrow keys to control the
move of snake
    s.onkey(partial(arrow_key, 'Down'), 'Down')
    s.onkey(partial(arrow_key, 'Left'), 'Left')
    s.onkey(partial(arrow_key, 'Right'), 'Right')
```

```
    s.onkey(game_pause, 'space') # user can use space key to pause or restart
the game
    if gameStart == False:
        init_state() # initialize the original state and show the introduction
        s.tracer(0)
        s.onscreenclick(game_on) # click the screen to start the game
    time_update() # update the game
    contact_update() # update the number of contact
    s.mainloop() # close the game
```

19. `is_bounded(Object)` :

```
def is_bounded(Object): # judge whether the object is in bounded range
    global gamePause, g_keypressed
    x, y = Object.position()
    if -240 <= x <= 240 and -280 <= y <= 200:
        if x >= 235 and Object.heading() == 0: return False
        elif x < -235 and Object.heading() == 180: return False
        elif y >= 195 and Object.heading() == 90: return False
        elif y <= -275 and Object.heading() == 270: return False
        else: return True
    else: return False
```

20. `distance(a, b)` :

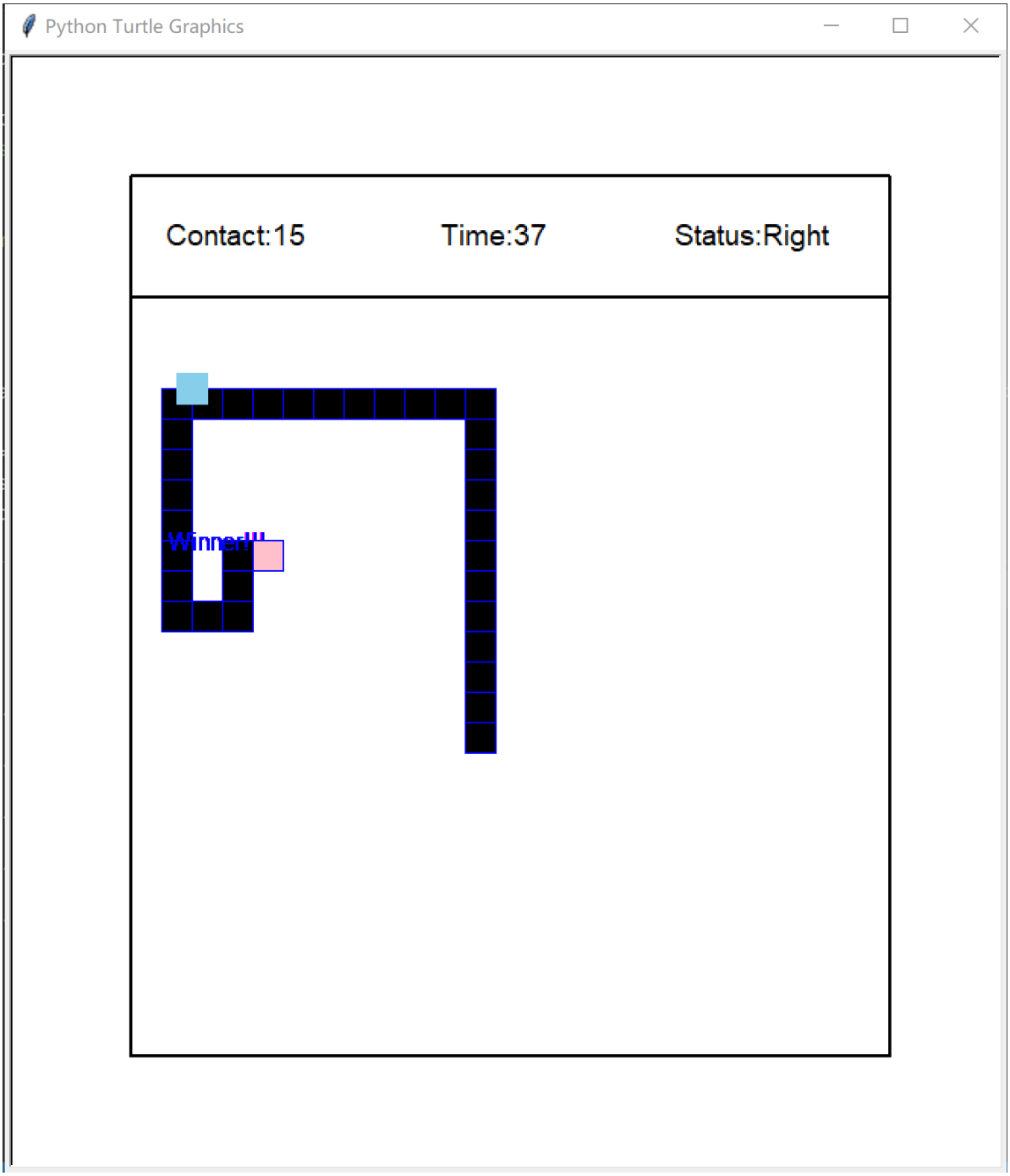
```
def distance(a, b): # calculate the absolute distance of two tuples
    dx = abs(a[0] - b[0])
    dy = abs(a[1] - b[1])
    return (dx*dx+dy*dy)**0.5
```

21. `arrow_key(key)` :

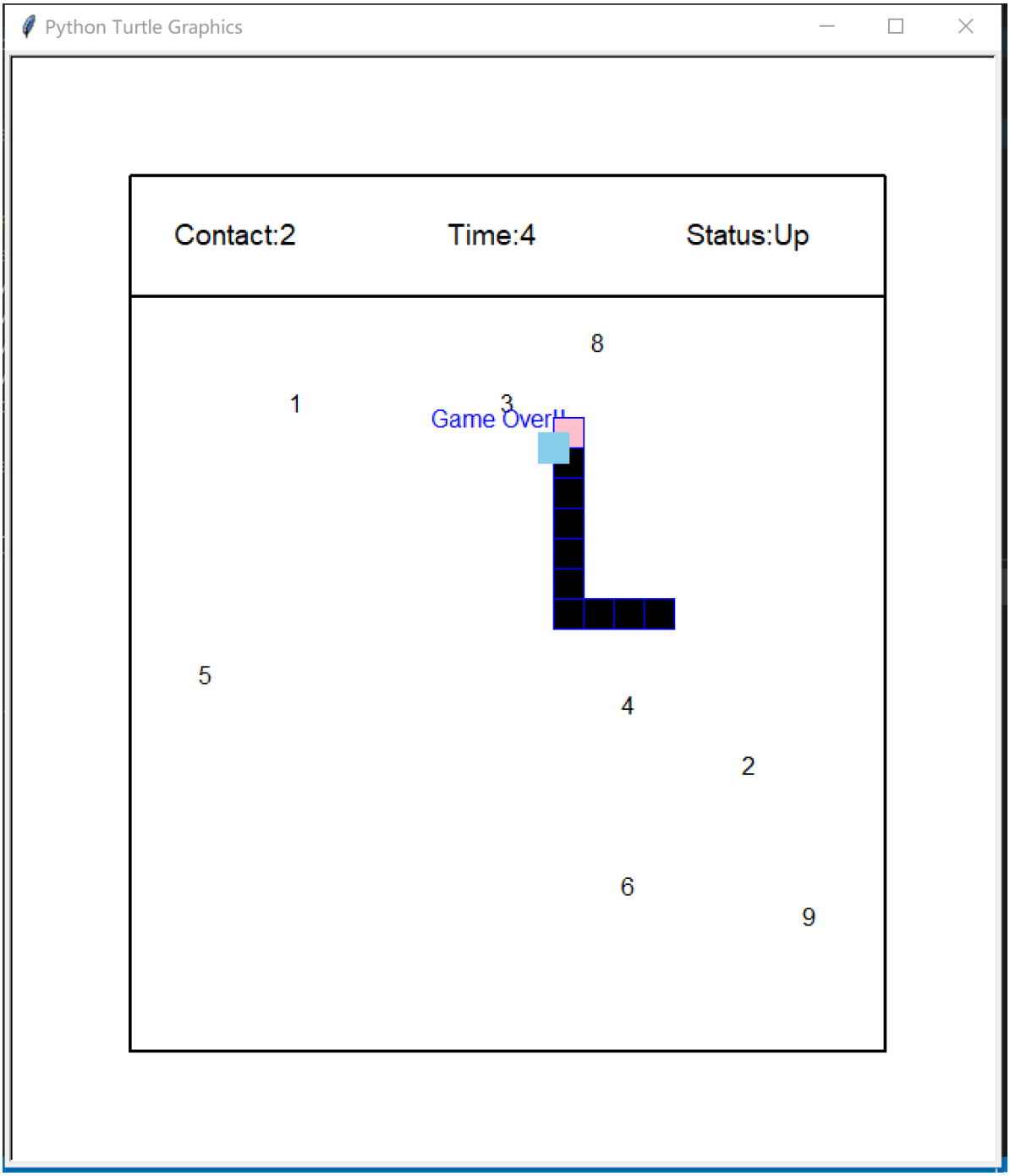
```
def arrow_key(key): # use arrow key to control the direction of snake
    global g_keypressed, gamePause
    g_keypressed = key # denote the current direction when keypressed
    if key in direction.keys():
        tt.setheading(direction[key])
    status_update() # update the state
    if gamePause == True and g_keypressed != 'Paused':
        gamePause = False
```

3. Sample Output

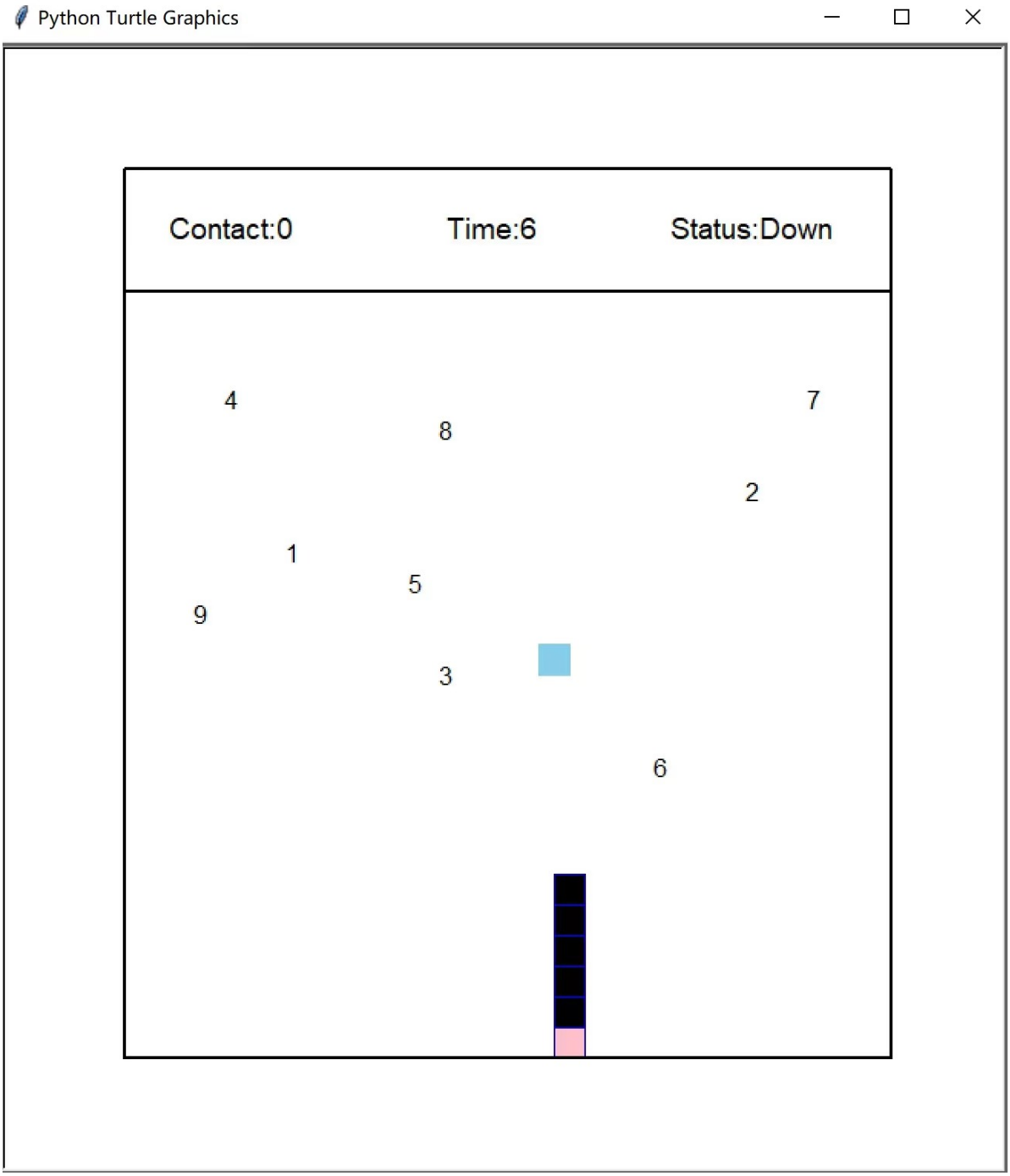
1. **Winner:**


Python Turtle Graphics
Contact:15
Time:37
Status:Right
Winner!!!

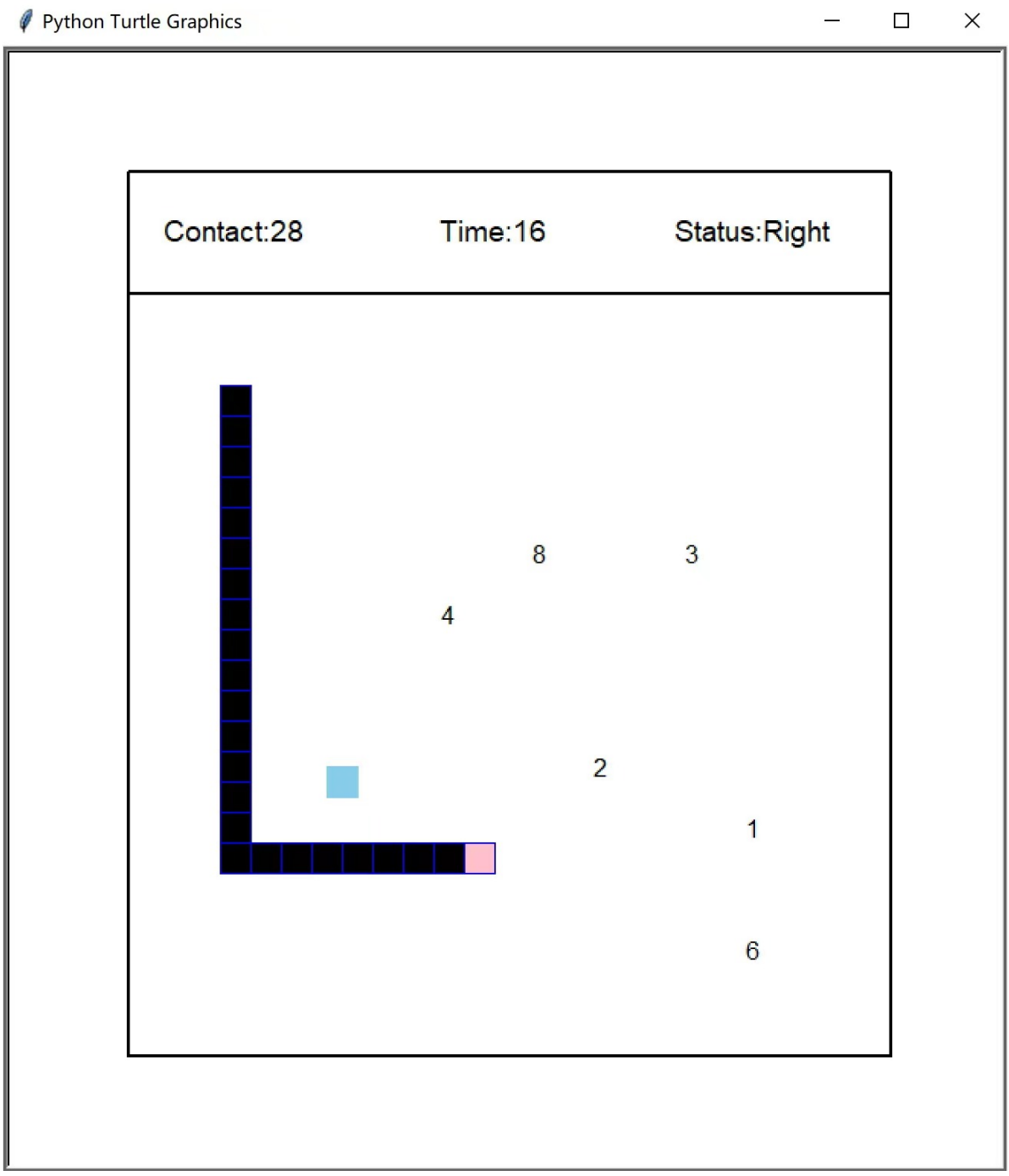
2. **Game over:**



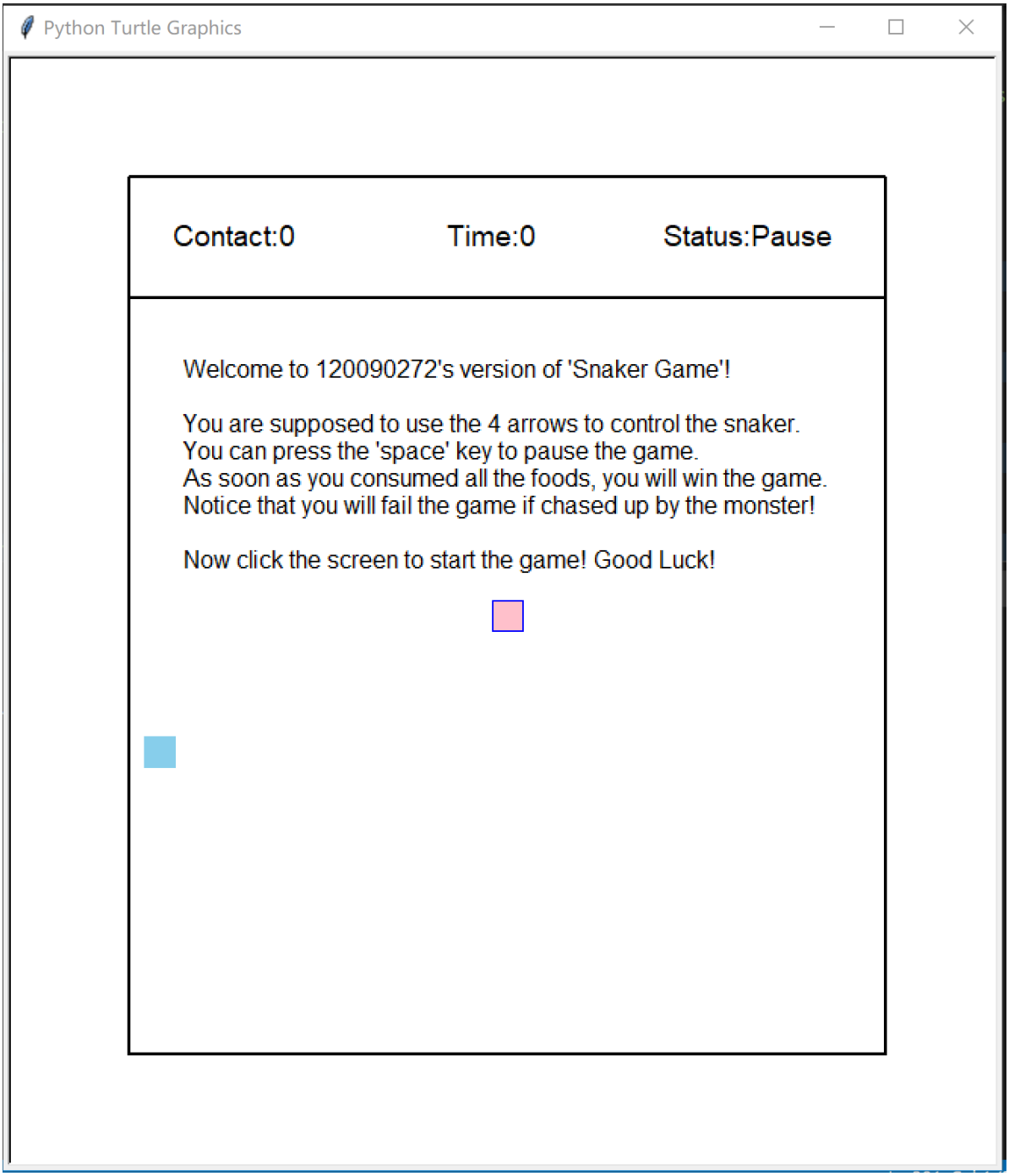
3. **With 0 food item consumed:**



4. **With 3 food item consumed:**



5. **Initial state:**


Python Turtle Graphics
Contact:0
Time:0
Status:Pause
Welcome to 120090272's version of 'Snaker Game'!
You are supposed to use the 4 arrows to control the snaker.
You can press the 'space' key to pause the game.
As soon as you consumed all the foods, you will win the game.
Notice that you will fail the game if chased up by the monster!
Now click the screen to start the game! Good Luck!