# Design

**My thoughts and overall approach (Basically the structure of my program)**

1. Configure the game screen. Bind the four arrow keys with the move direction. Let the user to use the four arrow keys to maneuver the snake, and use the space bar to pause the snake motion.
2. Configure a pen to write the introduction texts.
3. Write the introduction content.
4. Configure the snake. Set the initial length of the tail to be 5. Set the snake head.
5. Configure the monster. On startup, place the monster on a random position with a fair distance from the snake.
6. Call listen to activate the screen event process.
7. Game startup. The "onclick" function calls should have 2 parameters x, y. The two parameters represent the locations where the mouse clicks.
8. Then hide the introduction texts and start the game.
9. Set the food items. Represent them as numbers from 1 to 9. Display the food within the game area in random. Ensure the food items will not overlap with each other.
10. Set the monster direction. Divide the screen area into 4 parts. Compute the angle the snake towards the monster. Check which part the angle points to. Then determine the monster motion direction.
11. Move the monster. Calculate the monster and snake body collisions. Set the monster velocity to be a suitable value. Determine whether the user loses the game. If not, then continue move. If yes, then the monster stops motion.
12. Move the snake head. Check how many times the snake still needs to decelerate. If no need to decelerate again, then back to fast velocity. Because previously when the snake extends the length, the tail sticks. So now pop the tail location then it can move forward.
13. Move the snake body.
14. Let the snake consume the food. Let the snake extend as the value of the food item number. As the tail is being extended, the movement of the snake will slow down.
15. Set the screen title of the screen. Keep track of the total elapsed time and monster-snake collisions. Call this function itself every 1000 milliseconds (1 second). That is to keep track of the total elapsed game time in seconds.

**Describe the date types I use to develop my program for tracking the various game objects ( The following are all the global variables that I use in my program)**

1. g_food = [ ]: Used to save the food items.
2. g_eaten_food = [ ]: The food items that have been consumed by the snake.
3. g_body_id = [ ]: The stamp ID of the snake body.
4. g_decelerate_time = 4: The time that the snake needs to decelerate. Its initially default value is 4.
5. g_track = [(0,0)]: Save all the locations that the snake head has passed
6. g_snake_body = None: The last block of the snake body. Equally the snake "tail" block.
7. g_game_screen = None: Set the game area.
8. g_snake_head = None: Set the snake head.
9. g_monster = None: Set the monster component.
10. g_pen = None: Set the pen to write texts.
11. g_elapsed_time = 0: The elapsed time of the game.
12. g_contact_monster = 0: The times that the monster collides the snake body part.

13. g_time_rate = 500: The motion time rate (velocity). Its initially default balue is 500.
14. g_end = False: Whether the game is over.

**Describe the motion logic for both snake and monster**

A. The motion logic for snake:
   1. Move the snake head: Let the head moves forward for one unit. The original location of snake head becomes the body part.
   2. Move the snake body: Let the snake tail block go to snake head location. Then delete the last stamp (the original block before the original tail block). Create a new stamp to save the location of the snake tail now and save the stamp ID of it. Then delete the last stamp ID. Then move the snake tail back to the end.
      ## Later when the snake eats the food, the snake should create blocks equal to the number at the snake tail location. Then we need the tail to do stamp at the end. So every motion of the body, I will let the tail go back to the end.

B. The motion logic for monster:
   1. First determine the monster direction: Divide the screen area into 4 parts. Compute the angle snake towards the monster. Check which part the angle is in. Then determine the monster motion direction.
   2. Then move the monster. Set the monster velocity to be a suitable random value.

**Describe the expansion logic for the snake tail**

The original location of snake head becomes the body part. If the snake doesn't eat food, then clear the last block (the tail). If the snake eats the food and is extending the body length, then we don't clear the last block. And we don't move the blocks between head and tail.

**Describe the body contact logic between the monster and the snake**

Save the track of the snake and update it. If the distance between the monster and any item of the track list is equal or less than 20, then we consider the monster contacts the snake. Then we plus the contact time by one.

# Functions

**Describe the usage of all my newly defined functions, including details of parameters.**

1. **configureScreen:**
   Parameters: w = 500, h = 500 (set the default width and height of the screen area)
   (1) Set the game area. Define the game area as 500 × 500 in dimension.
   (2) Use Screen() to configure the game area.
   (3) Set the background color to be white.
   (4) Use tracer(0) to disable auto screen fresh.
   (5) Use the four arrow keys to maneuver the snake. Use space bar to pause the snake motion.
2. **configureTurtle:**

Parameters: instance = None, shape = “square”, fill_color = “”, border_color = “”, x = 0, y = 0

(1) Define a turtle module to make other turtle objects
(2) Set the speed to be 0
(3) Set the shape, color and coordinate of the turtle.
(4) Use penup() to avoid draw unnecessary lines on the screen area.

3. **configureSnake:**
   No parameters.
   (1) Set the initial snake component.
   (2) Set the initial length of the snake body to be 5. The tail of length 5 contains the snake tail block. So we just set it to be 4. Later when the snake consumes food, the extended length will exclude snake tail. Use stamp() to save a copy of the turtle shape at the current position.
   (3) Set the snake head.
   (4) Manually refresh the game area.
4. **configureMonster:**
   No parameters.
   (1) Set the monster component.
   (2) On startup, place the monster on a random position with a fair distance from the snake.
   (3) Manually refresh the game area.
5. **configurePen:**
   No parameters.
   (1) Define a pen to write the introduction texts.
   (2) Set the speed to be zero.
   (3) The pen color is black.
   (4) Use penup() to avoid the pen to draw unnecessary lines on the screen area.
   (5) Use hideturtle() to hide the turtle.
6. **snake_food:**
   No parameters.
   (1) Set the food items. Represent them as numbers from 1 to 9.
   (2) Display them within the game area in random.
   (3) Ensure the food items will not overlap with each other.
   (4) Manually refresh the game area.
7. **monster_direction:**
   No parameters.
   (1) Set the monster direction.
   (2) Divide the screen area into 4 parts. Compute the angle snake towards the monster.
   (3) Check which part the angle points to. Then determine the monster motion direction.
8. **screen_title:**
   No parameters.
   (1) If the game isn’t over, then plus the elapsed time by one.
   (2) Call the function itself every 1000 milliseconds (1 second).
   (3) That is to keep track of the total elapsed game time in second.
9. **introduction:**
   No parameters.
   (1) Set the introduction content.
   (2) Use the pen to write the content.
10. **touch_screen_boundary:**
    Parameters: part (the part that we want to check whether it touches the boundary)
    (1) Check whether the monster or snake reach the boundary.

(2) If they reach the boundary, then stop motion.

11. **determine_win:**
    No parameters.
    (1) Check whether the user wins the game.
    (2) If the number of food that the snake has eaten is 9, then the user wins the game.
    (3) Show “win” texts on the screen.
    (4) Block all the key to be pause.
    (5) The end status is true.
12. **determine_lose:**
    No parameters.
    (1) Check whether the user loses the game.
    (2) If the monster touches the snake head, then the user loses the game.
    (3) Block all the key to be pause.
    (4) The end status is true.
13. **show_win:**
    No parameters.
    (1) Show on the screen to inform the user of the success.
    (2) Make the direction of the snake head and the monster to be stop.
    (3) Use red color pen and go to origin to write the texts.
    (4) Hide the turtle.
14. **show_lose:**
    No parameters.
    (1) Show on the screen to inform the user of the failure.
    (2) Make the direction of the snake head and the monster to be stop.
    (3) Use red color pen and go to origin to write the texts.
    (4) Hide the turtle.
15. **go_up:**
    No parameters.
    Set the direction of the snake head to be up.
16. **go_down:**
    No parameters.
    Set the direction of the snake head to be down.
17. **go_left:**
    No parameters.
    Set the direction of the snake head to be left.
18. **go_right:**
    No parameters.
    Set the direction of the snake head to be right.
19. **pause:**
    No parameters.
    Set the direction of the snake head to be stop.
20. **move_part:**
    Parameters: part (the part that we need to move).
    One unit move, distance equals to 20 every time. Manually update the screen area.
21. **snake_body_move:**
    No parameters.
    (1) Let the snake tail block go to snake head location.
    (2) Delete the last stamp (the original block before the original tail block).
    (3) Create a new stamp to save the location of the snake tail now and save the stamp ID of it.
    (4) Delete the last stamp ID.

(5) Move the snake tail back to the end.
(6) Later when the snake eats the food, the snake should create blocks equal to the number at the snake tail location. Then we need the tail to do stamp at the end. So every motion of the body, I will let the tail go back to the end.

**22. Snake_head_move:**

No parameters.

(1) Check whether the snake head touches the screen boundary.
(2) If it doesn't touch the screen, then continue.
(3) Move the snake body and the snake head. Determine whether the user wins the game.
(4) Save the position of the snake head into track list.
(5) Let the snake eat the food.
(6) The snake needs to decelerate when it is eating food. Check how many times the snake still needs to decelerate.
(7) If the time doesn't equal to zero, that means the snake still needs to decelerate. Then it minus decelerate time by one every move.
(8) If the time equals to zero, then the snake doesn't need to decelerate again. Then change the time rate back to fast one.
(9) Because previously the tail sticks to the end. Now pop it then it can move forward.
(10) Use ontimer to call this function after a fixed delay.

**23. eat_food:**

No parameters.

(1) If the distance between the snake head and the food item is less than 20, then it means that the snake has eaten the food.
(2) After the snake eats the food, we should clear the food.
(3) But after clearing the food item, there still exists a turtle. So we should also move the turtle far far away.
(4) Append the food item to eaten food list.
(5) Use stamp() to extend the snake body as the value of the food item number.
(6) As the tail is being extended, the movement of the snake will slow down. The decelerate time is equal to the food item number.

**24. monster_move:**

No parameters.

(1) If the game isn't over yet, then move the monster.
(2) Calculate the monster-snake body collisions.
(3) Set the monster velocity to be a suitable value.
(4) Use ontimer to call this function after a fixed delay.

**25. start_game()**

Parameters: x,y (the function onclick calls should have 2 parameters. x, y represent the location where the mouse clicks.)

(1) User onclick(None) to unbind the onclick and start_game function. Then next time we click the mouse, it won't start again.
(2) Clear the introduction contents and begin the game.

**26. main:**

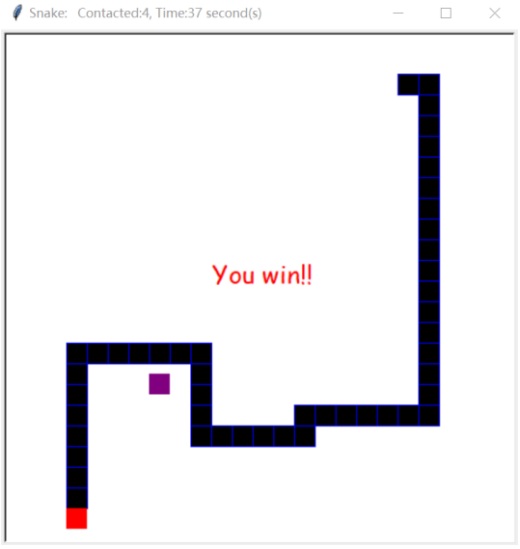
No parameters.

Main function. The onclick function of the screen objects binds a function to a mouse-click event. Call listen() to activate the screen event process. Place listen() before mainloop().

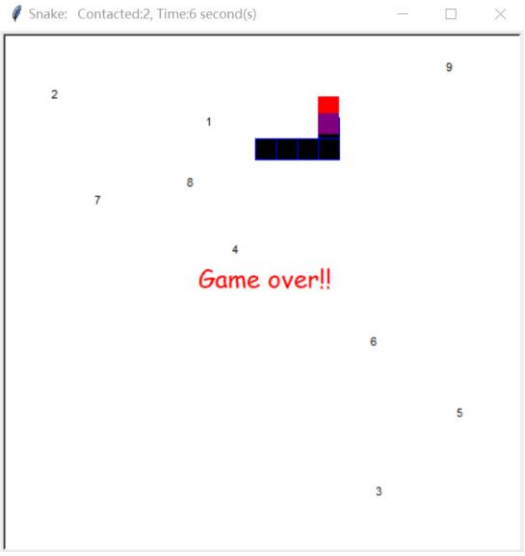
# Output

**Show samples of output (including status) from my program.**

**(1) Winner**

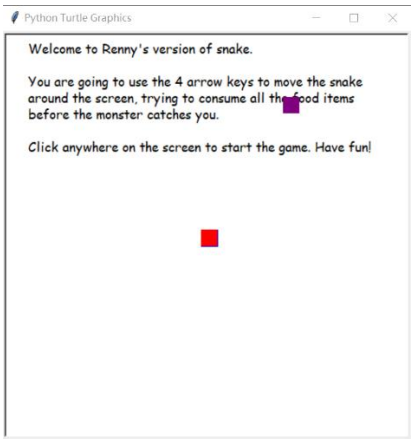


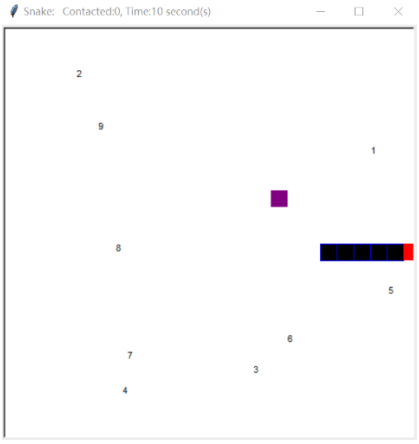

**(2) Game over**



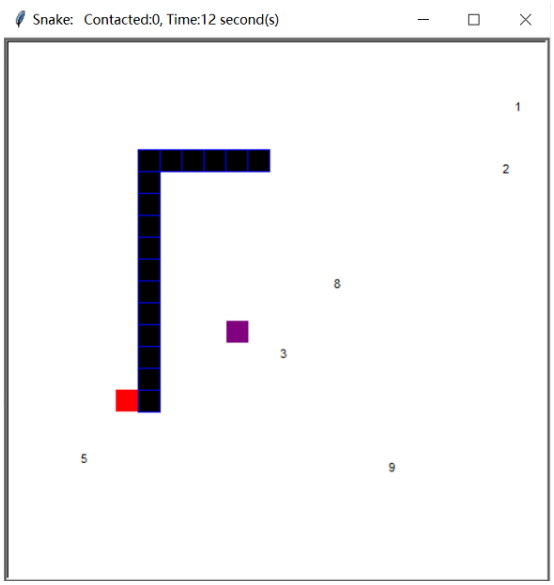

**(3) 2 others showing various stages of the game:**

**(i) With 0 food item consumed**




**(ii) With 3 food items consumed**


Snake: Contacted:0, Time:12 second(s)
1
2
8
3
5
9