Design Doc

Overview

The program is written for running a Snake game, which contains 3 objects: snake, monster and food items represented by a set of numbers from 1 to 9. The head of the snake is represented by a red square while the tail of the snake is represented by a set of black squares with blue borders, and the monster is represented by a purple square.

The goal of the game is to move the snake within the motion area in four directions (right, up, left, down) trying to consume all the food items and avoiding head-on collision with the monster. The monster is programmed to try to catch the snake, it will also move in four directions (right, up, left, down) towards the head of the snake at a variable speed. While the head of the snake cross any food item, the food item will be consumed by the snake, and the snake's tail will extend in size equal to the value of the number being passed.

The Snake game begins with an introduction displayed on the screen, after a mouse click on the screen, the game will start 0.5 second later. If the monster makes a head-on collision with the snake, the player will fail. If the snake consumes all the food items and hasn't been caught before fully extend its body, the player will win the game. During the game, the number of contacts with the monster and the body of the snake, total elapsed game time, and the motion of the snake will be shown on the screen, and updated once a while.

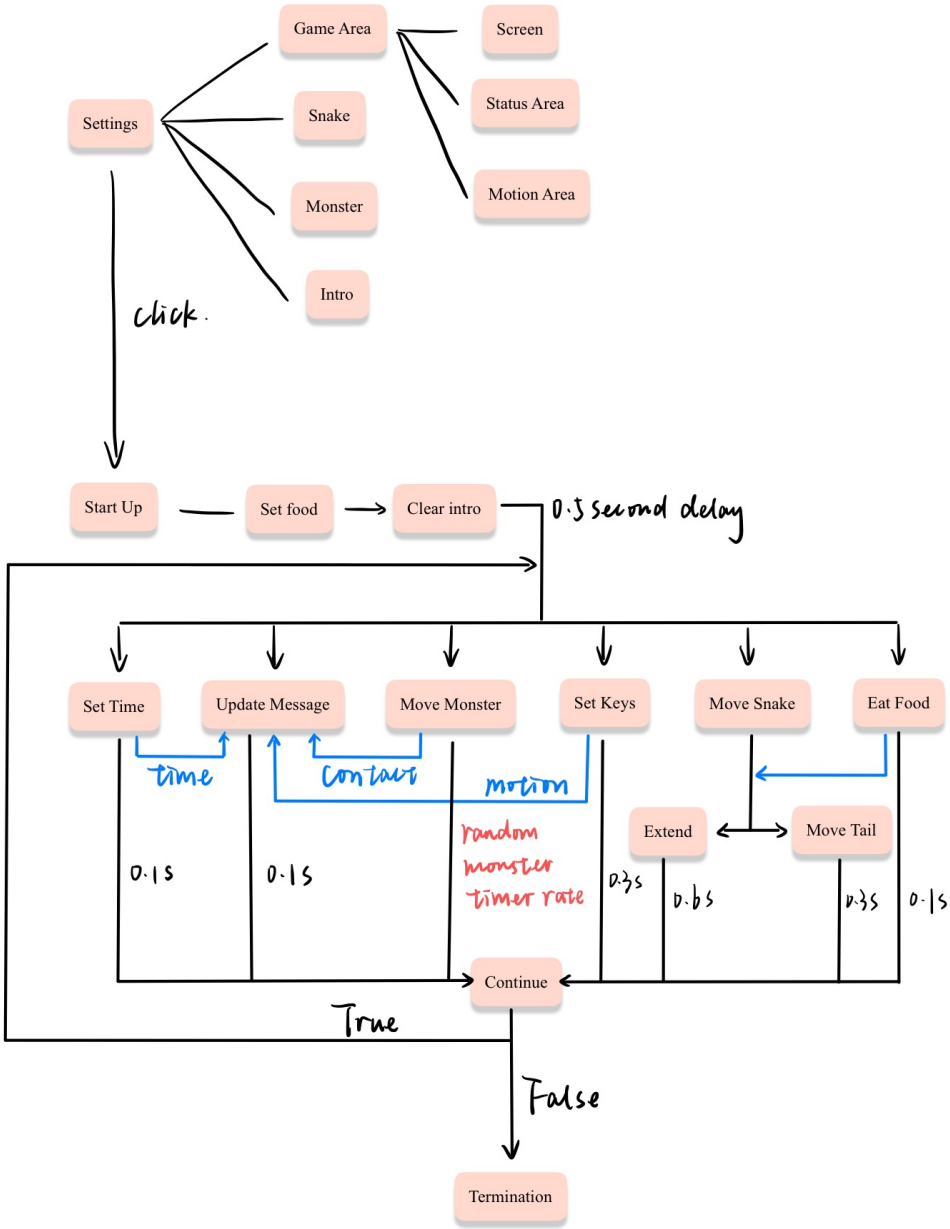
Data Model

- **Snake (head)**: Turtle object. With red color and square shape, initially at the center of the motion area, which has the coordinate (0,-40).
- **Snake (tail):** stamps of the head of snake. Two global variable **g_tailPositions** and **g_tailLength** describe its information. g_tailPositions is a list recording all the stamps' current positions, every time the snake moves or extends, the list will update to record its new positions. g_tailLength is an integer initially to be 5, representing the length of the tail should be. For example, when the snake consumes a food item and hasn't finished extending, the length of the snake tail will be larger than the length of the list tail positions.
- **Monster:** Turtle object. With purple color and square shape, initially at a random position which the distance between monster and snake is more than 200 pixels.
- **Food items:** a list of 9 turtle objects. Each of the element inside the **g_food** list is a turtle object represented a food item. They have random positions inside the motion area (cannot coincide with the head of the snake at the beginning of the game). And the turtle object is set to be hidden. Only

the number it writes on the screen can be seen. After being consumed, the turtle object will be transformed into an empty string ‘’.

Program Structure

At first, set up all data model, including game area (screen, status area, motion area), snake, monster, and intro message. After the mouse click on screen, start up the game. First set up food items, and then clear the intro message. After 0.5s delay, go to the main loop component (set time, update message, move monster, set keys, move snake (extend or move tail), eat food). If the game should not continue, stop calling all the 6 function and go to the termination.

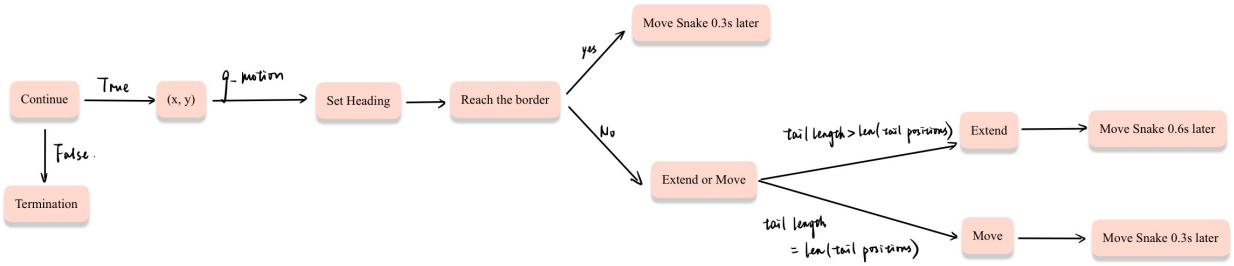


Functional components: settings, start up, main loop, termination.

- Settings: set up all data model, including game area (screen, status area, motion area), snake, monster, and intro message.
- Start up: first set food items, then clear intro.
- Main loop: call set up game time function every 0.1s to update the elapsed time of the game; call update message function every 0.1s to update time, contact, motion message on the screen; call move monster function to move the monster in random timer rate between every 0.4s and 0.7s; call set keys function every 0.3s to update the input keys and bind them to the global motion variable; call move snake function to extend the snake or move the snake, timer rate for extending is 0.6s and the timer rate for moving is 0.3s; call eat food function to check if the snake has eaten any food and update the length of snake body the snake should have.
- Termination: if monster catches up the snake, game over. If snake consumes all food items and finishes extending its body, the player wins the game.

Processing Logic

- Motion the snake: if the game is going to continue, first get the coordinate of the snake. Then set the heading of the snake according to the global variable **g_motion**. Check if the snake has reached the border or not, if it has reached the border, stop moving the snake forward, if not decide whether to extend the snake or move the snake. If the length of the tail at current positions is smaller than the global variable **g_tailLength**, extend the snake, else, move the tail. After extending, move snake after 0.6s, while after moving tail, move snake after 0.3s. if the game is not going to continue, go to the termination.



- Motion the monster: if the game is going to continue, first get coordinates of both the monster and the snake. Then compare the absolute value of the difference of two x-coordinates and the difference of two y-coordinates. In order to move closer to the snake, the monster should move in the direction which the difference is larger. If absolute value of the difference of two x-coordinates is larger, move left or right. In this situation, if monster is at the left hand side, move right, else if monster is at the right hand side, move left. If the absolute value of the difference of two y-coordinates is larger, move up or down. In this situation, if monster is higher, move down, else if the monster is lower, move up. After setting heading, move forward for 20 pixels. And then

check whether the monster coincide with any tail. If there is, number of contact increase by one. At last, randomly generate a timer rate (between 0.4s and 0.7s) for the monster, and motion the monster again after this timer rate.

- Extend the snake tail: first change the color of the snake head to the same as the tail (black with blue border), and stamp the head of the snake, then change the color of the head back to red. Append the head's position to the list **g_tailPositions**. Then move the snake forward for 20 pixels. At last, update the screen.
- Detect body contact between the snake and the monster: once after moving the monster, use a loop to check every tail if any of the tail's distance with the monster is smaller than 15. If there is a tail's distance with monster is smaller than 15, ends the loop, and increase the number of contact by one. Else if none of the nail coincide with the monster, call move monster function again after the random monster timer rate.

Functional Spec

- **setTurtle:** the function is written for the setting of all the Turtle objects. With input parameters, shape, color, x-coordinate, and y-coordinate. Set the shape and color of the turtle object to be same with the input. Set the pen of turtle object up to avoid drawing any lines. At last, set the position of turtle object to be (x, y). Return the turtle object t.
- **setGameArea:** the function is written for setting screen, status area, and motion area. Set the screen title to be "Snake", and set the size of the screen to be 660x740. Then turn off the auto screen refresh by tracer(0). Set the status area by setTurtle function with input y=250. Change the color of status area to be black border and no fill color. And change the size of the status are to be 80x500 (height: 4 times of the default size (20 pixels), width: 25 times of the default size, border: 3 times of the default size). Set the motion area by setTurtle function with input y=-40. Change the color to be black border and no fill color. Change the size to be 500x500 (25 times of the default height and width). Return **g_screen**, **g_statusArea**, and **g_motionArea**.
- **findAllPositions:** the function is written for finding all the possible positions of monster and all food items. Input parameters: a-minimum of the x-coordinate, b-maximum of the y-coordinate, c-minimum of the y-coordinate, and d-maximum of the y-coordinate. First set the allPositions variable to be an empty list. For all i in range a and b, with step 20, for all j in range c and d, with step 20, append (i, j) to allPositions, and return allPositions as output.
- **setMonster:** the function is to set up the monster. Set monster by setTurtle function with input color purple. Find all possible positions of monster by findAllPositions function with input a=-230, b=250, c=-270, d=210. Use a loop to randomly choose a position from all possible positions, check if the distance between the chosen position and the snake is more than 200 pixels. If it is, set the position of the monster and break the loop. If not, continue the loop until a suitable position is found. Return **g_monster**.

- **setFood:** the function is to set up all the food items. First find all possible positions for food items by findAllPositions function with input a=-245, b=240, c=-290, d=200. Remove (-5, -10) position since food item should not coincide with the snake at the beginning of the game. Then randomly choose 9 positions from all the possible positions. Set the 9 food items by setTurtle function one by one. Hide the turtle pen, and set the position. Then write the number of the food item on the screen. Return **g_food**.
- **setIntro:** the function is to set up the intro message. Set intro by setTurtle function with x=-195, y=50, hide the turtle pen, and write intro message on the screen. Return **g_intro**.
- **setMessage:** the function is to set up the status message. Set status message by setTurtle function with y=240, hide the turtle pen, and write status message (with **g_time**, **g_contact**, **g_motion**) on the screen. Return **g_message**.
- **continueGame:** the function is to decide whether the game should be continue or not. If the distance of the monster and snake is smaller than 15, which means monster makes a head-on collision with the snake, game over, return False. Else if **g_food** is a list of 9 empty string, which means all the food items have been consumed, and the length of the list tail positions equals with tail length, which means the snake finishes extending itself, the player wins the game, return False. Else, return False.
- **moveMonster:** if the game is going to continue, first get coordinates of both the monster (xm, ym) and the snake (xs, ys). Then compare the absolute value of the difference of two x-coordinates and the difference of two y-coordinates. In order to move closer to the snake, the monster should move in the direction which the difference is larger. If absolute value of the difference of two x-coordinates is larger, move left or right. In this situation, if monster is at the left hand side, move right, set the heading to be 0, else if monster is at the right hand side, move left, set the heading to be 180. If the absolute value of the difference of two y-coordinates is larger, move up or down. In this situation, if monster is higher, move down, set the heading to be 270, else if the monster is lower, move up, set the heading to be 90. After setting heading, move forward for 20 pixels. And then check whether the monster coincide with any tail. Use a loop to check every tail if any of the tail's distance with the monster is smaller than 15. If there is a tail's distance with monster is smaller than 15, ends the loop, and increase the number of contact by one. At last, randomly generate a timer rate (between 0.4s and 0.7s) for the monster, and motion the monster again after this timer rate.
- **eatFood:** the function is to decide whether a food item is eaten, and change the tail length. If the game is going to continue, check all the food items in **g_food**, if a food item is not an empty string which means that it has not been eaten yet. Get the y-coordinates of both food and snake. If the distance between the food item and the snake head is smaller than 15, and y-coordinate of food is less than snake, increase the **g_tailLength** by k+1, and clear the number written by the Turtle object, then change the food item to be an empty string. Call the function every 0.1s to check whether the snake has eaten a new food item or not.
- **snakeExtend:** the function is to extend the snake's tail once. Change the color of the snake head to the same as the tail (black with blue border), and stamp the head of the snake, then change the

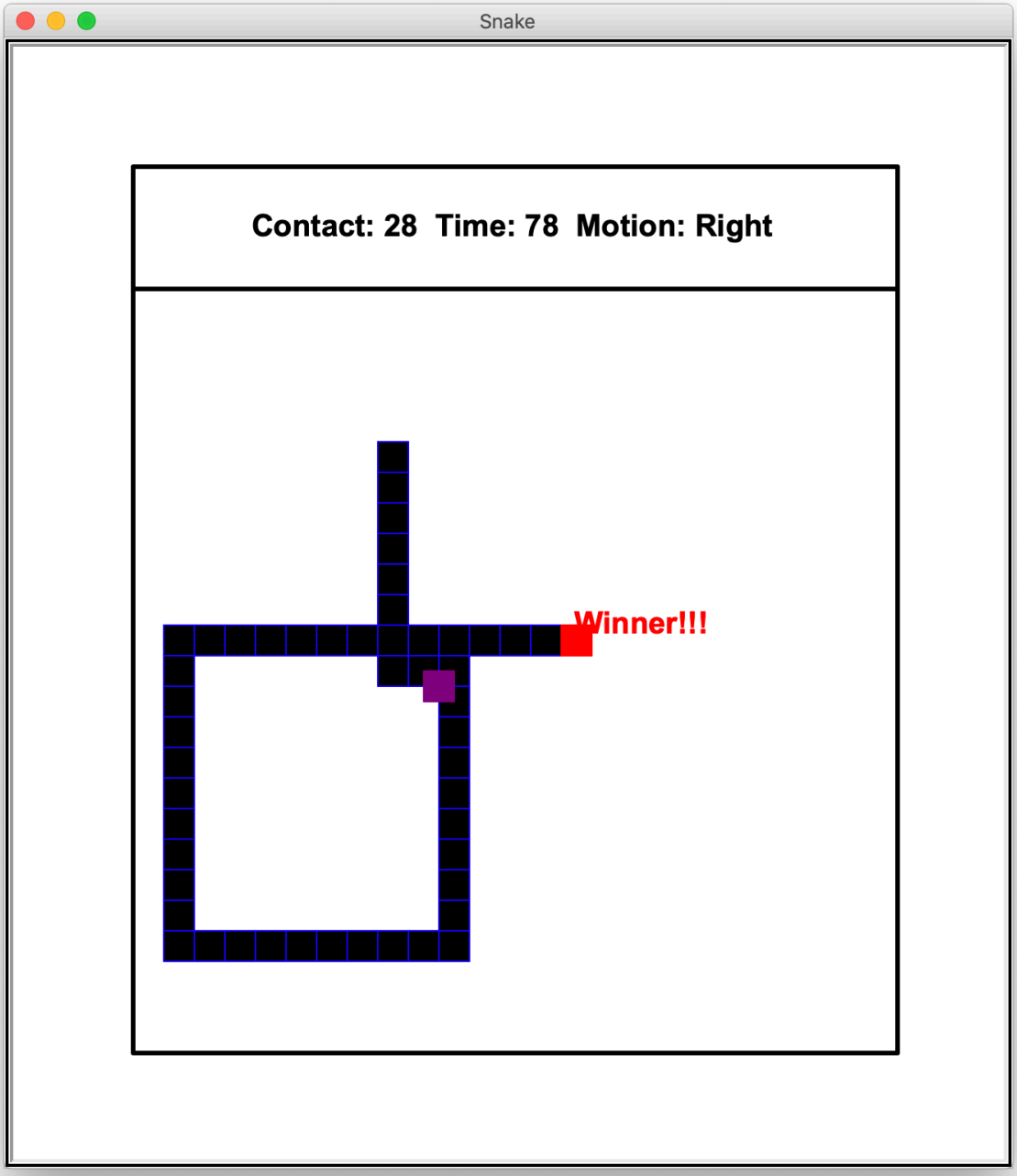
color of the head back to red. Append the head's position to the list **g_tailPositions**. Then move the snake forward for 20 pixels. At last, update the screen. Call the moveSnake function after 0.6s.

- **moveTail:** the function is to move the snake. Change the color of the snake head to the same as the tail (black with blue border), and stamp the head of the snake, then change the color of the head back to red. Append the head's position to the list **g_tailPositions**. Then move the snake forward for 20 pixels. Clear the first stamp of the snake, and remove the first element of the list **g_tailPositions**. At last, update the screen. Call the moveSnake function after 0.3s.
- **extendOrMove:** the function is to decide whether to extend the snake or move the snake. If the length of the tail at current positions is smaller than the global variable **g_tailLength**, extend the snake by calling function snakeExtend, else, move the tail by calling function moveTail.
- **moveSnake:** if the game is going to continue, first get the coordinate of the snake. If the global variable **g_motion** is "paused", call moveSnake function after 0.3s. If **g_motion** is not "paused", set the heading (0, 90, 180, 270) of the snake according to **g_motion** (right, up, left, down). Check if the snake has reached the border (right: 230, up: 190, left: -230, down: -270) or not, if it has reached the border, stop moving the snake forward and call function moveSnake again after 0.3s, if not decide whether to extend the snake or move the snake by function extendOrMove. If the game is not going to continue, call function termination.
- **moveSnakeRight:** the function is to set the global variable **g_motion** to be "Right".
- **moveSnakeUp:** the function is to set the global variable **g_motion** to be "Up".
- **moveSnakeLeft:** the function is to set the global variable **g_motion** to be "Left".
- **moveSnakeDown:** the function is to set the global variable **g_motion** to be "Down".
- **pauseSnake:** the function is to set the global variable **g_motion.** If **g_motion** is not "paused" before, change **g_motion** to be "paused". If it is "paused" before, get the heading (0, 90, 180, 270) of the snake, and set **g_motion** ("Right", "Up", "Left", "Down") according to the heading.
- **setKeys:** the function is to bind keys with functions by ontimer. Bind moveSnakeRight with the right keys, and so on. Bind pauseSnake with the space bar. Call the function every 0.3s to update the motion entered by the player.
- **setTime:** the function is to set the elapsed time of the game. Global variable **g_time** equals time at now minus the start time and further minus 0.5s (delay and the beginning of the game). Call setTime function every 0.1s to update the elapsed time.
- **updateMessage:** the function is to update the status message and display it on the screen. If the game is going to continue, clear the message written before, and then write the new message on the screen. Call the function every 0.1s to update the status message.
- **termination:** the function is to define the termination of the game. If the distance between the monster and the snake is smaller than 15, which means monster catches up the snake, game over, let **g_monster** write "Game Over!!!" on the screen. If all element in the list **g_food** is empty strings and the length of the list **g_tailPositions** equals **g_tailLength**, which means snake consumes all food items and finishes extending its body, the player wins the game, let **g_snake** write "Winner!!!" on the screen.

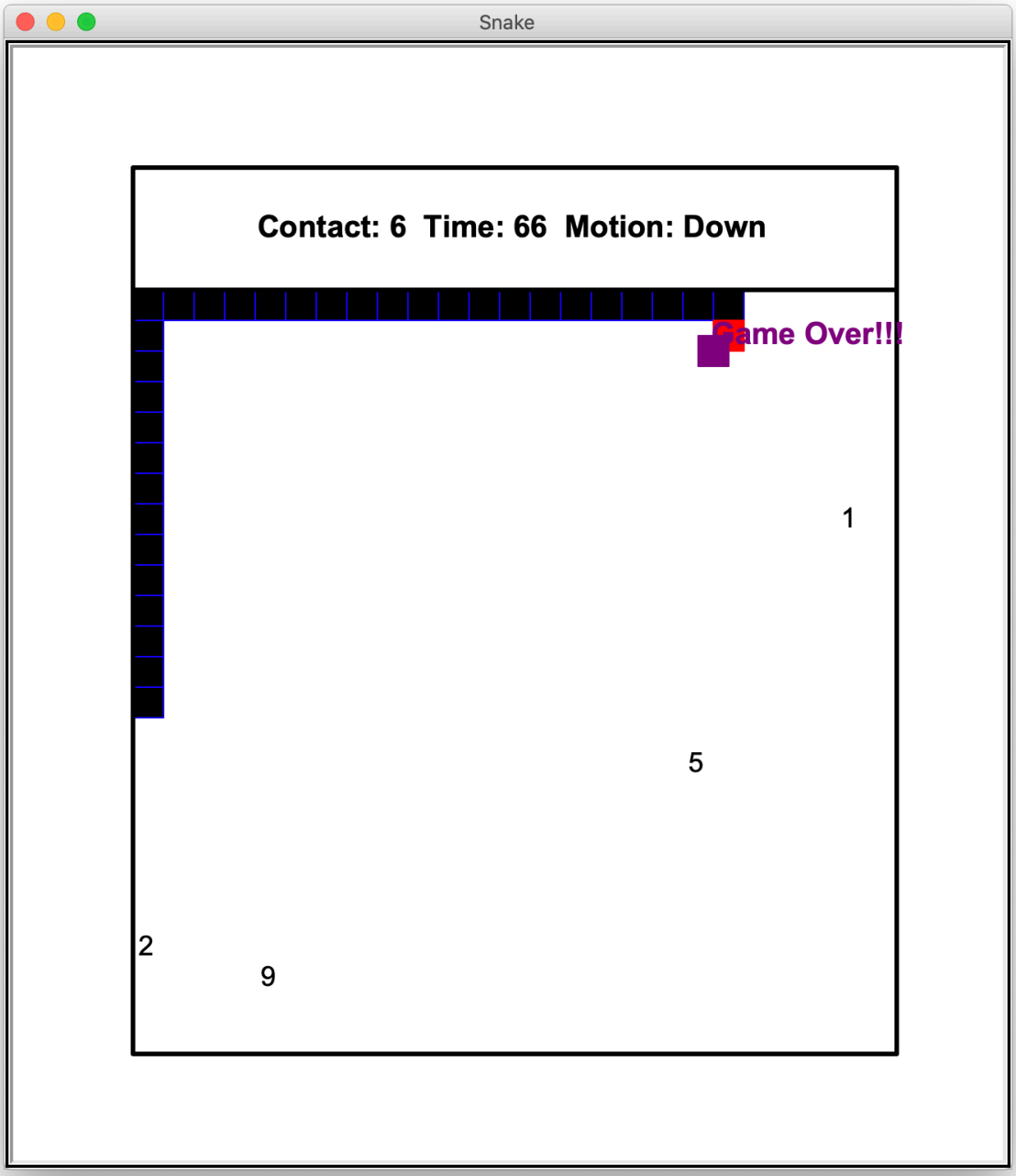
- **startUp:** the function is to start up the game. Initially, after setting, startUp function is bind with click on the screen. Now change the bind of mouse click on the screen to be None to avoid calling startUp function more than one time. Clear the intro message, and set food items by setFood function. Update the screen, and record the starting time of the game. Call setTime function, updateMessage function, moveMonster function, setKeys function, moveSnake function, and eatFood function after a delay of 0.5s.

Sample Output

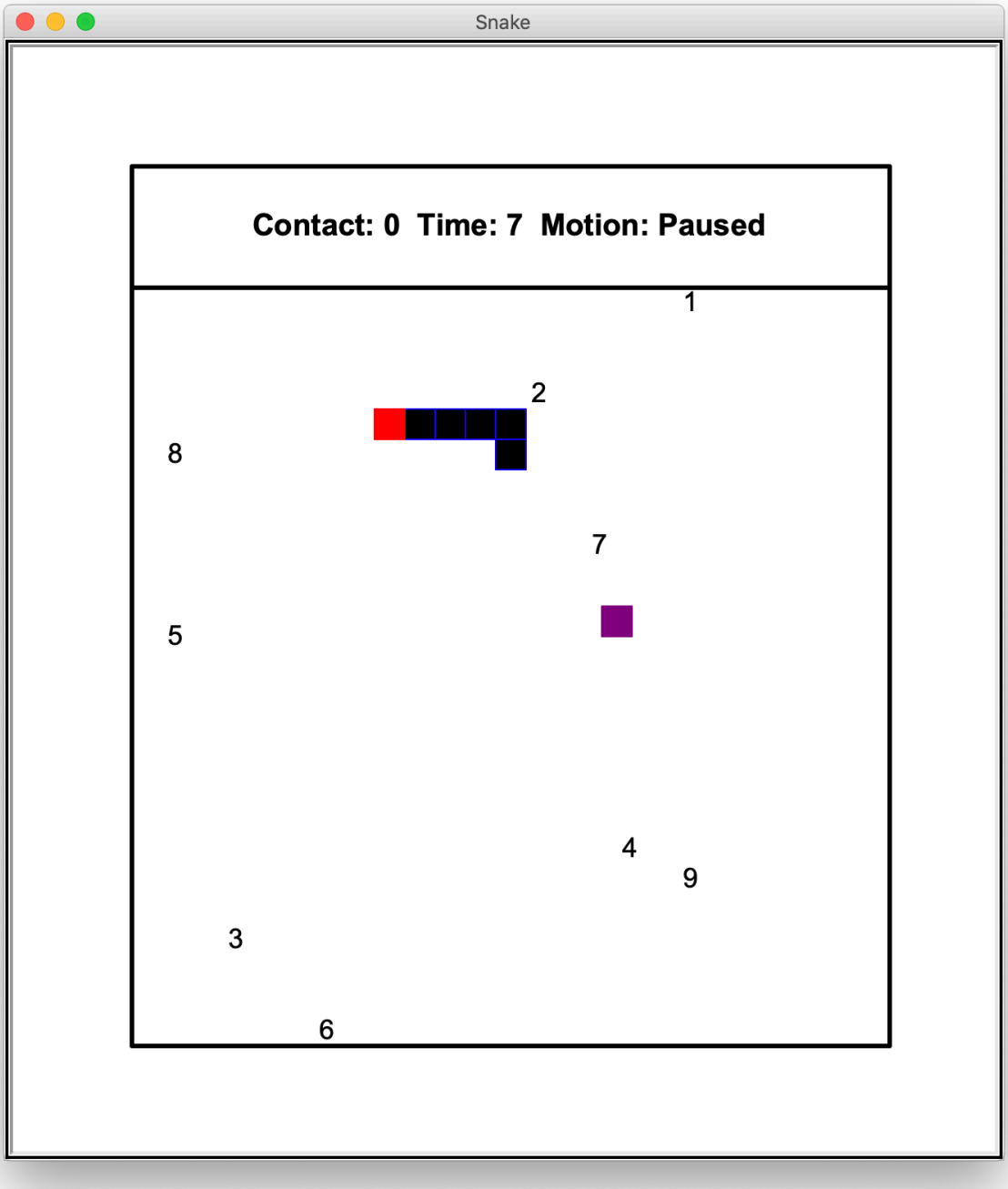
- Winner



- Game Over



- With 0 food item consumed



- With 3 food items consumed