1. **Design**
   a. **Overview**

      This program allows the user to play a snake game in a limit area. Use the 4 arrow keys to control the snake and try to collect 9 fruits to win. Also, the player needs to avoid being caught by the purple monster (contacted with snake's head).

   b. **Data Model**

      1. **Snake and its tail:**

         Head: turtle itself.

         Tail: change the color of head and stamp it along its path for certain times according to the expected snake length.

      2. **Food items:**

         Use 9 invisible turtles to randomly move to 9 places and write from 1 to 9 by aligned center.

   c. **Program Structure**

      At first, use a turtle to draw the frame and write the welcome text. Create and hide the snake (head) at the center of playing area, do the same to the monster and place it randomly in restricted area with certain distance to snake. When the user clicks the screen, clear the welcome text, and show both snake and monster.

      Second, turn off the auto refresh and refresh snake and monster at a manual set rate. For each time interval, use a variable assigned all input motion and the execution must be the last one at the end of interval. To execute the motion, I make snake and monster to move a unit length which is fit to the default size of the turtle square (20 pixels) each time interval. To make different speed, the snake refresh with a slower rate when has eaten fruits, and the monster refresh at a random rate faster or slower than the snake.

      Third, create 9 fruit turtles and place and place randomly. Use a dictionary with key of tuple of its coordinate and value of its sequence number. While the head is sufficiently closed to a fruit, clear the writing and remove it from the dictionary.

      Fourth, at the start of interval, check the absolute distance in x-direction and y-direction. Then the monster moves in axis of the larger one, check if it is positive or negative to decide the direction.

      Fifth, use a list with limited length to store path of snake (head). For items index, which is smaller or equal to snake length, is the snake body for monster to check if the snake is contacted or caught, for others, it is preparation for extension of

the snake in following.

d. Processing Logic

1. The motion of snake and monster

At the start of interval, check the absolute distance in x-direction and y-direction. Then the monster moves in axis of the larger one, check if it is positive or negative to decide the direction. If I restrict the snake in the certain area, I have no need to set a module to avoid it going out of the area. For example, if the monster is at the right edge of the playing area, then its distance in x-axis must be larger or equal to the snake's one. If it is equal, it must be under these 2 circumstances. First, y distance is not 0, the monster moves in y-direction. Second, y distance is small enough and the game ends. If it is negative, it is like the above. If its absolute distance larger, then the monster goes left. If not, go in y-direction.

Second, motion and pause for snake. If the new motion is equal to the last one and still an arrow key, continue. If it is different and it is an arrow key, execute the new one. If it is space and the last one is not space, use another variable to store the motion of last interval and assign the motion of this interval with paused (space). If it is space and the last one is space, take that variable to recover the last motion which is arrow key.

Third, move in restricted place. After executing each motion, check its x-coordinate or y-coordinate. If it is out of boundary, return a Boolean type and replace the snake to that boundary. I consider it finish a complete move and record its position only after this step.

2. The expansion of tail

As I mentioned above, I have a list of position of the path of the head. If and only if the head is really moved but not keep the paused motion of the last interval, the new position will be insert to 0, which means that heading to the frame will not change this list. That is, if I take the first n items to stamp, n-1 will be the length of the snake in this interval (the head and the first one is overlapped), and the stamp of head will be the tail.

What I need to emphasize that the aimLength and snakeLength is different (they are the variable names I use in program). For example, the snake is now length of 4 (including head) and not expanding (aimLength is also 4 and also including head). Then, if the snakes eat a fruit equal to 5, the aimLength becomes 9. From next interval, the snakeLength +1 until it is equal to aimLength. Also, it pluses 1 if and

only if it is not heading to a boundary or paused. That is, it will extend 1 block after a real move.

3. Contact detection

For the head, check the distance between these 2 turtles and return true if it is smaller than a manual set distance.

For the body, check the items in the list of position of the path of the head from 0 to n, representing the snake position of each block. If it is smaller than a manual set distance for a item, contact+1 and ends the iteration immediately to avoid contact+1+1 for one contact because of sufficiently closed to 2 blocks at the same time.

2. Function Specifications

initialization(): Setup window with height of 660 pixels, width of 740 pixels, titled with 'Snake by Luke Chan'. Turn off the auto refresh. Use a turtle to draw the frame. Use a turtle to write welcome text. Create snake (head) and hide. Create monster, place it randomly in restricted area with certain distance to snake and hide. Reset the motion to 'Paused'. Reset the variable to store last directional motion to None. Reset snakeLength to 1. Reset aimLength to 1. Reset collision (contact counts) to 0. Reset time (timer) to 0. Reset locasionList to [(0, -40), (0, -40)] (2 items to avoid bugs). Create a list to store the position of each block of the snake. Reset the flag (game end) to False. Update the screen to show welcome text and frame.

fruit(): Create 9 fruit turtles (invisible). Place them randomly in store (use dictionary with coordinate : sequence number).

eatfruit(): Check the distance between the head and each food (with tuple(dictionary of fruits.keys()). If it is small enough, use this coordinate to find the sequence number. Clear this turtle and remove it from dictionary. Use ontimer with a high rate to in case of 2 fruits which are sufficiently closed.

statusBar(): Clear the last write and write contact, number, and motion.

catch(): Calculate the x distance and y distance between head and monster. And move

according to the results. Then check if it is contacted with body, if it is, contact counts+1. Use ontimer with random rates.

catch_test(): To check whether the monster is sufficiently closed to head. If it is, set the flag to True (means game end).

go_up/down/left/right(): Let the head to go up/down/left/right. If it is out of boundary, replace it and return True.

draw(): locationList means the path of the head. Move the turtle from 0 to snakeLength in locationList and stamp to draw the body. And then go back to the expected position for this interval.

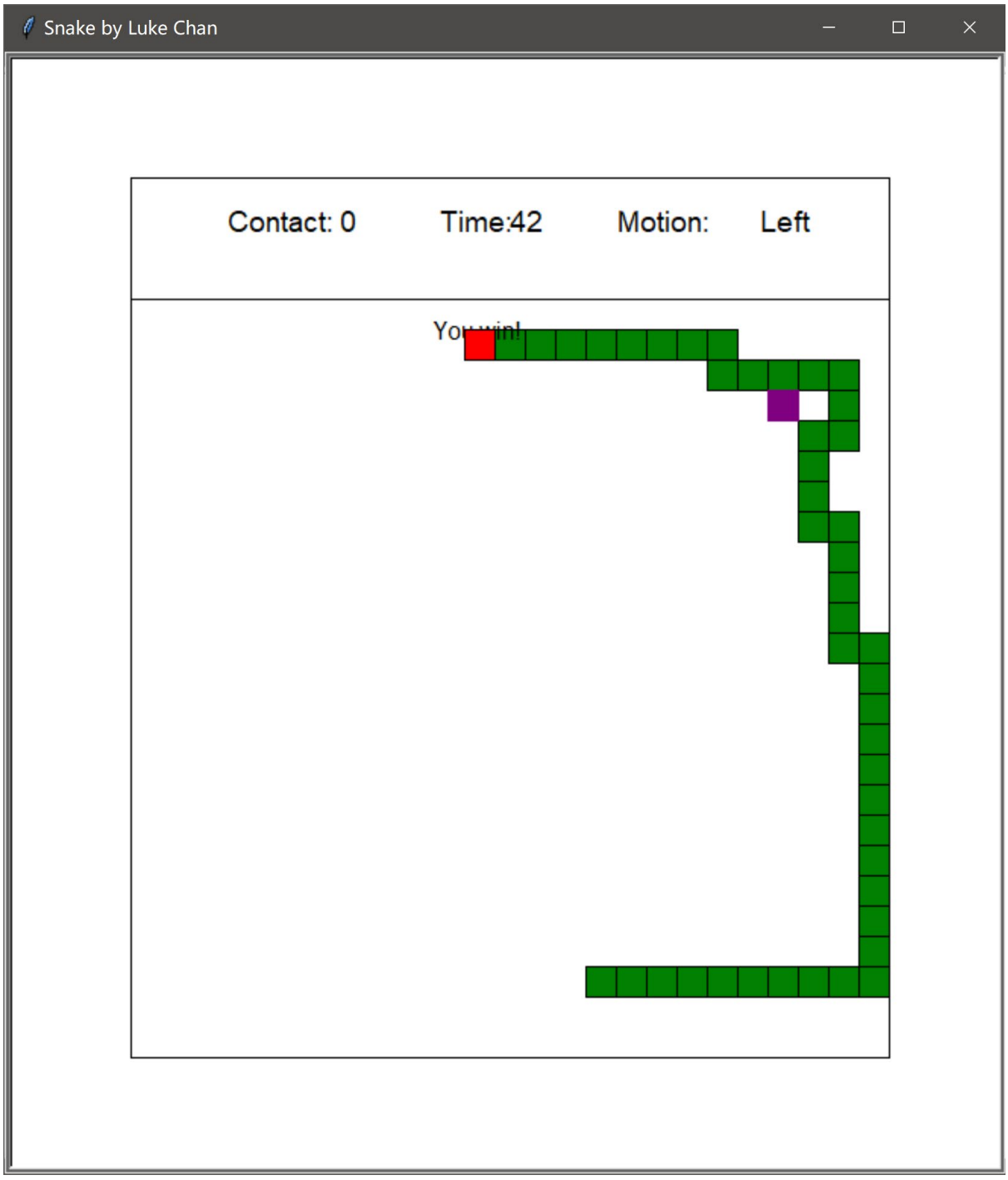
direction(direction: str): Accept the keyboard press and store it with motion. For ‘space’ means ‘paused’, it stores the current motion and pause. For meaning of ‘unpaused’, it recovers the last valid motion.

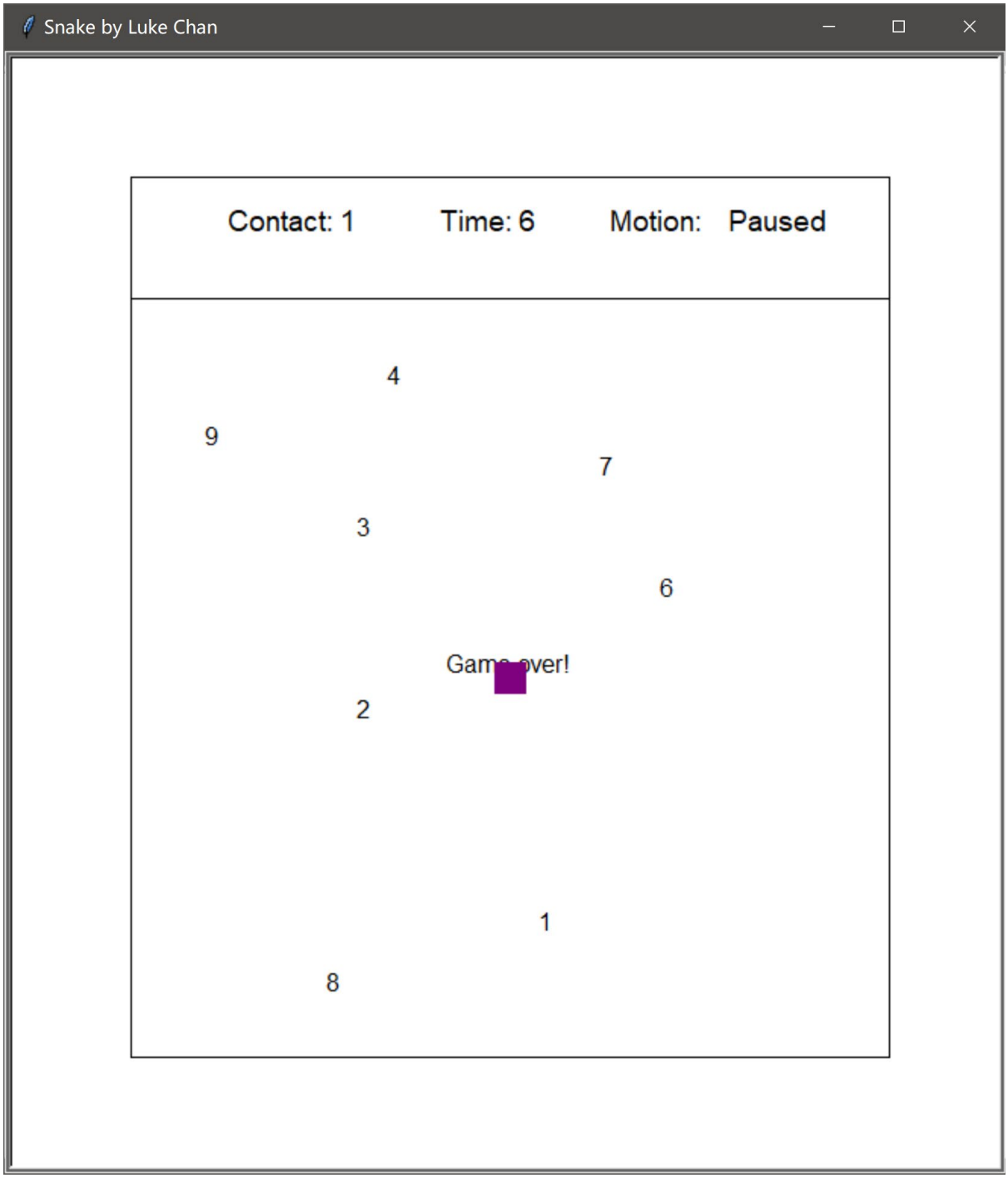
repeat(): This is the repeating module for the snake and other functions. Reset the k = None at first to and then assigned by go_up/down/left/right to detect whether the snake is heading the boundary. Call statusBar() to refresh information. Add the new location of head if the game is not paused and the snake is not heading the boundary. Remove the last item if the locationList is longer than 47 to improve performance. Reset slow = False. If snakeLength < aimLength (means extending) and lastHeadLocation != locationList[0] (means moving), make slow = True (means to refresh with slower rate for ontimer at last) and snakeLength += 1 . draw() to draw the snake. If snakeLength >=46 (have eaten up all fruits and fully extended), write ‘win’, if flag ==True, write ‘game over’, else continue.

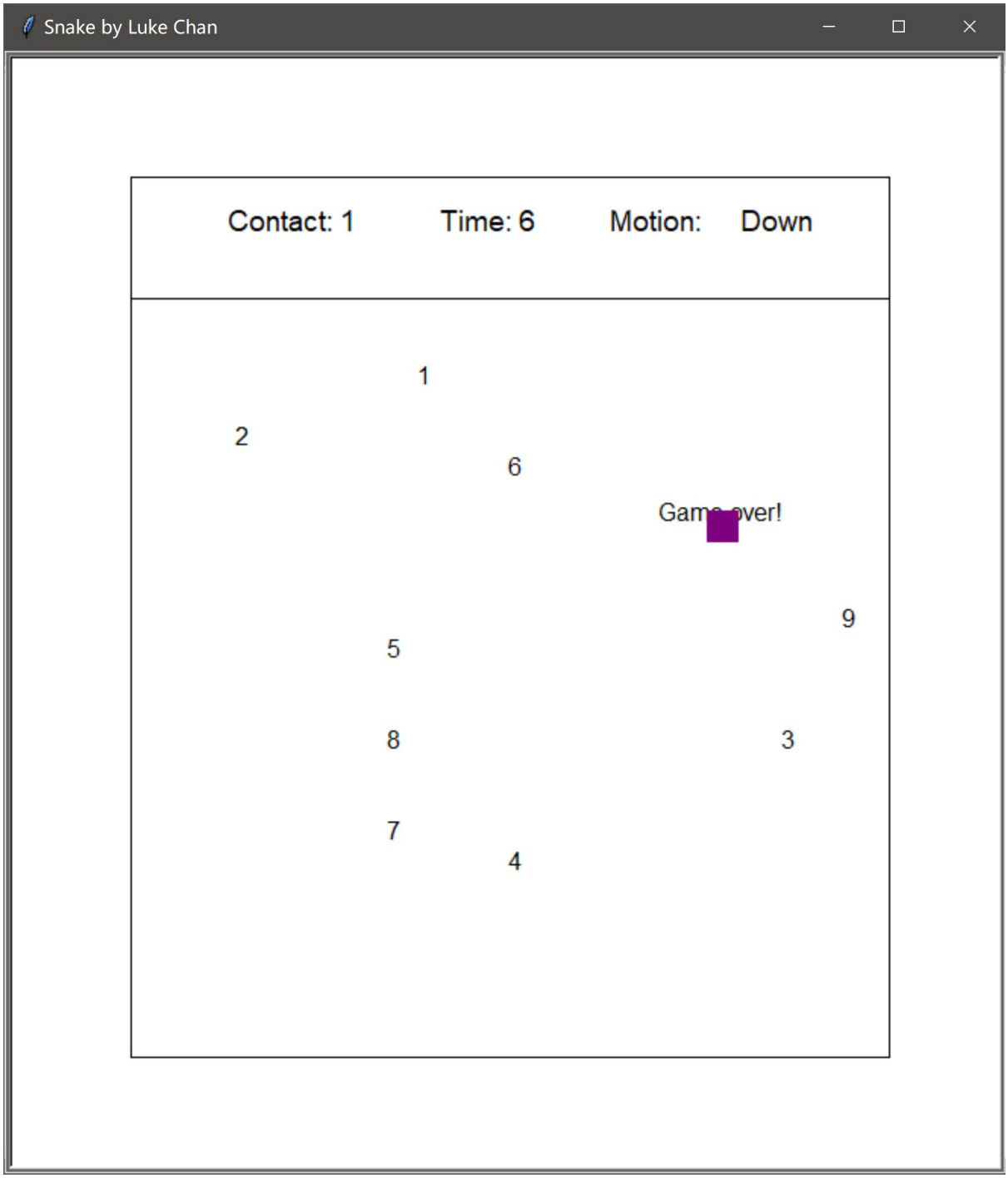
game(): Reset onscreenclick to None to avoid click again. Clear welcome text and show head and monster. Call fruit(), eartfruit(), catch(), catch_test(), repeat(). Listen on press key and convey it to direction().

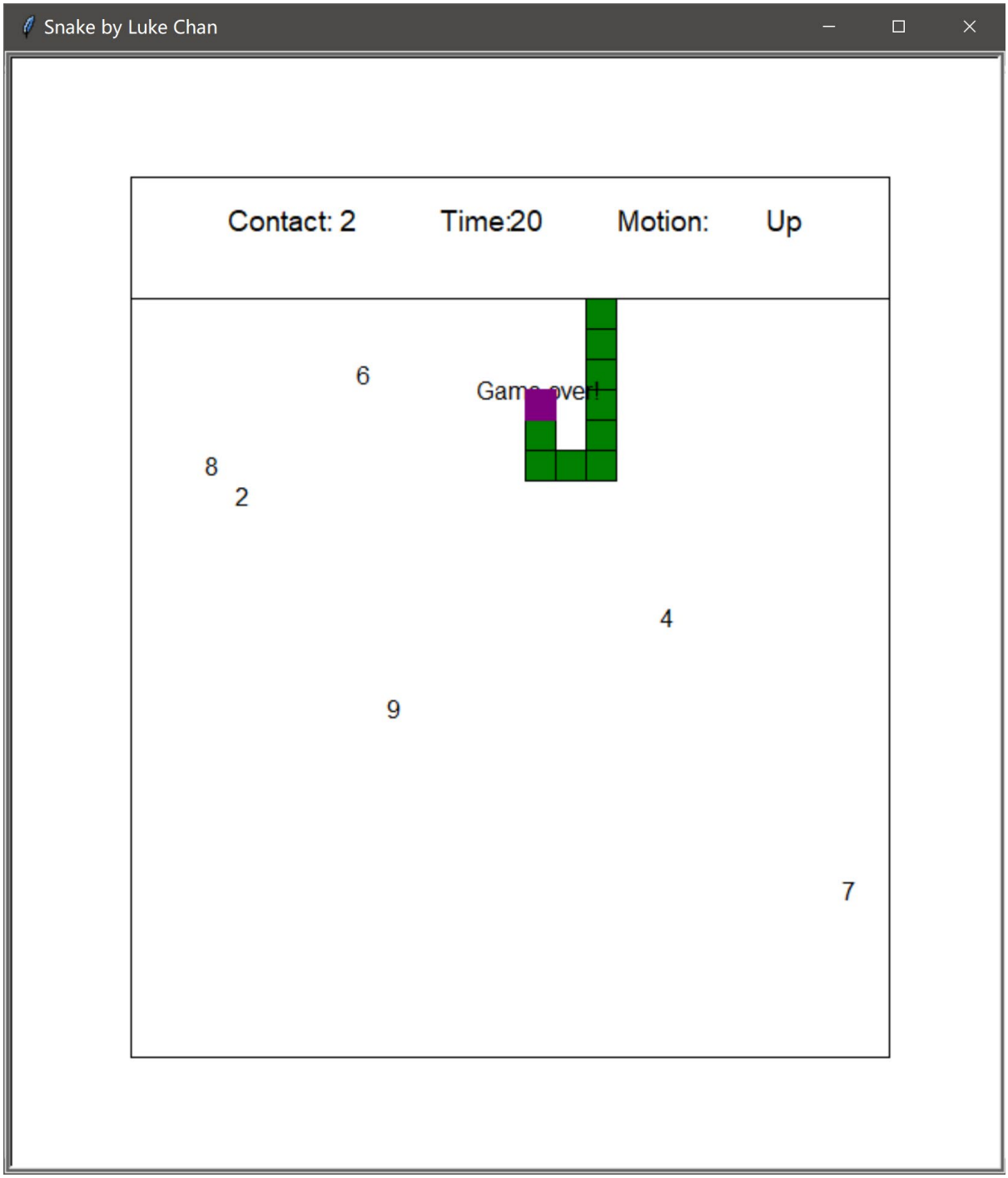
main(): Call initialization(). Wait for click, and hold the window.

Samples:




Snake by Luke Chan
Contact: 1
Time: 6
Motion: Paused
4
9
7
3
6
Game over!
2
1
8


Snake by Luke Chan
Contact: 1
Time: 6
Motion: Down
1
2
6
Game over!
9
5
8
3
7
4


Snake by Luke Chan
Contact: 2
Time:20
Motion:
Up
6
Game over!
8
2
4
9
7