1. Design

a. Overview

This project is all about designing and developing a Snake game. In short, Color Snake is the game where player maneuvers a line which grows in length. The composition of the game itself consists of 3 main objects. Those are the monster, a set of numbers as the food items and the snake itself. Initially, the snake is represented with a sequence of squares, its head is “SpringGreen” colored and its body is “Black” colored. Each of the nine foods has its own color and its own number. When snake consumes the food, its head will be altered to the food’s color. The length of the snake increases with a value equal to the number of food that is consumed.

The goal of the game is to move the snake around with four arrow keys (up, down, left, right) and to consume all of the nine foods before the monster collides its head toward the snake’s head. Although the monster itself is designed to follow the snake’s head (with intention to make a head-collision), its move is still randomized in either horizontal or the vertical direction.

A popular high-level programming language, Python, is used in designating this game. The programming language was created by Guido van Rossum (and the community) and released in the late 20^{th} century. There are some reasons why the language is used to develop the game. Those include easy-to-implement code and designation of readability.

b. Data Model

1) Boolean

- In Python, boolean variables are defined by the True or False value. Though it is possible to use the integer type of variable (indicating 0 as the False and 1 as the True), boolean variable saves up ton of the allocated memories. For example, onGoing = True.
- In this program, boolean is used to evaluate a True or False value. In its example, it is used to determine ceaseless of the game, whether the snake is currently paused or not, and validating whether the food is generated with the according distribution of distance.

2) Dictionary
   - In Python, dictionary acts as the map in the other programming language. It maps the keys with a specific value. The key itself is not limited to a specific data type (it allows the key to be a string, or even a list).
   - In this program, dictionary is used to determine the delta x and delta y when moving the snake. In the example, dx, dy = {"down" : (0, -20), "left" : (-20, 0), "right" : (20, 0), "up" : (0, 20), "pause" : (0, 0)}[snakeHead.direction].
3) Integer
   - In Python, integer is a data type that stores numerical value. For example, snakeSpeed = 200.
   - This Color Snake game is a snake game that is affected by a bunch sequence of numbers including its body length, its speed, current total food, etc.
4) List
   - In Python, list contains items which separated by comma and enclosed within the square brackets [ ]. It is possible to access the value inside using the slide operator

[ ] and [:] with indexes starting at 0 in the beginning of the list. For example, foodTurtle = [None] * foodTotal.

- In the program, list is one of the most important variables used. List maintains several items and compact them in one variable. List is used in storing the position of food, the position of snake's body, etc.

5) String

- In Python, strings are identified as a contiguous set of characters represented in the quotation marks. For example, backgroundColor = "DarkSeaGreen".
- This program uses string in many parts. For example, to change snake's head color everytime it is consuming the food, the variable snakeHeadInner is used as the string data type.

6) Turtle

- Python provides a "Turtle Graphics" library. Turtle enables users to create pictures and shapes by providing them with a virtual canvas. Not only that, but turtle allows users to program in an interactive way.
- This program mainly uses the python turtle in most of its primary parts. All the snake head, the snake bodies, even the foods are using the turtle data type.

c. Program Structure

1) Line 49 – 62

- This click() function is used to start the game right after the user click the screen.

2) Line 64 – 71

- This drawLine() function is used to draw a line given the coordinate of point1 and point2. The line starts from (point1_x, point1_y) to (point2_x, point2_y)

3) Line 73 – 76
    - This drawPolygon() function is used to draw a polygon given the set of coordinates of points.
4) Line 78 – 101
    - This generateFood() function is used to generate a total of n foods each have the distance with the other larger and according to distribution constant.
5) Line 103 – 105
    - This insideMotionArea() function is used to check whether a point is inside the game motion area (inside the border).
6) Line 107 – 109
    - This manhattanDistance() function is used to check the manhattan distance between 2 point (point1 and point2).
7) Line 111 – 136
    - This monsterMove() function is used to move monster in approaching the head of snake randomly in the vertical direction or the horizontal direction.
8) Line 113 – 119
    - This contact() function is located inside the monsterMove() function. This contact() function is used to detect and increase the count of body contact when monster is overlapping the body of snake.
9) Line 120 – 122
    - This setMove() function changes the length to the default size (20) with maintaining the same direction. (i.e. length = (length / abs(length)) * min(20, max(20, length)))

10) Line 138 – 141
    - This snakeDown() function alterates the snake's head direction to the "down" direction.
11) Line 143 – 146
    - This snakeLeft() function alterates the snake's head direction to the "left" direction.
12) Line 148 – 217
    - This snakeMove() function moves the snake according to the last pressed key by user.
13) Line 150 – 159
    - This consumeFood() function allows snake to consume and remove the food when the distance between snake's head and food is less than or equal to the default size.
14) Line 160 – 172
    - This snakeBodyMove() function replicates the current snake's head as its own body and stamp it while moving its head.
15) Line 173 – 194
    - This statusUpdate() function updates the status of the game including the total body contact, the current motion, and total time.
16) Line 195 – 203
    - This writeMessage() function writes a "message" at "point" location with customized color.
17) Line 219 – 222

- This snakeRight() function alterates the snake's head direction to the "right" direction.

18) Line 224 – 232
    - This snakeTogglePause() function toggles the pause state (on to off / off to on).
19) Line 234 – 237
    - This snakeUp() function alterates the snake's head direction to the "up" direction.
20) Line 239 – 243
    - This toggleScreen() function toggles the screen tracer (0 to 1 / 1 to 0)
21) Line 245 – 311
    - This main body of code initializes every items required in the game. Those include the screen setup, introduction, draw border in status area and motion area, monster initialization, snake's head initialization, keys binding initialization, and text status initizalilation.

d. Processing Logic

1) As described previously, both of the snake's head and the monster are using the turtle graphics variable to represent them. In the motion itself, the program allows the user to input several keys (those are [Up], [Down], [Left], [Right], [Space]). The screen schemes every sequences of time and listen to the inputted keys of user. If there is not any key to be pressed or there are several keys that are pressed, the last pressed keys will be considered. Since the snake will move about 20 units (equal to 1 block), the dictionary is used to define both of delta x and delta y. In this case, "up" direction equals to (0, 20), "down" direction equals to (0, -20), "left" direction equals to (-20, 0), "right" direction equals to (20, 0), "pause" direction equal to (0, 0) for both delta x

and delta y.

ontimer() function (available in turtle class) plays the most important role in determining delay of move (i.e. speed of each object). The ontimer function allows user to input 2 parameters (function, delayTime). The function will execute the function() right after the delayTime is passed (delayTime is in millisecond units). In here the default delay for the snakeHead is 200; however, the delay is increased up to 400 when the snake is currently growing its body after eating food. On the other hand, the default delay for the monster itself is randomized from 200 to 350. This ontimer function is placed before the start of the game and inside the move function. For example, def move_snake(): #code screen.ontimer(snakeMove, snakeSpeed). The snakeMove() function also determines the move of snake based on the last pressed keys as mentioned previously. Given the exact delay and its moving position, snake is able to move according to what it is required to do so.

For the monsterMove() function, the move of monster is based on its delta of x and delta of y. Here we let dx = snakeHead.xcor() - monster.xcor() and dy = snakeHead.ycor() - monster.ycor(). If dx is less than 0, that means the snake's position is in the left of the monster. This means that monster should consider its move toward the left direction rather than the right direction. Conversely, if dy is more than 0, that means the monster should consider its move toward the right direction. Same applies in vertical direction.

Theoretically, the snakeSpeed (in here, it is defined as its delay) should not exceed the monsterSpeed (in here, it is also defined as its delay). This is one of the reasons

why the monster delay is randomized with min(snakeSpeed, randomizedDelay) = snakeSpeed.

2) To expand its tail. We use the stamp() function and clearstamps() function provided in turtle class. To “stamp”, we have to change the current head’s snake color into its own body (we assume the head as the newly born body). After changing its head, we use the stamp() function and revert the snake’s head color to its original head color. With all of these, we are able to expand the snake’s tail. However, the snake will expand with no limitation (i.e. the most left-behind tail is always located in its origin / center position). Now, to remove that we can use the clearstamps(1). This allows the function to remove the very last tail. Since the length of snake is important (as we have to consider adding the value of eaten food to its length), the use of clearstamps(1) may only be used once len(head.stampItems) is larger than length. In here length is defined as snake’s body length. To maintain that, we can simply use a global integer variable.
3) This aspect is connected with the previous tail extension. First thing to know, when the monster is making a contact with snake’s body, the prior distance between monster’s position and one of the body’s position is smaller than 20 must be held. This requires us to maintain each of the body’s position. Again, to do that, we can simply use a list of tuple (tuple of 2 integer data types that represents x0 and y0). While extending its tail (actually its neck), we save the current snakeHead’s position before moving it towards its direction (i.e. after the stamp() function, we execute snakeBodyPosition.append(snakeHead.position())). However, again, with the same mistake, we might encounter the position of snake’s body in the origin / center

position as the snakeBodyPosition is used to have that position (even after the snake's position does not locate in there anymore). This is why we have to remove the very first encountered position of its body. We are able to remove that with del snakeBodyPosition[0] or snakeBodyPosition.pop(0). But this program uses snakeBodyPosition = snakeBodyPosition[1:]. There is no specific reasons since all of those are running in O(length) time.

2. Function Specification

1) def click():

   This procedure mainly starts the game (it triggers once the user click anywhere on the screen). Before the game starts, it remove the description that is written before. After that, this procedure calls several procedure including the generateFood(foodTotal) (will be explained later). Then it turns off the onclick by executing screen.onclick(None) to reduce some unknown errors.

   Before the last part, this click procedure set the screen.ontimer for both snakeMove() procedure and monsterMove() procedure. In the end, it keeps updating the screen as the both of the moving procedures run.

2) def drawLine():

   This procedure draws a line given the coordinate of point1 and point2. The line consists of (point1_x, point1_y) and (point2_x, point2_y). To do that, we first declare a turtle variable to write the whole thing. Before moving itself to the point1 coordinate, we have to use the .penup() procedure so that it won't make the line from origin to point1. After reaching itself in the point1 coordinate, we simply use the .pendown() procedure to start writing and move itself to point2. To move the turtle we can use the .goto() procedure.

3) def drawPolygon():

   This procedure draws a polygon given the set of coordinates of points. Say the points consists of n points, n = [point1, point2, …, pointN]. To draw the polygon, we use the drawLine() procedure described above. We simply draw the line between point1 to point2, point2 to point3, …, pointN-1 to pointN and pointN to point1.

4) def generateFood():

   This procedure generates a total of n foods. Each of the food have distance with the other larger and according to the distribution distance constant. Mainly it generates the food farther away from the other food (in here we also consider the snake and monster as the "foods", then we later remove them as they are not really the foods). We keep randoming the x between -225 to 225 and y between -265 and 185 (both are inclusive) until the manhattanDistance between the (x, y) and every foodPosition until it is larger than distanceDistribution. To display those foods, we are using the turtle variables to represent each of them. We simply move them according to the randomized position before and set up its shape as the foodShape (here it is equal to "square"). Now, in general views described in this assignment, the food should only show its number (i.e. we should hide its turtle). But to make it more variant, we allow the food to be displayed (with .turtlesize(0.25)) and a specific color that was declared in the global constant of food colors. This allows us to distinguish the color of food so that we are able to see the snake's color head varies according to the eaten food's color. We also write the number of the foods so that the length of snake is able to be noticed easily once it grows the length according to the number of foods.

5) def insideMotionArea():

   This function checks whether a point is located inside the motion area (it returns True if it is located inside, False if otherwise). After doing the coordinate things, we observe that the motion area is a square with size 500 x 500 (from the coordinate of (-250, -290) to (250, 210)). This is why a point is said to be inside the motion area if its x-coordinate is located between -250 and 250 and its y-coordinate is located between -290 and 210 (inclusively).

6) def manhattanDistance():

   To mention, there are several ways to count a distance. The normal distance that we know is the euler distance which equal to sqrt(delta_x^2 + delta_y^2). This, however, count the manhattan distance as abs(delta_x) + abs(delta_y). Though it somehow looks strange, this is way more useful than what it looks.

7) def monsterMove():

   This procedure moves the monster to approach the head of snake randomly in vertical direction or horizontal direction. This procedure consists of two other defs: contact() and setMove() which will be described later. To determine its move direction, we count dx and dy as the difference of x-coordinate and y-coordinate. Define dx = monster.xcor() - snakeHead.xcor() and dy = monster.ycor() – snakeHead.ycor(). We take the opposite direction of the current dx and the current dy as the move orientation for the monster. Generally, if the snakeHead is located in the up-right position of the current monster; then, the monster should only consider the “up” and “right” direction for its move. To make the monster looks more natural, we use the random move to determine whether the monster should move in horizontal direction or vertical direction. To make the monster

move in 20 units (considered as 1 block), we use the setMove() function to maintain the same direction while modifies its unit.

8) def contact():

   This procedure detects and increases the count of body contact when the monster overlaps the snake's body. To determine whether it touches the body, we use the manhattanDistance() function that is previously described. When the manhattanDistance of the monster and snake's body position is less than or equal to 20, the contact increases by 1 and the procedure is returned (as we define the contact in every move of the monster).

9) def setMove():

   This setMove() function returns a compressed length to the units of 20 (i.e. it returns 20 * negativity_of_length)

10) def snakeDown():

    This snakeDown() procedure alterates the snake's head direction toward the "down" direction. However, snake's can not just move "up" then move "down" straight away. This is why there are several constraints that need to be considered such as the last move should be "left" or "right" or from the "pause" motion

11) def snakeLeft():

    This snakeLeft() procedure alterates the snake's head direction toward the "left" direction. The rest is similar with the snakeDown() procedure.

12) def snakeMove():

    This snakeMove() procedure moves the snake according to the last pressed key by user. This procedure consists of four other defs: consumeFood(), snakeBodyMove(),

statusUpdate(), and writeMessage() which will be described later. To move the snake, we firstly define the value of dx and dy through the direction of the snakeHead. We can use the dictionary with keys as the string type (for the direction) and tuple of two integer data types (for the delta x and delta y). It will be used as following: dx, dy = {"down" : (0, -20), "left" : (-20, 0), "right" : (20, 0), "up" : (0, 20), "pause" : (0, 0)}[snakeHead.direction]. The we can simply move the snake to (x + dx, y + dy) where x = snakeHead.xcor() and y = snakeHead.ycor(). After each move, we have to call the consumeFood() procedure to remove the eaten food and proceed with the tail extension and snake's head color changing. After that we proceed with the statusUpdate() procedure to update the status including the body contact count, motion, and total time elapsed.

13) def consumeFood():

This consumeFood() procedure consumes and removes the food when the distance between snake's head and food is less than the default size. To determine whether the food is able to be consumed or not, we use the manhattanDistance function. If the manhattan distance between snake's head position and food's position is less than or equal to 20, then the food is eatable. After consuming the food, we clear the text (the number of the food that is written previously) and hide the turtle using the hideturtle() procedure. After that we proceed with the changing head color of snake and its tail extension.

14) def snakeBodyMove():

This procedure creates a body with the use of stamp as what it has been described in the beginning. This procedure append the current head position to the list of positions for the

snake body position. Then it changes the snake's head color into the body one, "stamp" it and revert it back to the head's color. Now, if the current snake length is larger than what it should be, we have to remove the very-last tail of the snake. We can do that using the clearstamps(1) procedure provided by turtle library. For the list of position itself, we have to remove the first element in the list (in this program, snakeBodyPosition = snakeBodyPosition[1:])

15) def statusUpdate():

This procedure update the status area in the top of screen. Firstly, the program check whether the game is over or not by considering the length of the stampItems of the snakeHead. If the stampItems is equal to 50 (i.e. 5 + (1 + 2 + 3 + … + 9)), the game is over and the player is announced to be winner. Else, if the distance between snakeHead and monster is less than or equal to 20, then the game is over and the player is announced with "GAMEOVER" message. Other than that, the update status area will be updated according to the current state of status such as the number of contacts, the motion, and the time.

16) def writeMessage():

This procedure writes a message at point given its location with customized color. To write the message we firstly use the turtle variable as its sign. We use the write() procedure provided from the python turtle library and hide the turtle itself.

17) def snakeRight():

This snakeRight() procedure alterates the snake's head direction toward the "right" direction. The rest is similar with the snakeDown() procedure.

18) def snakeTogglePause():

This procedure toggles the pause state, on to off or off to on. We can simply use the xor operation provided from the python library. The use is given as following: snakePause ^= True. Then we can adjust the snakeHead direction and the last key direction based on the current pause state.

19) def snakeUp():

This snakeUp() procedure alterates the snake's head direction toward the "up" direction. The rest is similar with the snakeDown() procedure.

20) def toggleScreen():

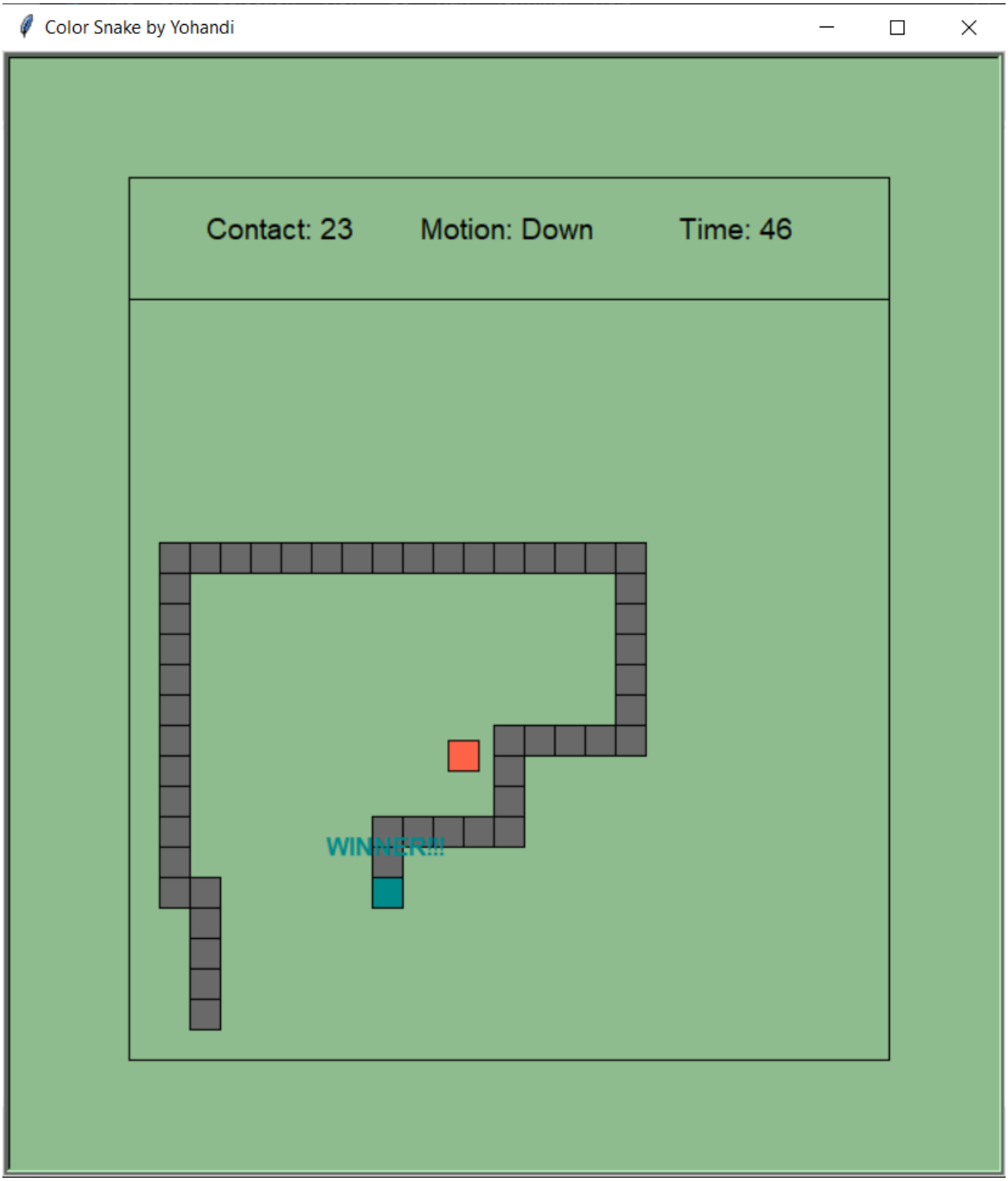
This procedure toggles the tracer of screen (0 to 1 or 1 to 0). To do that we simply execute toggleStatus = 1 – toggleStatus.

21) if __name__ == "__main__":

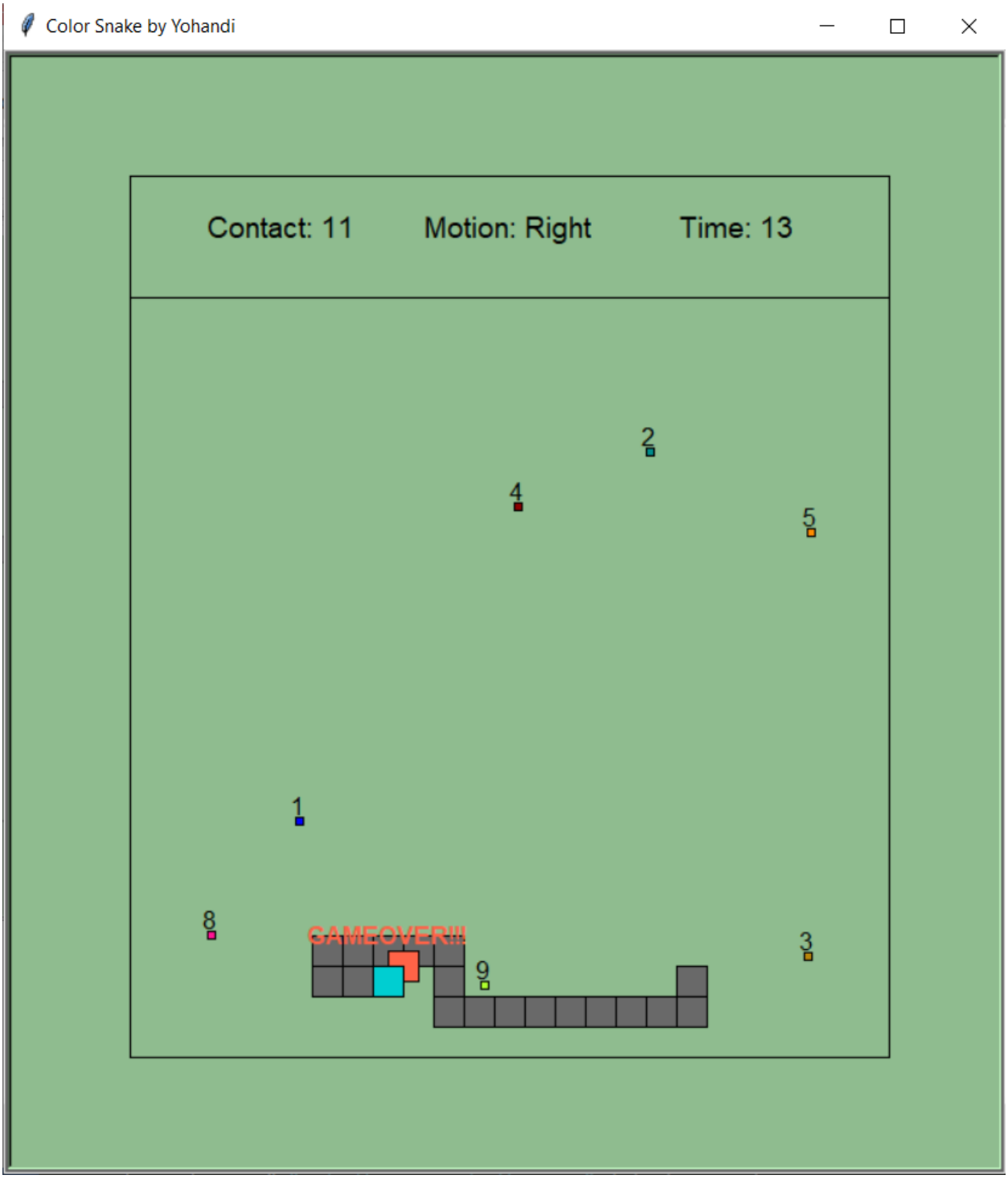
This body main part initializes most of the objects in the game such as the screen setup, introduction text, drawing border for upper status area, drawing border for lower motion area. Not only that but it also initialize both monster and snake's head according to the distance distribution. Then it runs the game accordingly.

3. Output

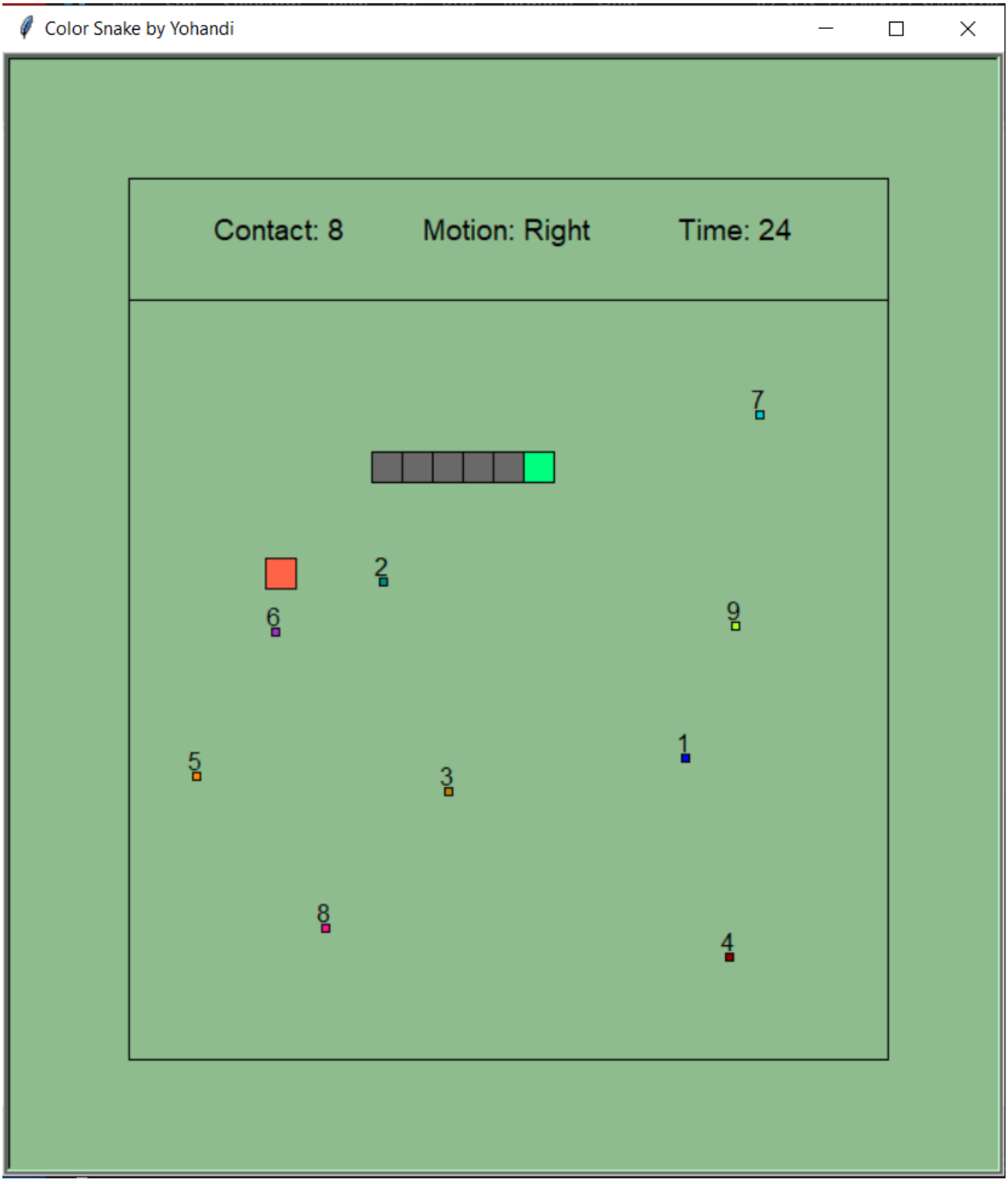
i. Winner


Color Snake by Yohandi
Contact: 23
Motion: Down
Time: 46
WINNER!!!

ii. Game over


Color Snake by Yohandi
Contact: 11
Motion: Right
Time: 13
2
4
5
1
8
GAMEOVER!!!
9
3

iii. Various stage of the game: 0 food item consumed



iv. Various stage of the game: 3 food items consumed (1, 2, and 4)