Design doc

1. design

a. overview

This program allows users to play the snake game. First of all, the user can click the mouse to start the game. Random numbers representing the food will be displayed. The users can press the four arrow keys to maneuver the snake's four moving directions, which respectively up, down, left, and right. If the head of the snake is caught by the monster before the snake consume all the food and gets fully extended, the user loses the game. On contrary, the user wins.

Before beginning the game, a description will be showcased. After click the screen, the description will vanish automatically. The title of the snake contains three parts: contact, time, and motion. They will be automatically updated as the game going. The represent the times the Monster touches the snake's body, the time used, and the direction of the moving, respectively.

The user can press the space key to pause the game. The motion will be Paused in turn. The user can press the arrows or the space to restart the game.

b. Data model

1. the data model for the snake:

The turtle objects for the snake:
tail: used to draw the snake's tail. It is black in color with blue boundary, size 20*20
head: used to draw the head of the snake. It is red in color, size 20*20

The other essential global variables used for the snake:
g_length: it is an integer.it records the length of the tail. Initially, the length of the snake is 6 (which you can see from the screen that only 5 squares of the tail, because one of which is hidden),the g_length will record the updated length of snake.
g_extend: it is an integer. Its number stans for the length that the snake is about to extend.
g_list1: a list used to record the direction of the snake's motion. Initially, it contains element 'N' and each time you press the arrow keys, it will record the right, left, up, down moves as 'a', 'b', 'c', 'd' respectively. If only N in the list, it means the user clicked th mouse but haven't press the arrow keys to start the game.

g_list2: a list used to store the stamp_id of the tail stamps. The main purpose of using it is to obtain how many stamps are out. By comparing it with the g_length, which is the length of the tail, if the elements in g_list2 is larger than or equal to the g_length, the g_ taillist will remove the last element and the g_list2 will pop the stamp to be clear. Then use clearstamp() function to remove that stamp to make the snake maintain its length.
g_list4: a subsidiary list used to check whether the Monster contacts the tail of the snake. For more information, please refer to the Processing Logic part.
g_taillist: a list used to record the position of the tails. This tail will record the existing positions of the tail stamps, by their coordinates. The coordinates will be stored in the list by tuples. as the snake moving, the list will automatically delete the vanished coordinates (the judgement is dependent on g_list2) and constantly generate new coordinates for the tail stamps

2. the data model for the food items

The turtle object to write the food:
pen4: used to write the food's numbers in the screen.

Other important global variables for the food:
g_dict_food: a dictionary used to generate the key-value pairs of the food items. In each item of the dictionary, the key is the coordinates of the food elements, which is packed in tuples. the value is the number of the food. Initially the dictionary is empty. After click the screen, the dictionary will contain 9 elements and later on the dictionary will delete the eaten food, once the snake's head contacts the food. If all food is consumed, the dictionary will be added one key-value pair: 10: 'finished', which is used for the examination of termination in the end.

3. other data model which is important for the program

The turtle objects for drawing and writing:
pen: draw the boundary and title line of the screen
pen1: write the description
pen2: write the status of the motion and Contact
pen3: write the status of time
pen5: write the terminal words: 'Game over' and 'Winner'

The turtle objects for Monster:
Monster: a square and purple turtle. Control the move of the monster.

Other important global variables:
g_i: used to check whether the game is started. If g_i ==0, the game hasn't got started; else, the game is going on. Initially g_i =0
g_j: used to check whether the game is paused. If paused, g_j ==1; else, g_j==0. At first g_j=0 and since g_i initially=0, g_j will not affect the game that hasn't started.
g_k: used to start the first move.

g_Time: the time used in the game
g_motion: the direction of the motion for the head, or paused
g_Contact: the times that the monster contacts the body of the snake

c. program structure

Basically, the program contains the following five parts:

Part 1. the initialization of the game

In this part, the program is intended to initialize the game, this includes the drawing of the boundaries, the title, and writing the descriptions. On top of that, this part will initialize all the turtles to be used. For instance, the color of the turtles, the boldness and content to be write, the heading directions, and whether to up or down the pen. This part will properly place the Monster and head objects.

The functions used in this part:
initialize_the_head(): the function is to initialize the turtle object head, which stands for the head of the snake.
initialize_the_tail(): the function is to initialize the turtle object tail, which stands for the tail of the snake.
initialize_the_pens(): the function is to initialize the six pens used in writing and drawing the boundary. For instance, to hide the trace of them and so forth.
draw_the_title(): the function is used to draw the title lines of the screen.
draw_the_boundary(): the function is used to draw the boundary lines of the motion area.
write_the_title():the function is used to write the three words: ‘contact’, ‘time’, and ‘motion’.
write_the_description(): the function is used to write the description words displayed before the start of the game.
draw_the_carve(): the function is used to set the scale of the screen (660*740), set the pen size and draw and write the former 4 functions mentioned
write_the_content_1(): the function is used to write the number of the Contact and the status of motion.
write_the_content_2(): this function is used to write the time of the game.
initialize_the_monster(): this function is used to initialize the turtle object Monster, which stands for the monster required.
initializer (): this function is used to conduct all the functions mentioned in this part. It is the combination of all the initialization process.

Part 2. The control on the move of the snake

In this part, the program is intended to control the move of the snake. This includes the control over the motion of head, such as the direction, whether to collide with the boundary and so forth. Also, it includes the motion of tail, to extend when the head consumes some food and how to delete the stamps by the tail, making it act as if the snake is moving.

<u>The functions used in this part:</u>
right(): the function is used append ‘a’ in the g_list1. As mentioned above, this list is used to store the last direction the player chooses. Also, it will change the motion status and if the game is paused, it will restart the game.
left(): the function is used append ‘b’ in the g_list1. As mentioned above, this list is used to store the last direction the player chooses. Also, it will change the motion status and if the game is paused, it will restart the game.
up (): the function is used append ‘c’ in the g_list1. As mentioned above, this list is used to store the last direction the player chooses. Also, it will change the motion status and if the game is paused, it will restart the game.
down(): the function is used append ‘d’ in the g_list1. As mentioned above, this list is used to store the last direction the player chooses. Also, it will change the motion status and if the game is paused, it will restart the game.
head_dir(): the function is used to connect the right(), left(), up(), down() functions to the arrow keys Right, Left, Up, Down, respectively.
is_boundary(): the function is used to check whether the snake’s head touches the boundary of its motion area (500*500)
collide(): the function is used to tell whether the head of the snake arrives at the boundary.
tail_move(): the function is used to control the move of the tail. It will update the list for the tail’s elements and make and clear stamps representing the tail automatically.
snake_move(): the core of the snake’s move. It contains the control on the head’s move and the combination of the former judge, including whether in the boundary and which direction the snake is heading towards.

Part 3. The control on the move of the Monster.

In this part, the goal is to make the monster chasing the head of the snake, which means to find the nearest approach to the snake. On top of that, this part will record the contact between the monster and the body of the snake. The update of the contact times is conducted in this part.
<u>The functions used in this part:</u>
obtain_the_cooradinates(): this function is to find the coordinates of the head and monster.
Monster_move(): this function is used to control the move of the monster and record the times of contact. For more details, please refer to the Processing Logic part.

Part 4. The display of the food and the eating food process

In this part, the program will generate the initial food items and randomly displace them. In the process of the game, the food displacement will update itself each time the head consumes the food items. On top of that, this part will record the food that has been eaten and obtain the length the tail is about to extend.
<u>The functions used in this part:</u>
randomize_food(): the function is used to obtain the initial nine food items. It will store their coordinates and values in a dictionary.
place_food(): the function is used to write the food items in the carve.
is_food_eat(): the function is used to check whether the head is colliding with the food items.

The function will return the value of the food eaten and prepare the tail to be extended.

Part 5. The central control of the whole game

This part severs as the brain to connect the whole process of the program. It includes the start of the game, the terminate of the game and the pause of the snake's move.

The functions used in this part:

start(): to start the snake game. It includes the clear of the description, make the snake move, start the record of time, place the food.
terminate(): to check whether the game has ended or not. It including check the winning situation and the losing situation
is_catch(): to check whether the monster catches the head of the snake. It serves as a subsidiary of the terminate() function.
pause(): to pause the game, when the game has been starting.
restart(): to restart the game, when the game has been paused.

d. Processing Logic

1. the logic used to motion the snake and monster

For the head of the snake, its motion is controlled by the player. Its motion direction is controlled by the key arrows, each time when not hitting the wall or not terminated, the head will move forward 20 automatically. The speed of the head depends on the rate of refresh. And the rate of refresh (ontimer() function) is affected by whether the head is consuming food or not.

As for the tail of the snake, its motion is automatically following the head. First of all, there is a goto() function, and the first square of the tail (which is hidden by the head)will go to the position of the head. Each time after the tail moves, it will leave a stamp at its new place. At the same time, we record how many stamps are in the carve. If there are more carve than the length required, we use clearstamp() function to remove the last square. When the tail leaves the stamp, the list g_list2 will record the id of the stamps. To remove the last square, we use pop(g_list2[0]), it can not only remove the id from the g_list2 and the return value can be used in clearstamp() function to clear the last stamp. As a result, the tail can be seen like moving after the head. Its speed is the same as the head.

As for the monster, its motion is also to chase the head. Its speed is randomized. To obtain the direction of the motion, the program will locate the relative locations of the head and the monster. If they are in the same line (vertically or horizontally), the monster will move towards the head indeed in this line. Else, the monster can randomly choose to move vertically or to move horizontally, and in all is approaching the head. The choose of direction in this situation depends on the random number from random.randint(0,1), if num=0, it will move horizontally, else, it will move vertically.

2. the logic used to expand the snake tail

First of all, when the head touches the food, the length of the tail will not immediately change. But in the program, the g_length global variable will change at no time. As mentioned above, g_list2's element is generated as the head moves one time, so the length of g_list2 will not immediately change. As the g_length change, the length of g_list2 will be shorted than the g_length. Therefore, before g_length equals the length of g_list2, the clearstamp() will not be executed. Therefore, the length of the tail is extended because each time the tail moves forward, it will leave a new stamp while the last stamp will not be cleared at time time.

And how do we know at a specific time whether the snake is extending? The technique is the global variable g_extend. g_extend is an integer. Each time when the head touch a food item, g_extend will add up the value of the food. Each time when the snake is moving, g_extend will subtract 1 if g_extend is not equal to 0. So we can set that if g_extend ==0, the head will move at a higher speed, and if g_extend is not 0, the speed of the head will be slower. Using this variable, we can distinguish how many moves the snake need to slow down to extend its body.

3. the logic the detect the body contact between the snake and the monster

As mentioned above, g_taillist is used to store all the positions of the tail stamps. Here I will show how to use the g_list4 and g_taillist to detect the contact.

Firstly, g_taillist contains all the coordiates of the tails. Each time after the monster moves, we can obtain the position of the Monster (x1, y1). Then we search through the g_taillist. For any coordinates in that list, if there exists a pair of coordinates in the list (x2,y2), such that the absolute value of (x1-x2) and (y1-y2) all smaller or equal to 20, it will automatically append number 1 to the g_list4.

Why do we need g_list4? That is because the speeds of the Monster and the head are different. The Contact only refers to the move about monster, so at each Monster's move, the g_Contact will increase at most once. Therefore, we cannot directly add another number to g_Contact as long as there are contacts.

We can avoid this problem by create g_list4. Each time we check whether there is the contact, we can simply check whether there is number 1 in the list. If not, then it will not add 1 to the g_Contact. Else, it will add 1 and only add once to the g_Contact. Pay attention that each time before the detect process, we should clear the g_list4.

2. function specifications

initialize_the_head(): this function will initialize the head turtle. It will pull the head pen up and hide its trace. It will set the color of head as red.

initialize_the_tail(): this function will initialize the tail turtle. It will pull the tail pen up and hide its trace. It will set the color of tail, making the boundary of tailing being blue and the body of tail being black.

initialize_the_pens(): for all the pens in this function, the function will hide the turtle of the four pens, hide the traces, and make the pens up. For pen3 (the pen for writing g_time), it will additionally set the position of the pen3 at (5,230). For pen5 (the pen for terminate), it will additionally set the color as red.

draw_the_title(): this function will use turtle pen to draw the upper rectangle. It will initially set the pen in (250,210), and then put the pen down the write. It will move through the (250,290), (-250,290),(-250.210) then moves back to (250,210), to form a complete rectangle.

draw_the_boundary(): this function will use turtle pen to draw the lower rectangle. It will initially set the pen in (250,210), and then put the pen down the write. It will move through the (250,-290), (-250,-290),(-250.210) then moves back to (250,210), to form a complete rectangle.

write_the_title(): this function will set the pen at (-230,230) first to write “Contact”, then set the pen at (-75,270) to write “Time”, then set the pen at (52,230) to write “Motion” . The format of the writing is “Arial,14,normal”

write_the_description(): this function will set the position of pen1 at (-230,50). Then it will use pen1 to write the description of the game. The format will be Arial at size 12.

draw_the_carve(): this function will at first set the size of the screen as 660*740. It will set the size of the pen as 4 and then draw the carve using the last four functions mentioned above

write_the_content_1(): this function will set pen2 at(-120,190) then writing the number of Contact. Then pen2 will be set at (150,190), to write the status of the head’s motion. The format of the writing is “Arial,12,normal”. Each time before the conduction, it will clear itself.

write_the_content_2(): this function will first detect whether the game has over or not. If not, it will go into the inner part. It will globalize the variable g_Time and it will conduct the function itself with the method ontimer for every 1 seconds (1000 ms). Each time conduct the function, it will add 1 to the g_Time. On top of that, it will write the Time used. The format of the writing is “Arial,12,normal”. The function is called only when the game has been started. Each time before the conduction, pen3 will clear itself.

initialize_the_monster(): this function will pull the monster up, hide its trace, and set the color of the Monster as purple. On top of that, the function use while True loop and random object to obtain a position for the Monster. The coordinates must satisfies abs(x)>140 and abs(y-40)>140, to make sure the monster is far from the head at the beginning. Then the loop break and the monster will be set to the position in turn.

initializer(): this function will first set the title of the turtle graphic as ‘Carlo’s snake game”. then it will draw the carve, initialize the turtles using the functions given above. Besides, it will write the initially g_Time = 0 for the starting interface. It will call the write_the_conten_1() to write the motion=Paused and Contact=0

right(): this function will globalize the variable g_motion. It will append 'a' to g_list1, turing g_motion to "right",and write the motion at that time via function write_the_content_1(). It will also call the restart() function. The restart() function is only useful when in pause.

left(): this function will globalize the variable g_motion. It will append 'b' to g_list1, turing g_motion to "left",and write the motion at that time via function write_the_content_1(). It will also call the restart() function. The restart() function is only useful when in pause.

up(): this function will globalize the variable g_motion. It will append 'c' to g_list1, turing g_motion to "up",and write the motion at that time via function write_the_content_1(). It will also call the restart() function. The restart() function is only useful when in pause.

down(): this function will globalize the variable g_motion. It will append 'd' to g_list1, turing g_motion to "down",and write the motion at that time via function write_the_content_1(). It will also call the restart() function. The restart() function is only useful when in pause.

head_dir(): this function will connect the above 4 functions with the arrow keys 'Up', Down', 'Left', 'Right' respectively. The method used in this function is onkey()

is_boundary(): this function will obtain the coordinates of the head first and check whether the snake is on the boundary or not by obtaining their corresponding coordinates. If so, it will return False. Else, it will return True.

collide(): this function is used to detect whether the head is on the boundary. It will return False if it the head is on the boundary and at the same time the direction of the head is towards the boundary. In other situations, it will return True.

tail_move(): this function is used to control the move of the tail. First of all, it will call the collide() to check whether the head can move. If not, the function will conduct nothing. If can, it will obtain the position of the head and go to the position of head. The tail will leave stamps at each new position and clear the last stamp if the number of stamps is about to exceed the length of the tail. In this function, g_list2 will record the id of existing stamps and g_taillist will record the coordinates of the existing tail stamps. For more information, please refer to the Processing Logic.

snake_move(): this function will first check whether the game is terminated or paused. If not, it will firstly obtain the coordinates of the head, and check the last element in the g_list1. Respectively, it will set the heading direction of the head using seth() method. Then it will check whether the snake is within the boundary (boundary itself is not included). If so, it can move 20 forward. If not, only specific directions can make the snake go, to put forward 20. If the motion is wrong, it will stay still. It will call the head_dir() and tail_move() in this function too. It will automatically refresh itself via ontimer function. If g_extend==0, the time in ontimer will around from 220-260; else, the time will be 450. If the game id paused, the update rate is 200 ms. If all the food is consumed and the snake is fully extended, it will add

10: ‘finished’ to the g_dict_food.
obtain_the_coordinates(): this function will obtain the coordinates of the head and the monster, and return their coordinates.

monster_move(): this function will globalize the g_Contact. At beginning, it will detect whether the game is terminated. If not, it will obtain the coordinates of the head and monster. Then it will tell the direction of the monster. The process can be seen in Process Logic. It will also check the contact of the monster and the body. The process can also be seen in Process Logic.

randomize_food(): this function will generate the initial food. The food is choose from random. The x-coordinate and y -coordinate are set as 20*random.randint(-11,11). This can ensure no overlap in the same square. Then set i=1. It is not until i=9 can the iteration stop. In each iteration, it will pack the coordinates and i to the g_dict_food. In each iteration, i will add 1 to itself.

place_the_food(): each time before the conduct, the pen4 will clear the content it write before. Then pen4 will go through the g_dict_food and at each position to write the value of the food. To make the number in the middle, if the coordinates in g_dict_food is (x,y), the pen4 will go to the position (x-5,y-5).

start(x,y): the parameters x and y have no specific meanings, just for the correctness of format. It will globalize g_i and g_motion. At first it will clear pen1, which is the description of the game. then set the default motion as “right”, randomize and displace the food. It will also call the write_the_content_1 and write_the_content_2 respectively. After that, g_i will add 1. (g_1==0 is stop, g_1==1 is going on)

terminate(): it is to check whether the game should end or not. the first condition is g_i==1 and 10 in g_dict_food. This is to ensure the game is going on while all the food is consumed and the snake is fully extended. Recall function snake_move(), if all the food is consumed and the snake is fully extended, it will add 10: “finished” to the g_dict_food. The other condition is the head is caught by the monster and the game is moving on. In each situation, it will find the position of the head (x,y) and pen5 go to the position (x-80,y) or (x-100,y), respectively. in the first situation it will write “Winner!!” while in the second situation it will write “Game over!!”. If it is in one of the two situations, it will return True. Else, it will return False.

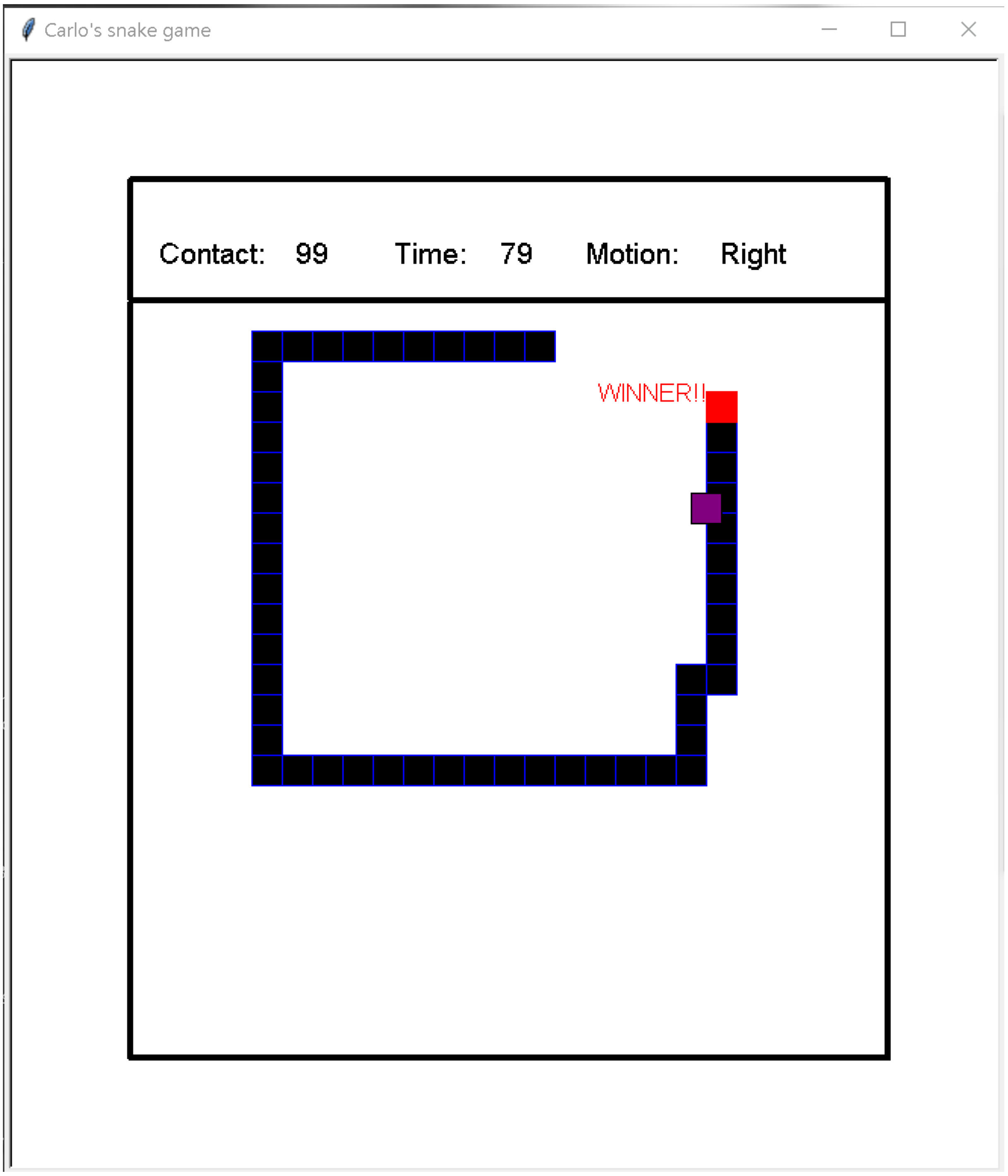
is_catch(): it will obtain the positions of the head and the monster. Then it will compare. If abs(x1-x2) and abs(y1-y2) simultaneously less or equal than 20, it will return True. Else, it will return False.

pause(): it is recalled by the space. If g_j==0(which means not paused), the call of the function will make g_j==1(which means is paused). Vice versa. It will change the status of motion at the same time.

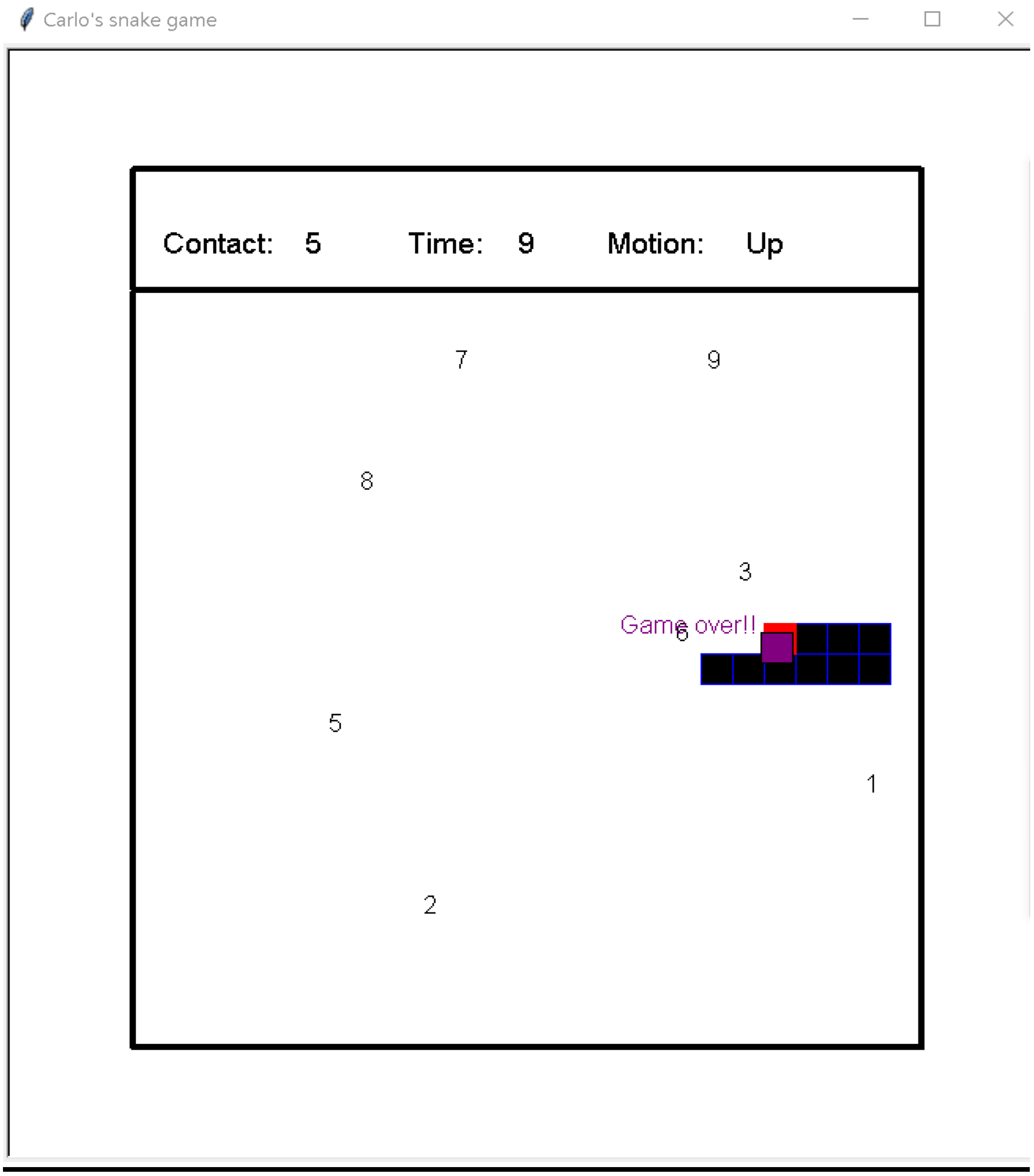
restart(): under paused situation, it will turn g_j==1 into g_j==0. Then it will refer to the last element in the g_list1 to determine the motion of the head, then change the g_motion respectively.

3. output

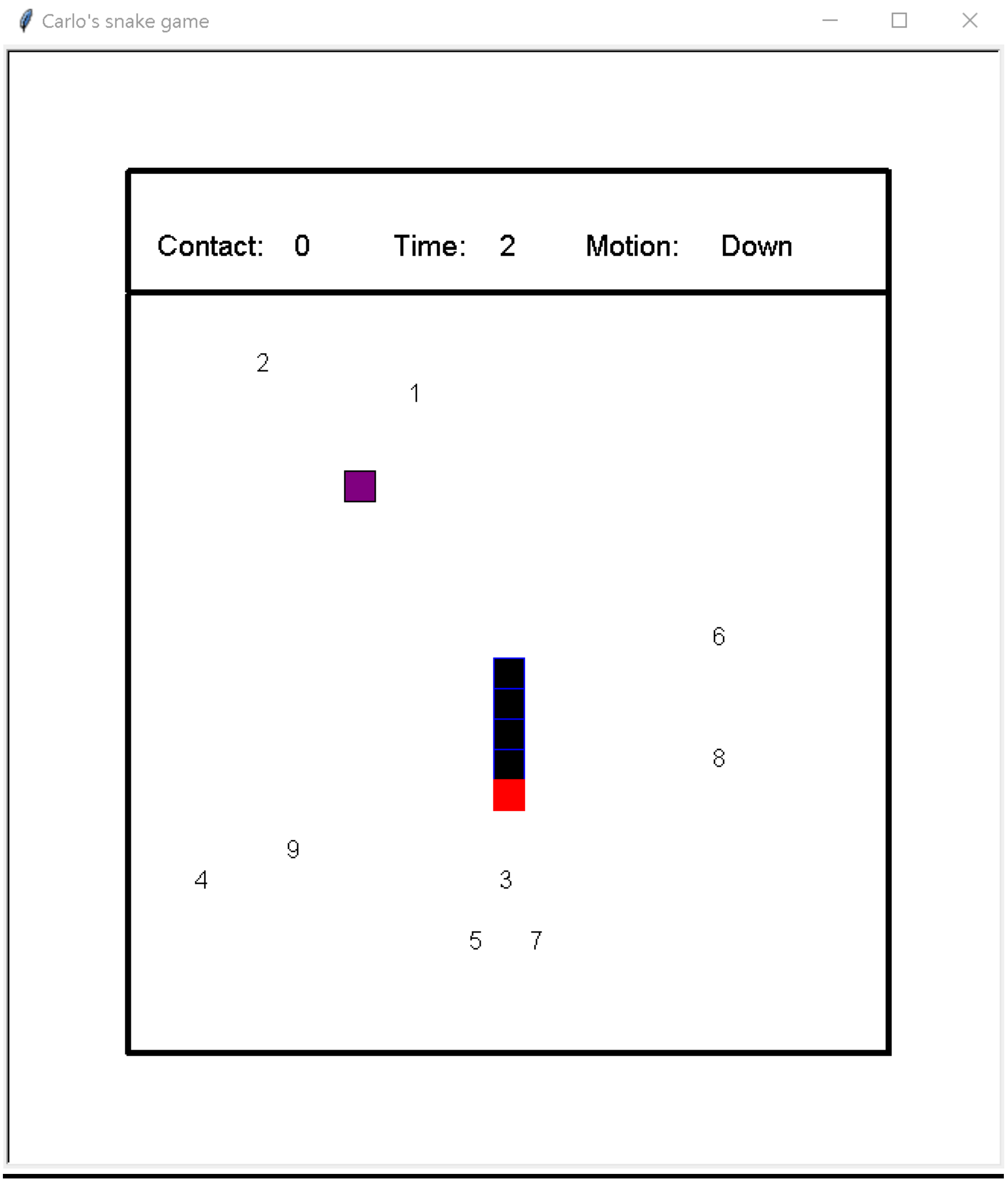
1. winner:



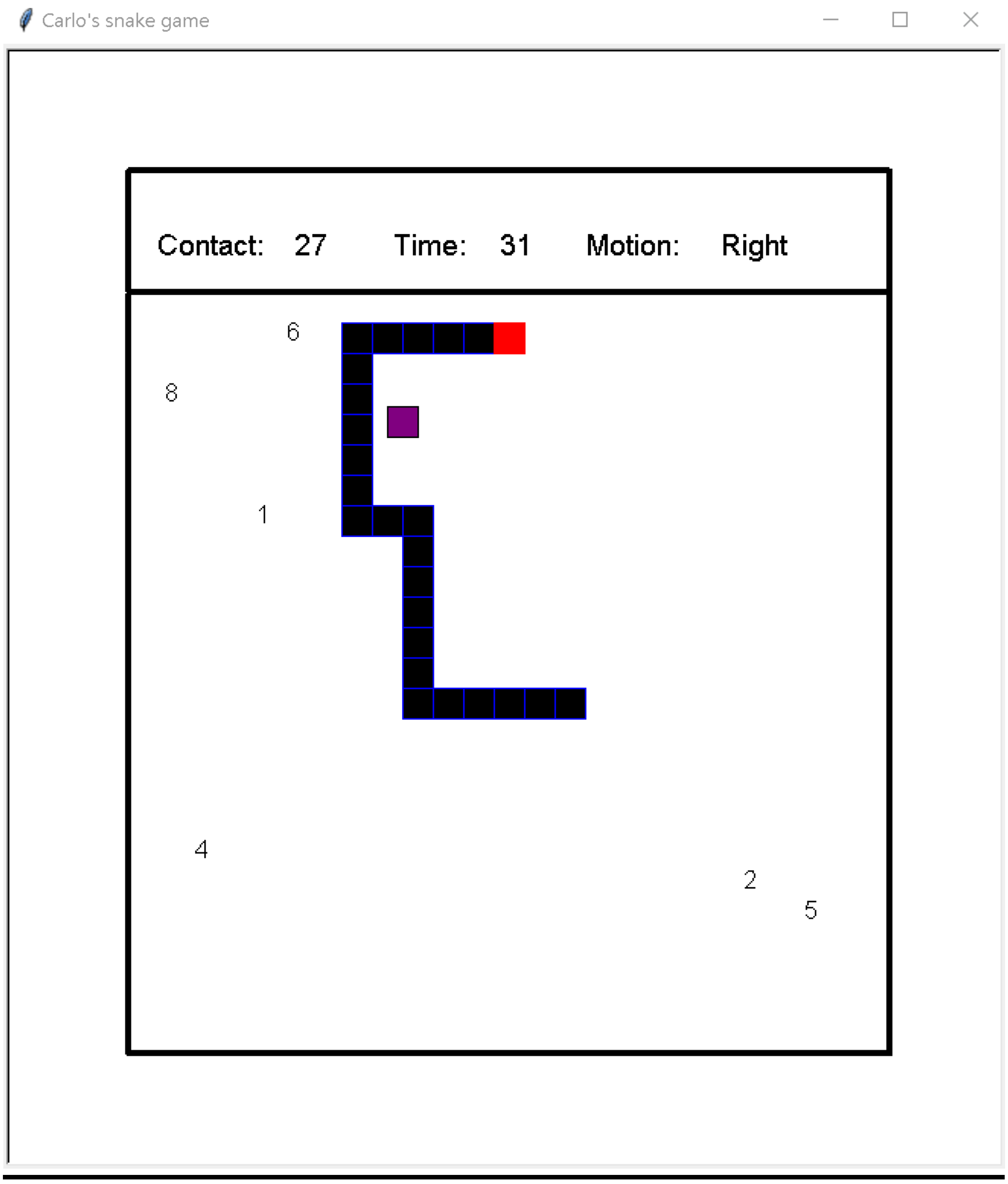
2. Game over:


Carlo's snake game
Contact: 5
Time: 9
Motion: Up
7
9
8
3
Game over!!
6
5
1
2

3. with 0 food consumed:



4. with 3 food consumed:



Contact: 27 Time: 31 Motion: Right