1. Design

1. Describe your thoughts and overall approach, including the breakdown of your logic into sub components and how these sub components are related/connected (basically the structure of your program)

Create monster and snake using Turtle. Create “w”(screen) using Screen(). Put monster in a random initial position. Put snake at origin. Show the startup instruction at the top of the screen.

onclick -> (1) clear the startup instruction (2) window’s title (3) set food (4) game begins.

Window’s title: Use ontimer(func, 1000) to change title with time flowing.

Set food: Create 9 food(turtles) use random position.

Game begins:

(1) Monster begins to move. (1) Suitable and random moving direction and speed. (2) Body contact with snake. (3) Whether game over or not.

(2) User use 4 raw keys to control snake motion. Once the snake move, judge (1) body contact and (2) eat food (3) snake status [“run”/ ”pause”/ “hit\_boundary”/ “eat\_food”/ “game\_over”/ “win”]

(3) Food will be clear if it be eaten. (1) Clear certain food(Turtle).(2) the matched number and (3) lengthen the snake’s body. (4) Slower the snake’s moving speed. (5) Whether win or not.

1. Describe the data types that you use to develop your program for tracking the various game objects (snake, monster and food items)

**Snake:**

(1) snake\_len[int],

(2) snake\_speed\_timer (use to change snake’s speed),

(3) state[str] (“run”/ ”pause”/ “hit\_boundary”/ “eat\_food”/ “game\_over”/ “win”),

(4) snake\_body[list] (contains multiple tuples which imply the snake’s body(squares) position)

(5) eat\_food[func]

(6) hit\_boundary[func]

(7) pause/ unpause[func]

(8) move\_snake[func]

**Monster:**

(1) body\_contact[int] (Once monster move, count body contact with snake),

(2) game\_over[func] (head on contact with snake)

(3) collapse[func] (Body contact with snake)

(4) monster\_move

**Food items:**

(1) f\_pos[dict] (key: food’s position (x,y), value: (“food\_i”<str>, i<str>), i = 1~9

(2) food[dict] (key: “food\_i<str>”, value: food <turtle>)

(3) win[func] (food is empty -> win)

1. Describe the motion logic for both snake and monster.

**Both use ontimer() to handle moving speed.**

**Snake:**

(1) User’s control. <moving direction, pause/unpause>.

(2) State tracking. <**state = “run”**, snake keep moving. During snake moving, it should estimate whether it hit the boundary and whether it eat food; **state = “game\_over”**, stop and show “Game Over”; **state = “win”**, stop and show “WIN”; **state = “hit\_boundary”**, set existing key\_pressed = None, the get new key and set state = “run” again so that the snake can move in a proper direction>

(3) extend body and clear the last stamp to move the snake. <Use stamp() to extend the snake. To change different color, it should get the color of head first and then change the color of snake, when stamp() is done, change the head’s color back. When the number of stampItems is bigger than body length, clear the last stamp. HINT: extend and clear stamps should in the same speed, say the speed of snake.>

**Monster**:

(1) Just move the single square. forword()

(2) Suitable random speed. ontimer(func, random.randint(\*, \*))

(3) Suitable and random moving direction. snake(x\_s, y\_s), monster(x\_m, y\_m)-> compare |x\_s–x\_m| and |y\_s–y\_m| to get moving trend -> forword()

1. Describe the expansion logic for the snake tail.

**func: extend\_snake(length=5, heading=0), clear\_extend()**

Use **stamp()** to extend the snake. To **change different color**, it should get the color of head first and then change the color of snake, when stamp() is done, change the head’s color back. **To control snake’s body length(make it move)**, when the number of stampItems is bigger than body length, clear the last stamp. HINT: extend and clear stamps should in the same speed, say the speed of snake.

(1) snake’s original length is 5

(2) eat food extend and slow down. e.g. snake eat “5”, then the parameter is 10. But the existing body length is 5, so the snake stamps (while the snake extending its tail, snake\_speed\_timer + 6 [slower the snake]) until the snake’s body length > 10 and begin to clear the last stamp.

(3) parameter “heading” depends on users’ control.

1. Describe the body contact logic between the monster and the snake.

fun: extend\_snake() and clear\_extend(), collapse()

collapse(pos1, pos2) pos1 and pos2 are tuples.

snake\_body[list], body\_contact[int]

(1) Inside extend\_snake(), the original head’s position will be appended into snake\_body[list] after it becomes a part of body. Inside clear\_extend(), snake\_body[list] will remove the first item. In conclusion, **snake\_body[list] always contains the existing body’s position.**

(2) collapse(pos1, pos2) will be called when monster move. pos1: monster’s current position. pos2: Iterate every items in snake\_body[list]

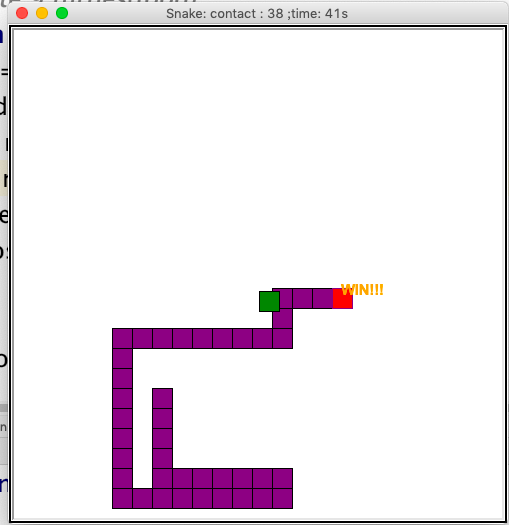
(3) If they collapse, means the two position[(x1, y1), (x2, y2)] |x1-x2| and |y1-y2| both smaller than the square’s side length. body\_contact + 1

2. Functions  
a. Describe usage of all your newly defined functions, including details of parameter(s)

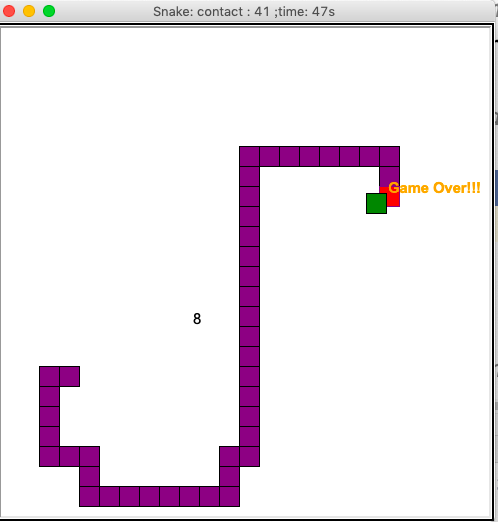
|  |  |  |
| --- | --- | --- |
| create\_screen() | Create screen 500 x 500, turn off auto refresh |  |
| get\_side\_len() | Get square’s side length |  |
| create\_turtle(shape, color, x=0, y=0) | Create monster or snake | (x, y) position |
| write(tt, txt, x, y, ft="normal") | Write “Game Over”, “WIN”, instruction, food number | tt: turtle; (x, y): position. ft: “normal”(default) or “bold” |
| set\_food() | Create 9 turtles(food). (use food[dict] to store them) |  |
| eat\_food() | Whether the snake eat food. Clear eaten food. change f\_pos[dict] and food[dict] |  |
| on\_key\_pause() | Press "space" to pause. |  |
| on\_key\_unpause() | Unpause when the second "space" was pressed. |  |
| on\_click(x, y) | Click everywhere to begin game-> set\_food(), begin\_game() |  |
| title() | Change the window’s title every second. (global variable) time + 1 |  |
| begin\_game() | ontimer(title, 1000), monster\_move(), get\_key(), move\_snake() |  |
| collapse(pos1, pos2) | Return bool | pos1, pos2 are both tuples |
| game\_over() | collapse(pos1, pos2) | pos1: monster pos, pos2: snake’s head pos |
| monster\_pos() | Get random initial pos of monster. Return (pos) x, y |  |
| monster\_move() | Get suitable direction of monster. game\_over(), collapse() |  |
| move\_up();move\_down(); move\_right();move\_left() | use extend\_snake(length, heading) with different heading |  |
| hit\_boundary() | Whether the snake hit boundary |  |
| move\_snake() | onkey(on\_key\_pause, “space”). move\_up();move\_down(); move\_right();move\_left(). eat\_food(), hit\_boundary(), |  |
| current\_key\_pressed(key) | Each time change key pressed, change the state of key\_pressed | key: current pressed one |
| get\_key() | onkey(),current\_key\_pressed(key) |  |
| clean\_extend() | Clear stamps like queue to move snake |  |
| extend\_snake(length=5, heading=0) | Extend the body of snake. clean\_extend() | length: snake’s body length; heading: depends on user |

3. Output  
a. Show samples of output (including status) from your program, including

i. Winner

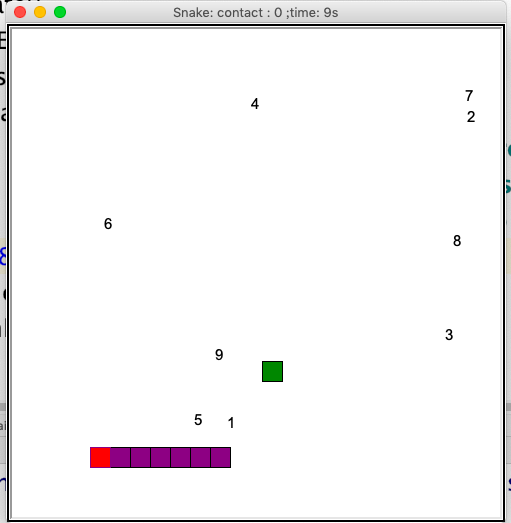


ii. Game over



iii. 2 others showing various stages of the game:

1. With 0 food item consumed



1. With 3 food items consumed

