**Coding Time: 2 days**

# Mission

Your mission is to design and implement a 2D snake game in your favorite programming language (front-end or back-end).

## **Details**

Rules of the game are as follows:

1. The game space is represented by a rectangle. The dimensions of this space should be received by your program as an input parameter before the game start e.g. width=800, height=600.
2. Each point, in (x, y) coordinates, within the game space represents the possible location of a player’s snake. For example, if your game space is represented by width=800(width) and height=600, then the possible locations for any player are within (0,0) to (799, 599) where (0,0) represents the top left and (799, 599) represents the bottom right.
3. At start of the game, the game should initialize the player’s snake location to somewhere near the center of the game space.
4. Each snake will have a head. Snake will always move in direction of the head. In the beginning, the snake’s length will be just one point. However, during the game the snake will need to eat the apples randomly appearing within game space one at a time. Each time the snake eats an apple, it’s length will grow by one point.   
     
   For example, assume the snake’s location is (500, 300) and the apple’s location is (501, 300). When snake eats this apple by moving right, it’s length will grow to 2. In this case, the snake will be represented by 2 points as follows:  
   (501, 300) <- (500, 300) where (501, 300) represents the head and (500, 300) represents the tail.  
     
   Now assume another apple appears at location (250, 450). If snake eats this apple coming from upwards -> downwards, it’s length will grow to 3. In this case, the snake will be represented by 3 points as follows:  
   (250, 450) <- (250, 449) <- (250, 448) where (250, 450) represents the head and (250, 448) represents the tail.  
     
   In general, if the snake eats n apples, its length will grow to n+1. Each of these n+1 points will represent the snake’s body.
5. At start of the game, the game will also show an apple for snake to eat. The apple’s location will appear randomly within the game space. This random location should not overlap with the snake’s location.
6. The snake eats an apple when the location of snake’s head arrives at the apple’s location. After that, the snake’s length is increased by 1.
7. Pressing the enter button should start the game.
8. When the game starts, the player could move his snake using the arrow keys. Pressing the arrow key once will start moving the player’s snake in that direction every second. For example, if the initial location is (400, 300) and if player presses the left key, the snake will start moving in left direction. In this case, after 1 second, his location will be changed to (399, 300). After 2 seconds, it will become (398, 300). After 3 seconds, it will become (397, 300) and so on.
9. Since the snake’s movement is controlled by arrow keys, the game will have a memory of latest 4 key presses and implement those. For example, if a player starts pressing the arrow keys randomly in a very quick fashion, the game will only implement the latest 4 keys he pressed, and the rest of the keys will be ignored.
10. The game will require the snake to eat a total of 15 apples. These apples will appear one by one meaning there will always be a single apple in game space for the snake to eat. Only when the snake has eaten it, the next apple will appear.
11. The snake will be required to eat all 15 apples within a given time period in minutes. This time period should be takes as user input before the game starts e.g. time=15 minutes.
12. During the time period of the game, snake will have total 3 chances. The snake will die in following cases:  
    a. If snake’s head crosses the border of the game. For example, if your game space is represented by width=800(width) and height=600, and if location of snake’s head is (799, 10) and if snake moves towards right, it will die.   
    b. If snake’s head touches its own body. Remember the snake will grow as it eats the apples meaning its maximum length will become 15 points on gaming space. In this case, if the snake is moving in a way, that its head’s location becomes equal to any other locations of its body, then the snake will die.
13. If the snake cannot eat all 15 apples within the given time period, the player loses, else he wins.
14. As the length of the snake grows, because player can move the snake in any direction, it’s a given that snake’s location might end up being not on a straight line, but rather in a ladder like manner. For example, imagine if snake’s length is 4, the location of its head is currently at (300, 300), at present it’s moving upwards so that all of its 4 point locations are represented by a straight vertical line as follows:  
    (300, 300) <- (300, 301) <- (300, 302) <- (300, 303)  
    Now if the player presses left key, then up key, then left key and then down key, then the next second the snake’s location will be as follows (implementing the left key action):

(299, 300) <- (300, 300) <- (300, 301) <- (300, 302)

After another second, the snake’s location will be as follows (implementing the up key action):  
(299, 299) <- (299, 300) <- (300, 300) <- (300, 301)

After another second, the snake’s location will be as follows (implementing the left key action):  
(298, 299) <- (299, 299) <- (299, 299) <- (299, 300)

After another second, the snake’s location will be as follows (implementing the down key action):  
(298, 300) <- (298, 299) <- (299, 299) <- (299, 299)

### **Guidelines**

1. Try to design and implement your game in a well-organized object-oriented way where different concepts are represented by corresponding classes (e.g. game space, snake, apple etc.). A good object-oriented approach might allow for extending the game later to multiple players.
2. Try to use the right data structures where needed e.g. queue and/or stack and/or linked list might help to implement the game in the right way.
3. Write good amount of comments in your code so that it’s readable and easy to understand.
4. Before delivering, try to test your code well using different possible scenarios.