

10.11.2016

## Group No 7

**Aashaka Shah** (14114001)

**Abhishek Jaisingh** (14114002)

**Akashdeep Goel** (14114004)

**Akshit Kalra** (14114006)

**Ambar Zaidi** (14114009)

## Problem Statement

This project aims to build a snake game which satisfies following requirements clearly :

1. Snake can enter a wall and emerge from the other (opposite) side.
2. Speed and size of snake must increase on eating a unit of food, thus increasing difficulty as the game progresses.
3. Game gets over as soon as snake touches its own body.



## Main Objects

### 1. Snake

- a. Attributes of Snake object - Length (Size) => 1 (Initially)
- b. Snake starts at bottom-left corner or grid-point (0, 0).
- c. Snake's length increases as it consumes more food.
- d. Snake's speed increases according to the equation:
  - i.  $\text{Speed} = 4.6 + \text{Score} * 0.66$
  - ii. Linear variation

### 2. Food

- a. Single unit increases the length of snake by 1.
- b. Each food unit appears at random locations on the grid.
- c. Food cannot appear on any of the cells currently occupied by snake.

### 3. Grid

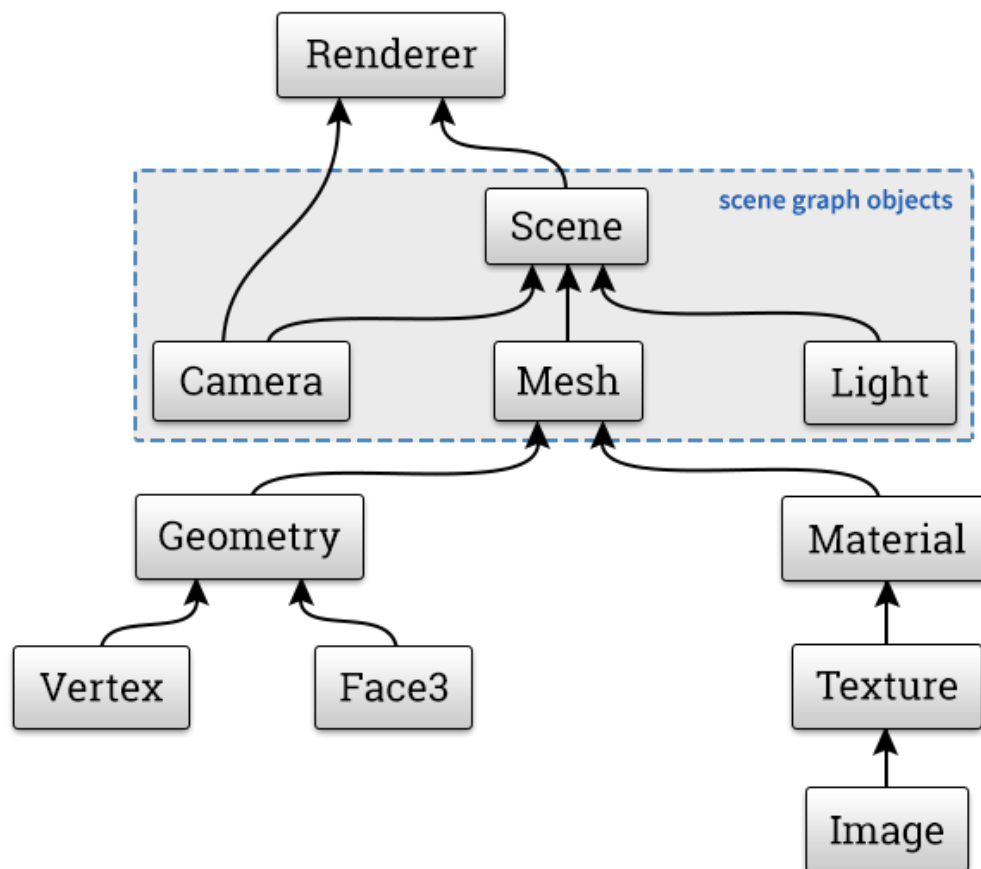
- a. Grid size can be adjusted.
- b. Number of rows and columns can be adjusted.
- c. Length and Width of columns can also be adjusted.

## Technologies Used:

- 1. HTML & CSS (for Frontend)
- 2. JavaScript (three.JS Library)

## three.js Basic Workflow

Three.js is a cross-browser JavaScript library/API used to create and display animated 3D computer graphics in a web browser. Three.js uses **WebGL**. WebGL in turn is based on **OpenGL ES 2.0** and provides an API for 3D graphics.



Everything made using three.JS library has three basic elements:

1. **Camera:** Used to define view point, front-plane, back-plane and aspect-ratio and viewing angle. Camera can be Perspective or Parallel.
2. **Mesh:** It is the basic form of any object which in turn is made up of geometry and material.
3. **Light:** Light sources must also be defined appropriately otherwise the object cannot be seen clearly. There are various kinds of light sources like Ambient Light, Directional Light etc.



## Outline of Steps taken:

1. Create the Scene
2. Create the Renderer
3. Create a Camera
4. Add Lighting
5. Add Meshes(Geometry+Material) to Scene
6. Add Controls
7. Render the Scene

## Algorithms & Data Structures

### Data Structure

We needed to model the snake using an appropriate data-structure, which fulfils the following requirements:

1. Maintains length dynamically, i.e. length should not be constant.
2. When no food is eaten, length must remain same, but snake must move forward.
  - a. A new cube must be added at the front of current snake.
  - b. The last cube must be removed.
3. If food is eaten by the snake, length should be increased by one.
  - a. Here we need only to add a new cube at front of snake.
  - b. No removal is required as the length is supposed to increase by one unit.

As only the head and tail of the snake needs to be updated, we require a data structure that supports the removal and addition at its two ends. So, we must use queues. But, we need to support these addition and removal operations at both ends.

The data structure that supports all these requirements is **Deque** (Double Ended Queue). In deque, elements can be added to or removed from either the front (head) or back (tail) efficiently and easily.

## Deque General JavaScript API

Operation	Common name(s)	JavaScript
insert element at back	inject, snoc	push
insert element at front	push, cons	unshift
remove last element	eject	pop
remove first element	pop	shift
examine last element	peek last	<obj>[<obj>.length - 1]
examine first element	peek first	<obj>[0]


## Algorithm / Pseudocode:

There are basically two main functions having some algorithmic logic:

---

```
Function getRandomFreeLocation():
    x = random(0, rows)
    y = random(0, cols)
    for i in (0, rows):
        for j in (0, cols):
            if ((x + i ) % rows, (y + j )% cols) not in queue:
                return ((x + i ) % rows, (y + j )% cols)
```

---



```
Function MainRenderLoop():
    takeInput();
    IF collision(positionX, positionY):
        Display Score
        OUTPUT("GAME OVER")
        Stop RenderLoop

    dequeue.push(newcube)
    IF foodEaten(positionX, positionY):
        SCORE = SCORE + 1
        food.location = getRandomFreeLocation()
    ELSE:
        dequeue.shift()
    calculateNewPosition(positionX, positionY);
```

---

## Snapshots

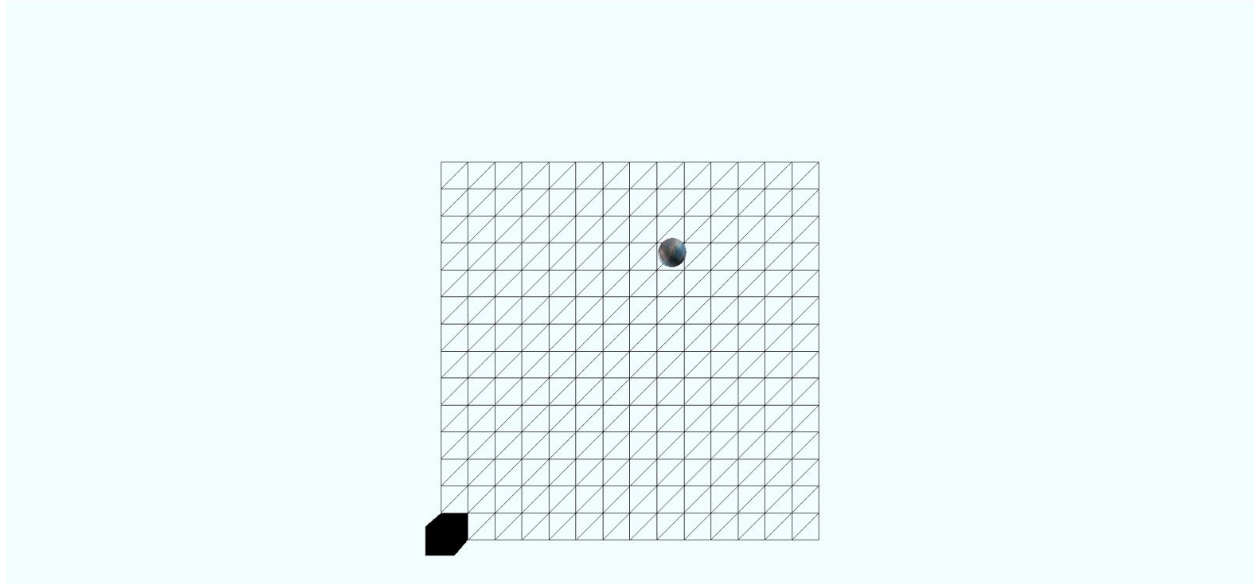
### Start of Game

Snake starts from bottom-left corner of grid with a score of zero.

Initial Score = 0

# Snake Game

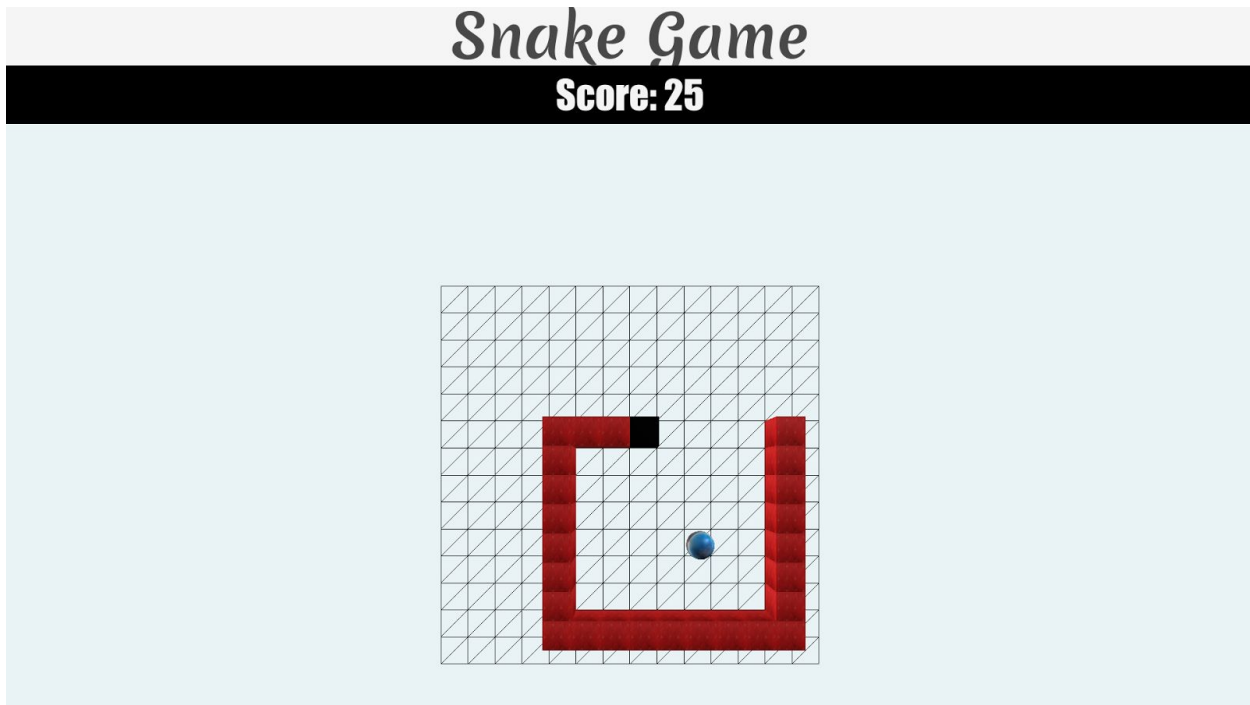
Score: 0



## Growth of Snake

As the snake eats more food, its length and speed increases. Speed has been modeled as a linear function of length / score.

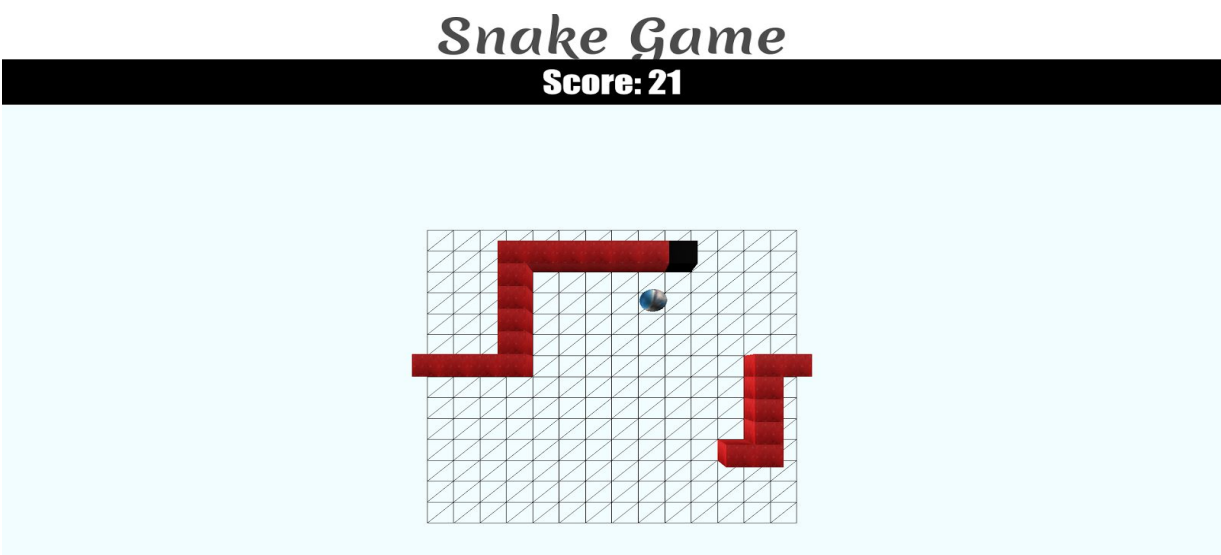
**Score = 25**



## Emergence from other side of wall

A snake can enter the wall and emerge from the opposite side.

**Score = 21**



## Game Over

When snake touches any part of its own body, game is over and final score is displayed.

**Final Score = 27**





# Snake Game

GAME OVER, Final Score: 27

