

 **Global Academy of Technology**



**Department Of Electronics and Communication Engineering**

**Report**

**On**

# JAVA MINI PROJECT

**IV Semester**

**Academic Year: 2018-2019**

**Group No.:6**

# TITLE: SNAKE GAME USING JAVASCRIPT

|  |  |
| --- | --- |
| **USN** | **NAME** |
| **1GA17EC042** | **GOWRAV S** |
| **1GA17EC041** | **GOKUL SAI R** |
| **1GA17EC049** | **HARSHITHA H R** |
| **1GA18EC406** | **NANDAN KUMAR N** |

**AIM / Objective of the Project:**

To design a snake game using JAVASCRIPT and HTML. And create a web page on which game runs.

**SYSTEM REQUIREMENTS:**

Visual Studio Code Complier,JavaScript,CSS,HTML, Chrome or any web browser to launch the game.

**INTRODUCTION:**

*Snake* is an older classic video game which was first created in late 70s. Later it was brought to PCs. In this game the player controls a snake. The objective is to eat as many apples as possible. Each time the snake eats an apple, its body grows.

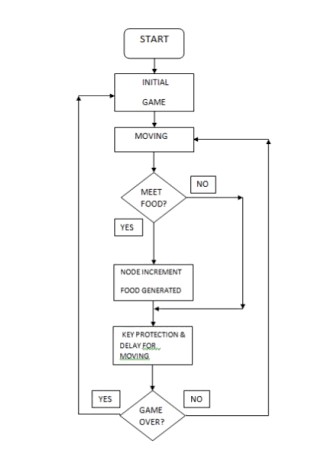
The following is an example game written in JAVASCRIPT based on the game called snake which has been around since the earliest days of home computing and has re-emerged in recent years on mobile phones.

It isn't the world's greatest game, but it does give you an idea of what you can achieve with a relatively simple JAVASCRIPT program, and perhaps the basis by which to extend the principles and create more interesting games of your own.

**C0NCEPT OF PROJECT:**

1. To reinforce many of the JAVASCRIPT and programming concepts you may have already met.
2. To provide valuable experience of the design and implementation of a large program.
3. To provide a framework for a more challenging, and thus rewarding, laboratory exercise.
4. To move the snake, use **‘up arrow'** for up, **‘down arrow'** for down, **‘left arrow'** for left and **‘right arrow'** for right as well **as ‘w’** for front, **‘s’** for backward, **‘a’** for left, **‘d’** for right. Again, there are constants you can change if you want to alter these settings.
5. Press close the tab to exit the game at any time.
6. The aim of the game is to collect the dots (food) and avoid the obstacles (crosses, borders, and the snake itself). As you collect food, the snake gets longer, so increasing your likelihood of crashing into yourself.
7. When you have collected enough food, you progress onto the next level, where your snake gets longer, and the amount of food to collect to progress through the level gets larger.
8. You get scored according to the length of the snake and the number of 'x' obstacles on the screen.
9. The speed increases every 5 level. You get a bonus when you complete the level of 1000, increasing by 1000 each level (e.g. complete level 5, you get a 5000 bonus). There is no concept of lives. Once you hit an obstacle, that's it, game over.  Make sure you do not have the caps lock on, otherwise the keys will fail to respond

**FLOWCHART:**



**PROGRAM:**

The program is divided into 3 sun programs and combined and linked using HTML.

The programs are as follows:

**1.FILE NAME:** snake.js

|  |
| --- |
| const base = require('./base')  Object.getOwnPropertyNames(base).map(p => global[p] = base[p])  // Constants const NORTH = { x: 0, y:-1 } const SOUTH = { x: 0, y: 1 } const EAST = { x: 1, y: 0 } const WEST = { x:-1, y: 0 }    // Point operations const pointEq = p1 => p2 => p1.x == p2.x && p1.y == p2.y  // Booleans const willEat = state => pointEq(nextHead(state))(state.apple) const willCrash = state => state.snake.find(pointEq(nextHead(state))) const validMove = move => state => state.moves[0].x + move.x != 0 || state.moves[0].y + move.y != 0  // Next values based on state const nextMoves = state => state.moves.length > 1 ? dropFirst(state.moves) :  state.moves const nextApple = state => willEat(state) ? rndPos(state) : state.apple const nextHead = state => state.snake.length == 0  ? { x: 2, y: 2 }  : { x: mod(state.cols)(state.snake[0].x + state.moves[0].x), y: mod(state.rows)(state.snake[0].y + state.moves[0].y)  } const nextSnake = state => willCrash(state)  ? []  : (willEat(state)  ? [nextHead(state)].concat(state.snake) |
| : [nextHead(state)].concat(dropLast(state.snake)))    // Randomness const rndPos = table => ({ x: rnd(0)(table.cols - 1), y: rnd(0)(table.rows - 1)  })    // Initial state const initialState = () => ({ cols: 20, rows: 14, moves: [EAST], snake: [], apple: { x: 16, y: 2 },  }) const next = spec({ rows: prop('rows'), cols: prop('cols'), moves: nextMoves, snake: nextSnake, apple: nextApple  }) const enqueue = (state, move) => validMove(move)(state) ? merge(state)({ moves: state.moves.concat([move]) })  : state  module.exports = { EAST, NORTH, SOUTH, WEST, initialState, enqueue, next, } |

**2.FILE NAME:** web.js

|  |
| --- |
| const canvas = document.getElementById('canvas') const ctx = canvas.getContext('2d')    // Mutable state  let state = initialState()    // Position helpers const x = c => Math.round(c \* canvas.width / state.cols) const y = r => Math.round(r \* canvas.height / state.rows)  // Game loop draw const draw = () => {  // clear ctx.fillStyle = '#232323' ctx.fillRect(0, 0, canvas.width, canvas.height)    // draw snake ctx.fillStyle = 'rgb(0,200,50)' state.snake.map(p => ctx.fillRect(x(p.x), y(p.y), x(1), y(1)))  // draw apples ctx.fillStyle = 'rgb(255,50,0)' ctx.fillRect(x(state.apple.x), y(state.apple.y), x(1), y(1))  // add crash if (state.snake.length == 0) { ctx.fillStyle = 'rgb(255,0,0)' ctx.fillRect(0, 0, canvas.width, canvas.height)  }  }    // Game loop update const step = t1 => t2 => { if (t2 - t1 > 100) { state = next(state) draw() window.requestAnimationFrame(step(t2))  } else { window.requestAnimationFrame(step(t1))  }  } |
| // Key events window.addEventListener('keydown', e => { switch (e.key) { case 'w': case 'h': case 'ArrowUp': state = enqueue(state, NORTH); break case 'a': case 'j': case 'ArrowLeft': state = enqueue(state, WEST); break case 's': case 'k': case 'ArrowDown': state = enqueue(state, SOUTH); break case 'd': case 'l': case 'ArrowRight': state = enqueue(state, EAST); break }  })  // Main draw(); window.requestAnimationFrame(step(0)) |

**3.FILE NAME:** base.js

|  |
| --- |
| const adjust = n => f => xs => mapi(x => i => i == n ? f(x) : x)(xs) const dropFirst = xs => xs.slice(1) const dropLast = xs => xs.slice(0, xs.length - 1) const id = x => x const k = x => y => x const map = f => xs => xs.map(f) const mapi = f => xs => xs.map((x, i) => f(x)(i)) const merge = o1 => o2 => Object.assign({}, o1, o2) const mod = x => y => ((y % x) + x) % x // http://bit.ly/2oF4mQ7 const objOf = k => v => ({ [k]: v }) const pipe = (...fns) => x => [...fns].reduce((acc, f) => f(acc), x) const prop = k => o => o[k] const range = n => m => Array.apply(null, Array(m - n)).map((\_, i) => n + i) const rep = c => n => map(k(c))(range(0)(n)) const rnd = min => max => Math.floor(Math.random() \* max) + min const spec = o => x => Object.keys(o)  .map(k => objOf(k)(o[k](x)))  .reduce((acc, o) => Object.assign(acc, o))  module.exports = { adjust, dropFirst, dropLast, id, k, map, merge, mod, objOf, pipe, prop, range, rep, rnd, spec } |

**FILE NAME :** snake2.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Snake</title>

<link rel="stylesheet" href="web.css">

<script>require=()=>({}); module={}; module.exports=()=>({})</script>

</head>

<body>

<canvas id="canvas" width="700" height="500"></canvas>

<script src="base.js"></script>

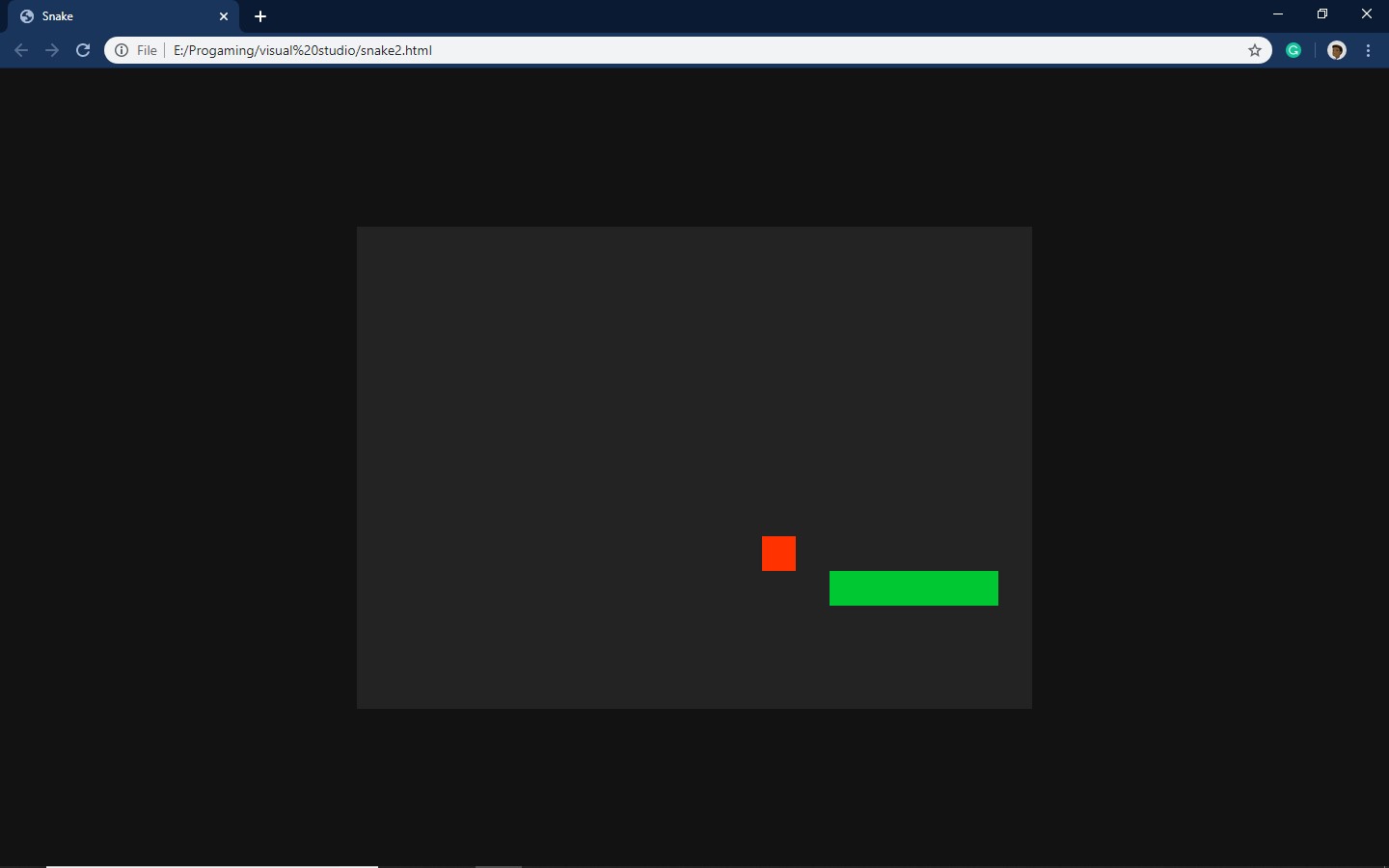
<script src="snake.js"></script>

<script src="web.js"></script>

</body>

</html>

**Simulated Output:**



Page No.

**Acknowledgement:**

We Gowrav S, Gokul Sai R ,Harshitha.H.R and Nandan Kumar are thankful to you Dileep Kumar Sir,for giving us an opportunity to showcase our project on JAVA and its application and understand the concept to its full extent. Thank you Sir.



Page No.