SNAKE GAME

A COURSE PROJECT REPORT

By

SAMBHRAM GHOSH(RA2011031010036)

Under the guidance of

DR.M.ANAND SIR

In partial fulfilment for the Course

of

18CSE361T- WEB PROGRAMING

in DEPARTMENT OF Networking and Communications

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chengalpattu District

NOVEMBER 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report “SNAKE GAME ” is the bonafide work of SAMBHRAM GHOSH(RA2011031010036) who carried out the project work under my supervision.

SIGNATURE

DR.M.ANAND SIR
Associate Professor
Department of Networking and Communications
SRM Institute of Science and Technology

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable Vice Chancellor

Dr. C. MUTHAMIZHCHELVAN, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our Registrar

Dr. S. Ponnusamy, for his encouragement

We express our profound gratitude to our Dean (College of Engineering and Technology) Dr. T. V.Gopal, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School

of Computing Dr. Revathi Venkataraman, for imparting confidence to complete my course project

We wish to express my sincere thanks to Course Audit Professor Dr.Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications and Course Coordinators for their constant encouragement and support.

We are highly thankful to our my Course project Faculty DR.M.ANAND, Associate Professor, Department of Networking and Communications for her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our HoD Dr. Annapurani Panaiyappan, Professor and HOD, Department of Networking and Communications and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who

directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

INTRODUCTION

Snake is a video game that originated during the late 1970s in arcades becoming something of a classic. It became the standard preloaded game on Nokia phones in 1998. The player controls a long, thin creature, resembling a snake, which roams around on a bordered plane, picking up food (or some other item), trying to avoid hitting its own tail or the edges of the playing area. Each time the snake eats a piece of food, its tail grows longer, making the game increasingly difficult. The user controls the direction of the snake's head (up, down, left, or right), and the snake's body follows. Here is a quick overview of how it is designed. In the snippets below, DOT_SIZE is the width (and height) of the section of the snake, say 20 pixels. (You can think of this as something similar to a square on a game board.) MAX_DOTS constant defines the maximum number of possible dots on the board. The size 600x400 defines the size of the board in pixels, so dividing that number by DOT_SIZE*DOT_SIZE computes the number of DOTS (i.e. squares) on the board. Going forward, a part of the snake will occupy one of these dots. This was the overall idea.

IMPLEMENTATION

Filename: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Web Programming Project-Snake</title>
  <style>
    body {
      height: 100vh;
      width: 100vw;
      display: flex;
      justify-content: center;
      align-items: center;
      margin: 0;
      background-color: black;
    }

    #game-board {
      background-color: #CCC;
      width: 100vmin;
      height: 100vmin;
```

```
      display: grid;
      grid-template-rows: repeat(21, 1fr);
      grid-template-columns: repeat(21, 1fr);
    }

    .snake {
      background-color: hsl(200, 100%, 50%);
      /*border: .25vmin solid black;*/
    }

    .food {
      background-color: hsl(50, 100%, 50%);
      /*border: .25vmin solid black;*/
    }
  </style>
  <script src="game.js" defer type="module"></script>
</head>
<body>
  <div id="game-board"></div>
</body>
</html>
```

Filename: snake.js

```
import { getInputDirection } from "./input.js"

export const SNAKE_SPEED = 5
const snakeBody = [{ x: 11, y: 11 }]
let newSegments = 0

export function update() {
  addSegments()

  const inputDirection = getInputDirection()
  for (let i = snakeBody.length - 2; i >= 0; i--) {
    snakeBody[i + 1] = { ...snakeBody[i] }
  }

  snakeBody[0].x += inputDirection.x
  snakeBody[0].y += inputDirection.y
}

export function draw(gameBoard) {
  snakeBody.forEach(segment => {
    const snakeElement = document.createElement('div')
    snakeElement.style.gridRowStart = segment.y
    snakeElement.style.gridColumnStart = segment.x
    snakeElement.classList.add('snake')
    gameBoard.appendChild(snakeElement)
  })
}

export function expandSnake(amount) {
  newSegments += amount
}
```

```
export function onSnake(position, { ignoreHead = false } = {}) {
  return snakeBody.some((segment, index) => {
    if (ignoreHead && index === 0) return false
    return equalPositions(segment, position)
  })
}

export function getSnakeHead() {
  return snakeBody[0]
}

export function snakeIntersection() {
  return onSnake(snakeBody[0], { ignoreHead: true })
}

function equalPositions(pos1, pos2) {
  return pos1.x === pos2.x && pos1.y === pos2.y
}

function addSegments() {
  for (let i = 0; i < newSegments; i++) {
    snakeBody.push({ ...snakeBody[snakeBody.length - 1] })
  }

  newSegments = 0
}
```

Filename: food.js

```
import { onSnake, expandSnake } from './snake.js'
import { randomGridPosition } from './grid.js'

let food = getRandomFoodPosition()
const EXPANSION_RATE = 5

export function update() {
  if (onSnake(food)) {
    expandSnake(EXPANSION_RATE)
    food = getRandomFoodPosition()
  }
}

export function draw(gameBoard) {
  const foodElement = document.createElement('div')
  foodElement.style.gridRowStart = food.y
  foodElement.style.gridColumnStart = food.x
  foodElement.classList.add('food')
  gameBoard.appendChild(foodElement)
}

function getRandomFoodPosition() {
  let newFoodPosition
  while (newFoodPosition == null || onSnake(newFoodPosition)) {
    newFoodPosition = randomGridPosition()
  }
  return newFoodPosition
}
```

Filename: grid.js

```
const GRID_SIZE = 21


export function randomGridPosition() {
  return {
    x: Math.floor(Math.random() * GRID_SIZE) + 1,
    y: Math.floor(Math.random() * GRID_SIZE) + 1
  }
}


export function outsideGrid(position) {
  return (
    position.x < 1 || position.x > GRID_SIZE ||
    position.y < 1 || position.y > GRID_SIZE
  )
}
```

Filename: input.js

```
let inputDirection = { x: 0, y: 0 }
let lastInputDirection = { x: 0, y: 0 }

window.addEventListener('keydown', e => {
  switch (e.key) {
    case 'ArrowUp':
      if (lastInputDirection.y !== 0) break
      inputDirection = { x: 0, y: -1 }
      break
    case 'ArrowDown':
      if (lastInputDirection.y !== 0) break
      inputDirection = { x: 0, y: 1 }
      break
    case 'ArrowLeft':
      if (lastInputDirection.x !== 0) break
      inputDirection = { x: -1, y: 0 }
      break
    case 'ArrowRight':
      if (lastInputDirection.x !== 0) break
      inputDirection = { x: 1, y: 0 }
      break
  }
})

export function getInputDirection() {
  lastInputDirection = inputDirection
  return inputDirection
}
```

Filename: game.js

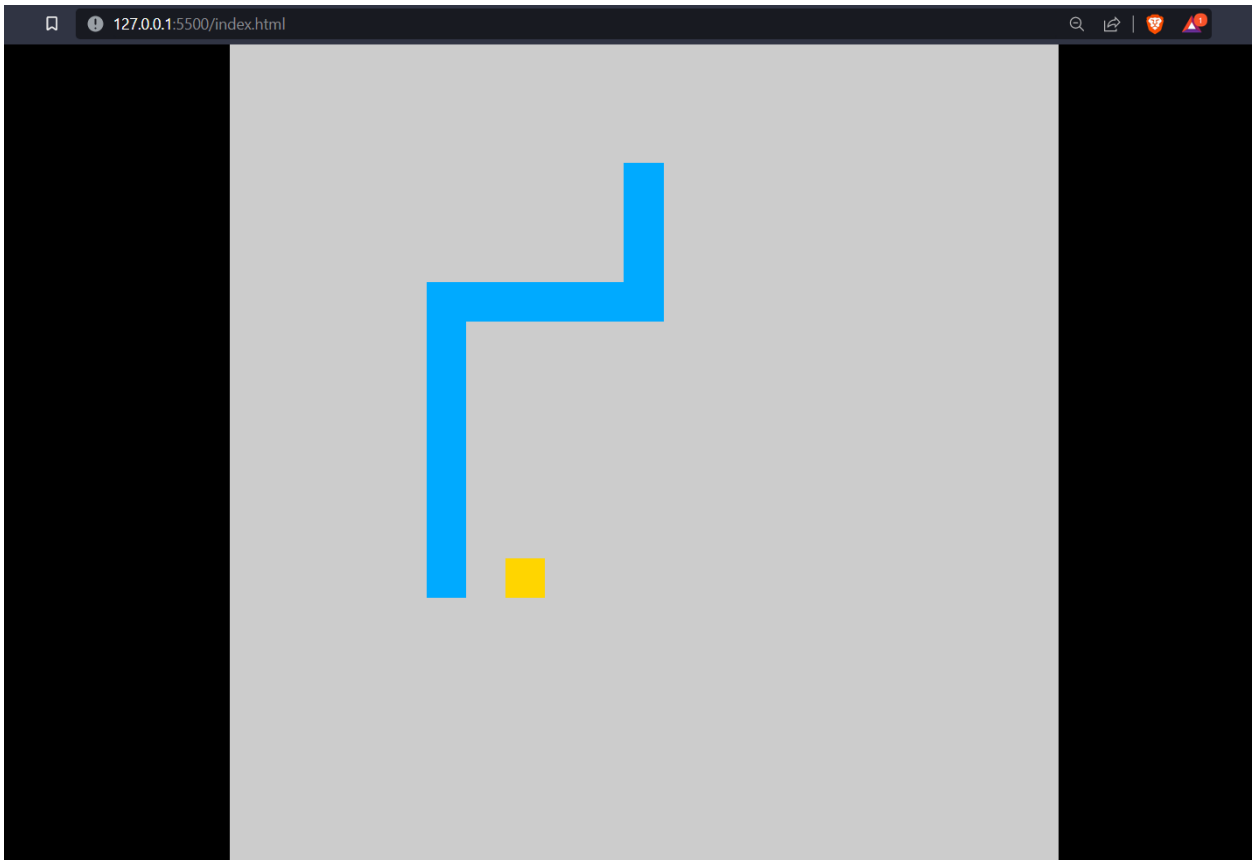
```
import { update as updateSnake, draw as drawSnake, SNAKE_SPEED,
getSnakeHead, snakeIntersection } from './snake.js'

import { update as updateFood, draw as drawFood } from './food.js'

import { outsideGrid } from './grid.js'


let lastRenderTime = 0

let gameOver = false

const gameBoard = document.getElementById('game-board')


function main(currentTime) {

  if (gameOver) {

    if (confirm('You lost. Press ok to restart.')) {

      window.location = '/'

    }

    return

  }




  window.requestAnimationFrame(main)

  const secondsSinceLastRender = (currentTime - lastRenderTime) / 1000

  if (secondsSinceLastRender < 1 / SNAKE_SPEED) return



  lastRenderTime = currentTime


  update()
```

```
  draw()
}


window.requestAnimationFrame(main)


function update() {
  updateSnake()
  updateFood()
  checkDeath()
}


function draw() {
  gameBoard.innerHTML = ''
  drawSnake(gameBoard)
  drawFood(gameBoard)
}


function checkDeath() {
  gameOver = outsideGrid(getSnakeHead()) || snakeIntersection()
}
```

RESULTS

Web Application : C:\Users\sambh\OneDrive\Desktop\Web Programming project\index.html

SCREENSHOTS OF PROJECT:



CONCLUSION

The coding of Snake was extremely difficult with many errors arising. Many systems had to be written numerous ways before a final working solution was found. For example, two different movement methods were used prior to final version; however, even the final version is flawed as vertical movement causes the snake to change scale. There were also issues with the food – snake collision detection. While the final version resulted in a snake that could eat food, the movement glitch caused the food to cause further size issues.

Despite the fact that the game could not truly be played due to the fact no score could be given, the game is still satisfying. With the exception of the size glitch when turning, the snake responds to user input and moves around the screen as directed. Given longer to work on this, the collision detection with the movement would be the first thing fixed. By fixing this, all other sections of code that are currently not working would run. The leaderboard would work as there would be correct scores input, and the snake would grow as the food would cause it to only increase by one and not varying numbers based on direction. In addition, fixing the movement would allow for the snake to die when colliding with itself. In the current state, the snake moves as a matrix so it can not kill itself as it would be impossible to move in any direction. This failure to establish a perfect movement system was the biggest disappointment of the game as all other problems stemmed from it.

For these reasons, it is recommended that anyone who wishes to recreate this game starts simply when writing the code. It is advisable that they first perfect the snakes movement controls before messing with the food generation. By taking the code in small sections, it is easier to get individual features to work. Building off this, use functions to contain each aspect of the game. Using functions made it easier to determine where errors were occurring when debugging the code. It also kept the code more organized.

REFERENCES:

- https://www.edureka.co/blog/snake-game-with-pygame/
- https://code-projects.org/snake-game-in-python-with-source-code-2/
- https://en.wikipedia.org/wiki/Snake_(video_game_genre)