

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of Bachelor of
Technology Degree in Information Technology

By
SHAURYA SAMEER
Registration No.: 39120110



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI-600119, TAMIL NADU

APRIL 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI

CHENNAI- 600119

www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **SHAURYA SAMEER (Reg. No: 39120110)** who carried out the project entitled "**Web Based 2D Multiplayer Game**" under my supervision from Feb 2022 to April 2022.

Internal Guide

Dr.Jeberson Retna Raj

Head of the Department

Dr. R. Subhashini

Submitted for Viva Voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **SHAURYA SAMEER** hereby declare that the project entitled “**Web Based 2D Multiplayer Games**” completed by me under the guidance of **Dr.Jeberson Retna Raj** is submitted in partial fulfilment of the requirements for the award of Bachelor in Technology Degree in Information Technology.

DATE:

SHAURYA SAMEER

PLACE:

DIGITAL SIGNATURE OF CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** at **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D, Dean**, School of Computing, **Dr. R. Subhashini, M.E., Ph.D. Head of the Department of Information Technology** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.Jeberson Retna Raj, M.E., Ph.D.,** for his valuable guidance, suggestions and constant encouragement that paved the way for the successful completion of my project work.

I wish to express my thanks to all teaching and non-teaching staff members of the **Department of Information Technology** who were helpful in many ways throughout the completion of the project.

ABSTRACT

Games innately seem difficult to build. With lots of moving pieces involved (quite literally), game development seems like it's restricted to developers who've immersed their feet deep into the holy waters of networking, mathematics, graphics, and the like. However, with the web evolving ever so rapidly and with all the new protocols and libraries that have spanned in recent years, it's now super simple to get started with building browser-based multiplayer games.

This study aims to build two 2D multiplayer browser-based games that can be played on any desktop out there. The first one is a fighting game with local multiplayer while the second one is a simple multiplayer snowball fighter game both made through JavaScript, HTML canvas, CSS and socket.io.

This project aims to address a particular trend in gaming and build a simple game that is playable on any device with any hardware specification whatsoever. JavaScript and HTML canvas offers easy access to building great games. They're easy to monetize and have a lot of potential in the current game market.

TABLE OF CONTENTS		
CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	i
	LIST OF FIGURES	ii
1	INTRODUCTION	1
1.1	ABOUT THE CONCEPT	1
1.2	ABOUT JAVASCRIPT AND OOPS	2
2	AIM AND SCOPE OF THE PRESENT INVESTIGATION	3
2.1	AIM	5
2.2	SCOPE	5
2.2.1	ACCESSIBILITY	6
3	EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED	7
3.1	HARDWARE/SOFTWARE REQUIREMENTS	7
3.1.1	WINDOWS/MAC/LINUX SYSTEM	7
3.1.2	ANY TEXT EDITOR LIKE VSCODE, SUBLIME TEXT, ATOM ETC.	7
3.2	SETTING UP	7
3.3	DEVELOPMENT OF THE GAME	8

3.3.1	CREATING INDEX.HTML AND INDEX.JS	8
3.3.2	CREATING NECESSARY CLASSES	8
3.3.3	CREATING INSTANCES OF OUR CLASS	10
3.3.4	PLAYER INSTANCE FROM FIGHTER CLASS	11
3.3.5	BUILDING OUR ANIMATION LOOP	12
3.3.6	BUILDING OUR KEY LISTENER	13
3.3.7	ADDING THE NECESSARY UI AND LINKING EVERYTHING WITH INDEX.HTML	14
3.3.8	MAKING SEPARATE FILES FOR UTILITY FUNCTIONS	15
3.3.9	SNOWBALL FIGHTING GAME	15
3.3.9.1	SETTING UP GITHUB/NPM AND INITIALISING INDEX.HTML	16
3.3.9.2	BUILDING LOGIC FUNCTIONS	17
3.3.9.3	BUILDING NECESSARY EVENT LISTENERS	18
3.3.9.4	CODING THE SERVER AND BUILDING OUR PACKAGE	19
4	HOSTING	20
4.1	HOSTING ON NETLIFY AND HEROKU	20
4.1.1	HOSTING 2D FIGHTING GAME ON NETLIFY	20
4.1.2	HOSTING SNOWBALL GAME ON HEROKU FROM GITHUB	22
5	SUMMARY AND CONCLUSIONS	23
5.1	FUTURE WORK	23

LIST OF FIGURES		
S. NO.	TITLE	PAGE NO.
1.1	Drawing a Rectangle using HTML Canvas	2
3.1	Sprite Class	8
3.2	Fighter Class	9
3.3	Rendering Canvas and making background sprites	10
3.4	Player Instance	11
3.5	Animation Loop Function	12
3.6	Key Listener Function	13
3.7	HTML UI and Render Code	14
3.8	Utility Function	15
3.9	Snowball game player render	16
3.10	Building Logic functions and collision detection	17
3.11	Event Listener and Socket API	18
3.12	Server Code	19
4.1	The game hosted on Netlify	20
4.2	Tie due to timer running out	21
4.3	Samurai winning by killing his opponent Tenguu	21
4.4	Hosted snowball game on heroku	22
	APPLICATION SNAPSHOTS	

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE CONCEPT

The HTML5 canvas allows us to use graphics on the screen, and draw and manage various shapes. The HTML Canvas 2D Context API (in mobile, wearable, and TV applications) defines a special canvas element that expresses images or shapes with JavaScript. The main features of the Canvas Element API include:

- Creating a canvas
- To draw and manage shapes, you must insert a `<canvas>` element in the HTML page.
- Using images and drawing shapes
- You can use images on the canvas by using the applicable method of the HTML Canvas 2D Context API.
- With the HTML Canvas 2D Context API, you can draw various shapes, such as rectangles, circles, and lines to a canvas.
- You can also draw and mask objects on the canvas.
- Using styles and transformations
- You can use a canvas to create text or lines other than images and shapes.

For all canvas objects (images, shapes, text, and lines), we can define colors (the `fillStyle` and `strokeStyle` attributes), shadows (the `shadowColor` and `shadowBlur` attributes), and gradation (the `createLinearGradient()` method). We can also use the transformation methods, such as `scale()`, `translate()`, `transform()`, and `rotate()`, to implement, for example, transparency or shape gradient transformations. In mobile applications only, in HTML5, the Scalable Vector Graphics (SVG) 2 API provides similar features as the canvas. Their difference is that SVG expresses graphics using vectors, while the canvas is based on pixels. To express complex graphics, use the canvas, and to express graphics with a liberal expansion or reduction, we use SVG.

```
const canvas = document.querySelector('#tester-canvas')
canvasSize()
const canvasContext = canvas.getContext('2d')
//
// This sets the canvas to the width and height of the window
function canvasSize() {
  canvas.width = window.innerWidth
  canvas.height = window.innerHeight
}
//
const drawRect = (x, y) => {
  canvasContext.fillStyle = '#00FFFF'
  canvasContext.fillRect(x, y, 50, 50)
}
drawRect(100,200)
drawRect(300,400)
```

Fig 1.1: Drawing a Rectangle using HTML Canvas

1.2 ABOUT JAVASCRIPT AND OOPs

JavaScript is the programming language of the web. It's easy to learn and becoming more and more prevalent in software development everyday (Netflix, PayPal, and Facebook all use a lot of JavaScript!). In EcmaScript 2015, also referred to as ES6, classes were introduced to JavaScript, which made it much easier for developers to think about and organize their programs. Also, JavaScript can run in any web browser, which means we just need a text editor!

JavaScript classes are a great way to make clean, reusable code and easily track the relationship between one object's properties & functions and another object's properties & functions. Objects have data and functions to manipulate data. The relationships and interactions between well-modelled objects will allow us to create systems of great complexity, and allow us to increase that complexity while keeping a maintainable and organized codebase.

Object modeling is a process that will organically break complex features down into granular components, as you'll see in our example project of creating a text role-playing game.

CHAPTER-2

AIM AND SCOPE OF THE PRESENT INVESTIGATION

The distributed Multiplayer Online Game system is complex since it involves technologies in computer graphics, multimedia, artificial intelligence, computer networking, embedded systems, etc. Due to the large scope of this problem, the design of these systems has not yet been widely addressed in the literatures. In this paper, we review and analyze the current MOG system architecture followed by evaluation. Furthermore, we propose a clustered-server architecture to provide a scalable solution together with the region-oriented allocation strategy. Two key issues, i.e., interesting management and synchronization, are discussed in depth. Some preliminary ideas to deal with the identified problems are described.

The distributed Multiplayer Online Game (MOG) service has grown very fast in recent years. Since the great success of several early multiplayer online games around mid-90's, such as DOOM and Quake from IDSoft [1], Warcraft from Blizzard [2], MOG is getting more and more investment from the entertainment industry. There are international tournament for several popular games. Besides the computer-based MOG, manufacturers of the traditional console-based video game launched online service. PS2 online and Xbox live were announced on 2001 and 2002, respectively. It is forecasted that about 114 million people worldwide are expected to play online games by 2006. Typically, MOG is a kind of distributed interactive real-time applications. Three classes are prominent among the applications with distributed interactive real-time attributes. They are military simulations, such as flight simulators or logistical simulations, Distributed Virtual Environments (DVE) or so-called Collaborative Virtual Environments (CVE), and the spotlighted multiplayer online games. However, the MOG system brings more problems than before due to its special characteristics, e.g., a large number of players, heterogeneous game platforms and disparity network bandwidth of players, dazzling graphic interface, a huge amount of game content, game fairness, etc. Thus, design of MOG systems gets more attention from academia

recently. Christophe and Laurent built a small-scale distributed multiplayer interactive application, called MiMaze, and studied the architecture issue and the synchronization problem. Nathaniel et al. presented a protocol to provide the anti-cheating guarantee in MOG. Song-Jin et al. [8] focused on a real-time programming methodology, where the time-triggered message-triggered object (TMO) with a global timestamp was proposed for a scalable online game. Eric et al. discussed a new synchronization scheme called trailing tail to lower the complexity. Although MOG has a huge market potential, very little research work has been published on MOG. The current research is scattered and isolated. In this paper we attempt to give an in-deep inspection of MOG to help researchers understand the system structure, the current research stage and the challenges. The whole MOG system consists of two basic platforms. The first one is the underlying infrastructure, i.e., the physical platform, such as networking and hardware (including processors and graphic cards, etc.) The physical platform imposes a physical limitation on the MOG system. The second platform, so-called the logical platform, is about the system architecture, e.g., how hardware is organized, and how software is implemented in hosts and players. The goal of the logical platform is to provide the best performance based on the existing physical platform.

Browser games are computer games played directly in the web browser and do not need software installation. This accessibility has made those extremely popular and browser games have lured many players who would never buy a regular computer game to start playing. In the recent years, the evolution of browser games has been fast and a lot of new ones have been launched. There is also been evolution on technologies used in browser games. Traditional implementation technologies, PHP, ASP and Java jsp-pages have gained new competitors in the form of browser plugin players, which allow use of sophisticated 3D graphics. In addition to implementation technologies, different kinds of accessories for browser games have appeared. The games can be located inside an application platform, for example Facebook social utility. Different browser plugins are used to enchant play experience either to process game data for the player, or to modify the game graphics. Game engines have a strong position in

regular computer game development and recently those have appeared into browser game development also. Most likely eventually those will establish same position in browser game development also. In our previous paper we presented a definition for a browser game, different browser game types, and financial opportunities related to the genre. The paper was later extended with community aspect and published in. In this paper, we present technologies used to implement browser games and technologies used in browser game accessories. The rationale is that because of different software platforms (Web vs. binary), a game engine for browser games needs different kind of features than a traditional game engine. Currently there is no common understanding what features are important for such game engine. In this paper we present a proposal for these feature sets. We also discuss the future of the browser game genre as a whole. This paper is based on earlier conference article, but it has been extended with numerous technical details and with game engine specifications aspect.

2.1 AIM

HTML Canvas is used to generate shapes, sprites and animate sprite frames easily. With the help of Canvas API, we can code beautiful looking sprites and physics for our game. Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server. Our aim is to create multiplayer games that multiple people can enjoy together without needing top of the line hardware.

2.2 SCOPE

Browser games are computer games played directly in the web browser and do not need software installation. This accessibility has made those extremely popular and browser games have lured many players who would never buy a regular computer game to start playing. In the recent years, the evolution of browser games has been fast and a lot of new ones have been launched. There is also been evolution on technologies used in browser games. Traditional

implementation technologies, PHP, ASP and Java jsp-pages have gained new competitors in the form of browser plugin players, which allow use of sophisticated 3D graphics.

2.2.1 Accessibility

There are already several software suites designed for ease of access and development of browser based multiplayer games. VSCode is well optimised for the use in web development related projects like these and it has one of the largest repositories available. Boiler plate codes are readily available and easy to setup and learn at this current date and time.

JavaScript is also very accessible to people who are not familiar with web development. Game development incorporates basic OOP concepts and anyone very well versed in them can easily create a game. Anyone well versed with OOP concepts even in non-game settings can easily dabble on to create a great browser game.

CHAPTER 3

METHODS OR ALGORITHMS USED

3.1 HARDWARE/SOFTWARE REQUIREMENT

- Windows/Mac/Linux Based System
- Any text editor like VSCode, Sublime Text, Atom etc.
- Any chromium-based browser like Google Chrome, Microsoft Edge etc.

3.1.1 *Windows/Mac/Linux System*

Windows, Mac or Linux based operating system for personal computers or any other internet of things devices.

3.1.2 *Any text editor like VSCode, Sublime text, Atom etc.*

Any text editor works. Some of them are:

- VSCode
- Sublime Text
- Atom

3.2 SETTING UP

To publish and install packages to and from the public npm registry or a private npm registry, you must install Node.js and the npm command line interface using either a Node version manager or a Node installer. We strongly recommend using a Node version manager like nvm to install Node.js and npm. We do not recommend using a Node installer, since the Node installation process installs npm in a directory with local permissions and can cause permissions errors when you run npm packages globally. Note: to download the latest version of npm, on the command line, run the following command: `npm install -g npm`

3.3 DEVELOPMENT OF THE GAME

3.3.1 Creating *index.html* and *index.js*

Setup git from cmd and open the project folder we used git init on. We create two files *index.html* – for displaying our game and *index.js* – to interact with *index.html* and store our game logic. We start by adding a few divs and our main canvas element which will display our game. We start by making our classes in *index.js* itself but we will later move them to their separate file.

3.3.2 Creating necessary classes

The below code is our class for the sprites we will be using for both our fighters.

```
class Sprite {
  constructor({position, imageSrc, scale = 1, framesMax = 1, offset = {x: 0, y:0}}) {
    this.position = position
    this.width = 50
    this.height = 150
    this.image = new Image()
    this.image.src = imageSrc
    this.scale = scale
    this.framesMax = framesMax
    this.framesCurrent = 0
    this.framesElapsed = 0
    this.framesHold = 15
    this.offset = offset
  }

  draw() {
    c.drawImage(
      this.image,
      this.framesCurrent * (this.image.width / this.framesMax),
      0,
      this.image.width / this.framesMax,
      this.image.height,
      this.position.x - this.offset.x,
      this.position.y - this.offset.y,
      (this.image.width / this.framesMax)* this.scale,
      this.image.height * this.scale
    )
  }

  animateFrames(){
    this.framesElapsed++

    if(this.framesElapsed % this.framesHold === 0){
      if(this.framesCurrent < this.framesMax - 1)
        this.framesCurrent++
      else
        this.framesCurrent = 0
    }
  }

  update() {
    this.draw()
    this.animateFrames()
  }
}
```

Fig 3.1: Sprite class

We'll be making 2 instances of this class for both of our players. The draw() method draws our characters while the animateFrames method counts which frame our character sprite is currently animating. The update() function creates our loop by calling draw() and animateFrames() repeatedly.

```
class Fighter extends Sprite{
  constructor({position, velocity, color = 'red', imageSrc, scale = 1, framesMax = 1, offset = {x: 0, y:0},
  sprites, attackBox = {offset:{},width: undefined, height: undefined}}) {
    super({
      position,
      imageSrc,
      scale,
      framesMax,
      offset
    })

    this.velocity = velocity
    this.width = 50
    this.height = 150
    this.lastKey
    this.attackBox = {
      position: {
        x: this.position.x,
        y: this.position.y
      },
      offset: attackBox.offset,
      width: attackBox.width,
      height: attackBox.height
    }

    this.color = color
    this.isAttacking = false
    this.health = 100

    this.framesCurrent = 0
    this.framesElapsed = 0
    this.framesHold = 15
    this.sprites = sprites

    this.dead = false

    for (const sprite in this.sprites) {
      sprites[sprite].image = new Image()
      sprites[sprite].image.src = sprites[sprite].imageSrc
    }
    console.log(this.sprites);
  }
}
```

Fig 3.2: Fighter class

Our Fighter class holds the data members for both our fighter characters. These data members include position, hitbox, attackbox, player state, scale, animation data etc. We'll create 2 instances of this class for both our fighters.

3.3.3 Creating instances of our classes

We've called the canvas API to render a 2d plane with 1024 width and 576 height as this screen size fits most computer screens. We've declared our gravity to make our players fall and also made two instances of Sprite class as background and shop which would render both the assets.

```
○ ○ ○  
  
const canvas = document.querySelector('canvas')  
const c = canvas.getContext('2d')  
  
canvas.width = 1024  
canvas.height = 576  
  
c.fillRect(0, 0, canvas.width, canvas.height)  
  
const gravity = 0.5  
  
const background = new Sprite({  
  position: {  
    x: 0,  
    y: 0  
  },  
  imageSrc: './assets/background.png'  
})  
  
const shop = new Sprite({  
  position: {  
    x: 600,  
    y: 160  
  },  
  imageSrc: './assets/shop.png',  
  scale: 2.5,  
  framesMax: 6  
})
```

Fig: 3.3: Rendering Canvas and making background sprites

3.3.4 Player Instance from fighter class

```
const player = new Fighter({
  position: {
    x: 100,
    y: 0
  },
  velocity: {
    x: 0,
    y: 0
  },
  offset: {
    x: 0,
    y: 0
  },
  imageSrc: './assets/player/Idle.png',
  framesMax: 8,
  scale: 2.5,
  offset: {
    x: 215,
    y: 156
  },
  sprites: {
    idle: {
      imageSrc: './assets/player/Idle.png',
      framesMax: 8
    },
    run: {
      imageSrc: './assets/player/Run.png',
      framesMax: 8
    },
    jump: {
      imageSrc: './assets/player/Jump.png',
      framesMax: 2
    },
    fall: {
      imageSrc: './assets/player/Fall.png',
      framesMax: 2
    },
    attack1: {
      imageSrc: './assets/player/Attack1.png',
      framesMax: 6
    },
    takeHit: {
      imageSrc: './assets/player/Take Hit - white silhouette.png',
      framesMax: 4
    },
    death: {
      imageSrc: './assets/player/Death.png',
      framesMax: 6
    }
  },
  attackBox: {
    offset: {
      x: 100,
      y: 50
    },
    width: 150,
    height: 45
  }
})
```

Fig 3.4: Player Instance

We create our fighter with his unique starting position and initialise his velocity, sprite asset for every action, attackbox, scale etc. We duplicate this for our other fighter as well and change all the necessary values.

3.3.5 Building our animation loop

```
function animate() {
  window.requestAnimationFrame(animate)
  c.fillStyle = 'black'
  c.fillRect(0, 0, canvas.width, canvas.height)
  background.update()
  shop.update()
  c.fillStyle = 'rgba(255, 255, 255, 0.12)'
  c.fillRect(0, 0, canvas.width, canvas.height)
  player.update()
  enemy.update()

  player.velocity.x = 0
  enemy.velocity.x = 0

  //player movement
  if(keys.a.pressed && player.lastKey === 'a' && player.position.x >= 0){
    player.velocity.x = -5
    player.switchSprite('run')
  }
  else if(keys.d.pressed && player.lastKey === 'd' && player.position.x + 50 <= 1024){
    player.velocity.x = 5
    player.switchSprite('run')
  } else {
    player.switchSprite('idle')
  }

  //jumping
  if(player.velocity.y < 0) {
    player.switchSprite('jump')
  } else if (player.velocity.y > 0) {
    player.switchSprite('fall')
  }

  //enemy movement
  if(keys.ArrowLeft.pressed && enemy.lastKey === 'ArrowLeft' && enemy.position.x >= 0){
    enemy.velocity.x = -5
    enemy.switchSprite('run')
  }
  else if(keys.ArrowRight.pressed && enemy.lastKey === 'ArrowRight' && enemy.position.x + 50 <= 1024){
    enemy.velocity.x = 5
    enemy.switchSprite('run')
  } else {
    enemy.switchSprite('idle')
  }

  //jumping
  if(enemy.velocity.y < 0) {
    enemy.switchSprite('jump')
  } else if (enemy.velocity.y > 0) {
    enemy.switchSprite('fall')
  }

  //collision detection
  if (rectangularCollision({
    rectangle1: player,
    rectangle2: enemy
  }) && player.isAttacking && player.framesCurrent === 4)
  {
    enemy.takeHit()
    player.isAttacking = false

    gsap.to('#enemyHealth',{
      width: enemy.health + '%'
    })
  }

  //on miss
  if(player.isAttacking && player.framesCurrent === 4){
    player.isAttacking = false
  }

  if (rectangularCollision({
    rectangle1: enemy,
    rectangle2: player
  }) && enemy.isAttacking && enemy.framesCurrent === 2)
  {
    player.takeHit()
    enemy.isAttacking = false
    gsap.to('#playerHealth',{
      width: player.health + '%'
    })
  }

  if(enemy.isAttacking && enemy.framesCurrent === 2){
    enemy.isAttacking = false
  }

  if(enemy.health <= 0 || player.health <= 0){
    determineWinner({player, enemy, timerId})
  }
}

animate()
```

Fig 3.5: Animation loop function

The above function determines our animation loop. It handles cases for our movement, HP deduction, jumping, collision detection, sprite switching etc. We call it after the function body to loop it.

3.3.6 Building our key-listener



```

window.addEventListener('keydown', (event) => {
  if(!player.dead){
    switch(event.key){
      case 'd':
        keys.d.pressed = true
        player.lastKey = 'd'
        break
      case 'a':
        keys.a.pressed = true
        player.lastKey = 'a'
        break
      case 'w':
        if(player.velocity.y === 0)
          player.velocity.y = -15
        break
      case ' ':
        player.attack()
        break
    }
    //enemy stuff
    if(!enemy.dead){
      switch(event.key) {
        case 'ArrowRight':
          keys.ArrowRight.pressed = true
          enemy.lastKey = 'ArrowRight'
          break
        case 'ArrowLeft':
          keys.ArrowLeft.pressed = true
          enemy.lastKey = 'ArrowLeft'
          break
        case 'ArrowUp':
          if(enemy.velocity.y === 0)
            enemy.velocity.y = -15
          break
        case 'ArrowDown':
          enemy.attack()
          break
      }
    }
  }
})

```

Fig 3.6: Key Listener function

We use `addEventListener()` to look for keys pressed and assign actions for both our players on respective buttons.

3.3.7 Adding the necessary UI and linking everything with index.html

```
<head>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap" rel="stylesheet">
  <style>
    * {
      box-sizing: border-box;
      font-family: 'Press Start 2P', cursive;
    }
  </style>
</head>
<body>
  <!--container div-->
  <div style="position:relative; display: inline-block;">
    <!--smaller container div-->
    <div
      style="position:absolute;
      display: flex;
      width: 100%;
      align-items: center;
      padding: 20px;
    ">
      <!-- player health -->
      <div style='position: relative; width: 100%; display: flex; justify-content: flex-end; border-top: 4px
solid white; border-left: 4px solid white; border-bottom: 4px solid white;'>
        <div style="background-color: red; height:30px; width:100%"></div>
        <div id="playerHealth"
          style="position:absolute;
          background: #818CF8;
          top: 0;
          right: 0;
          bottom: 0;
          width: 100%;
        ">
      </div>
    </div>

    <!-- clock -->
    <div
      id="timer"
      style="background-color: black;
      height: 50px;
      width: 100px;
      flex-shrink: 0;
      display: flex;
      align-items: center;
      justify-content: center;
      color: white;
      border: 4px solid white;
    ">
      10
    </div>

    <!-- enemy health -->
    <div style='position: relative; width: 100%; border-top: 4px solid white; border-right: 4px solid
white; border-bottom: 4px solid white;'>
      <div style="background-color: red; height:30px"></div>
      <div id="enemyHealth"
        style="position: absolute;
        background: #818CF8;
        top: 0;
        right: 0;
        bottom: 0;
        left: 0;
      ">
    </div>
  </div>
  <div id="displayText" style="position: absolute; color:white; align-items:center;justify-content: center; top:
0; right: 0; bottom: 0; left: 0; display:none">Tie</div>
  <canvas></canvas>
</div>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.9.1/gsap.min.js" integrity="sha512-
H6cPm97FAsgJkmLBA4s774vqoN24V5gSQL4yBTD0Y2su2DeXZVhQPxFK4P6GpDnZqM9fg1G3cMv5wD7e6cFLZQ==" crossorigin="anonymous"
referrerpolicy="no-referrer"></script>

  <script src="js/utility.js"></script>
  <script src="js/classes.js"></script>
  <script src="index.js"></script>
</body>
```

Fig 3.7: HTML UI and Render Code

The above code takes care of drawing UI for clock, HP, and winner announcement. It keeps track of them using 3 different divs, 1 each for player health, enemy health and timer.

3.3.8 Making separate file for utility functions

```
function rectangularCollision({rectangle1,rectangle2}){
  return (
    rectangle1.attackBox.position.x + rectangle1.attackBox.width >= rectangle2.position.x
    && rectangle1.attackBox.position.x <= rectangle2.position.x + rectangle2.width
    && rectangle1.attackBox.position.y + rectangle1.attackBox.height >= rectangle2.position.y
    && rectangle1.attackBox.position.y <= rectangle2.position.y + rectangle2.height
  )
}

function determineWinner({player,enemy,timerId}){
  clearTimeout(timerId)
  document.querySelector('#displayText').style.display='flex'
  if(player.health === enemy.health){
    document.querySelector('#displayText').innerHTML='Tie'
  } else if(player.health > enemy.health){
    document.querySelector('#displayText').innerHTML='Samurai Wins!'
  } else document.querySelector('#displayText').innerHTML='Tenguu Wins!'
}

let timer = 60
let timerId
function decreaseTimer(){
  if(timer>0){
    timerId=setTimeout(decreaseTimer,1000)
    timer-=1
    document.querySelector('#timer').innerHTML = timer
  }

  if(timer==0){
    document.querySelector('#displayText').style.display='flex'
    determineWinner({player,enemy,timerId})
  }
}
```

Fig 3.8 Utility functions

Rectangular collision detects for collision between the attack box and the hit box for our players and returns whether true or false. determineWinner() determines the winner and displays our text for whoever wins the match. Lastly, decreaseTimer() keeps decreasing the clock every second.

3.3.9 Snowball Multiplayer Game

3.3.9.1 Setting Up Github/npm and initialising index.html

We set up our project folder and initialise git using git init.

```

function renderAvatar (player) {
  if (player.eliminated) return

  ctx.save()
  ctx.translate(player.x, player.y)

  // draw body
  ctx.beginPath()
  ctx.arc(0, 0, 20, 0, 2 * Math.PI)
  ctx.closePath()
  ctx.fillStyle = player.color
  ctx.fill()

  // draw username
  ctx.textAlign = 'center'
  ctx.fillStyle = 'black'
  ctx.fillText(player.username, 0, 34)
  //draw eyes
  switch (player.rotation) {
    case FACING_DOWN:
      ctx.rotate(0)
      break
    case FACING_UP:
      ctx.rotate(Math.PI)
      break
    case FACING_LEFT:
      ctx.rotate(Math.PI / 2)
      break
    case FACING_RIGHT:
      ctx.rotate(Math.PI * 1.5)
      break
  }
  ctx.beginPath()
  ctx.moveTo(-5, 5)
  ctx.lineTo(-5, 17)
  ctx.moveTo(5, 5)
  ctx.lineTo(5, 17)
  ctx.stroke()

  ctx.restore()
}

function renderSnowball (snowball) {
  ctx.save()
  ctx.translate(snowball.x, snowball.y)

  ctx.beginPath()
  ctx.arc(0, 0, 8, 0, 2 * Math.PI)
  ctx.closePath()
  ctx.fillStyle = 'lightblue'
  ctx.fill()

  ctx.restore()
}

```

Fig 3.9 Snowball game player render

The canvas is initialised. The function `renderAvatar()` is used to render our player and `renderSnowball()` is used to render our player's snowball attacks.

3.3.9.2 Building logic functions

```
function logicStep (state) {
  state.players.forEach(player => { // just another way of writing f
    player.snowballs.forEach(snowball => {
      snowball.x += snowball.vx
      snowball.y += snowball.vy

      if (snowball.x < 0 || snowball.x > window.innerWidth ||
          snowball.y < 0 || snowball.y > window.innerHeight) {
        snowball.remove = true
      }
    })
    player.snowballs = player.snowballs.filter(snowball => {
      const shouldBeKept = (snowball.remove !== true)
      return shouldBeKept
    })
  })
  // collision algorithm
  state.players.forEach(playerA => {
    playerA.snowballs.forEach(snowball => {
      state.players.forEach(playerB => {
        if (playerA === playerB) {
          // don't hit yourself
          return
        }

        if (hitTestPlayerVsSnowball(playerB, snowball)) {
          snowball.remove = true
          if (playerB.playerId === myPlayerId) {
            // others must remove themselves
            playerB.eliminated = true
          }
        }
      })
    })
  })

  // remove eliminated players
  // state.players = state.players.filter(player => !player.eliminated)

  socket.emit('stateUpdate', state.players[0])
}
```

Fig 3.10: Building Logic functions and collision detection

logicStep() determines our game logic. It's simple shoot and hit, if a player successfully shoots another player, the player dies. We also code a collision algorithm to detect whether a snowball has hit our player.

3.3.9.3 Building necessary event listeners

```
document.addEventListener('keydown', function (e) {
  keyboardState[e.key] = true

  if (e.key === ' ') { // spacebar lol

    const myPlayer = gameState.players[0]
    const snowball = {
      x: myPlayer.x, y: myPlayer.y,
      vx: 0, vy: 0
    }

    switch (myPlayer.rotation) {
      case FACING_DOWN:
        snowball.vy = SNOWBALL_SPEED
        break
      case FACING_UP:
        snowball.vy = -SNOWBALL_SPEED
        break
      case FACING_LEFT:
        snowball.vx = -SNOWBALL_SPEED
        break
      case FACING_RIGHT:
        snowball.vx = +SNOWBALL_SPEED
        break
    }

    myPlayer.snowballs.push(snowball)
  }
})

document.addEventListener('keyup', function (e) {
  keyboardState[e.key] = false
})

socket.on('stateUpdateForwardedByServer', function (player) {
  if (player.playerId === myPlayerId) {
    // ignore own update
    return
  }

  let playerWasFound = false
  for (let i = 0; i < gameState.players.length; ++i) {
    if (gameState.players[i].playerId === player.playerId) {
      gameState.players[i] = player
      playerWasFound = true
      break // don't continue looping, pointless
    }
  }

  if (!playerWasFound) {
    // player we haven't seen before
    gameState.players.push(player)
  }
})
```

Fig 3.11: Event Listeners and Socket API

We build an event listener to determine movement and action for our players. We then map the necessary movement commands to their respective buttons. We also build socket.io logic to detect if any player is in the lobby and change the game state accordingly. We also add new players to the lobby using socket command.

3.3.9.4 Coding the server and building our package

We use socket api which is a library that enables low-latency, bidirectional and event-based communication between a client and a server. We then build our package using npm and publish it to github directly.

```
var express = require('express')
var app = express()
var path = require('path')
var server = require('http').createServer(app)
var io = require('socket.io')(server)
var port = process.env.PORT || 3000

server.listen(port, () => {
  console.log('Server listening at port %d', port)
})

// Routing
app.use(express.static(path.join(__dirname, 'public')))

io.on('connection', (socket) => {
  console.log('got a connection!')
  socket.on('stateUpdate', function (player) {
    console.log('got state update', player)
    io.sockets.emit('stateUpdateForwardedByServer', player)
  })
})
```

Fig 3.13 Server code

CHAPTER 4

HOSTING

4.1 Hosting on Netlify and Heroku

4.1.1 *Hosting 2D fighting game on Netlify*

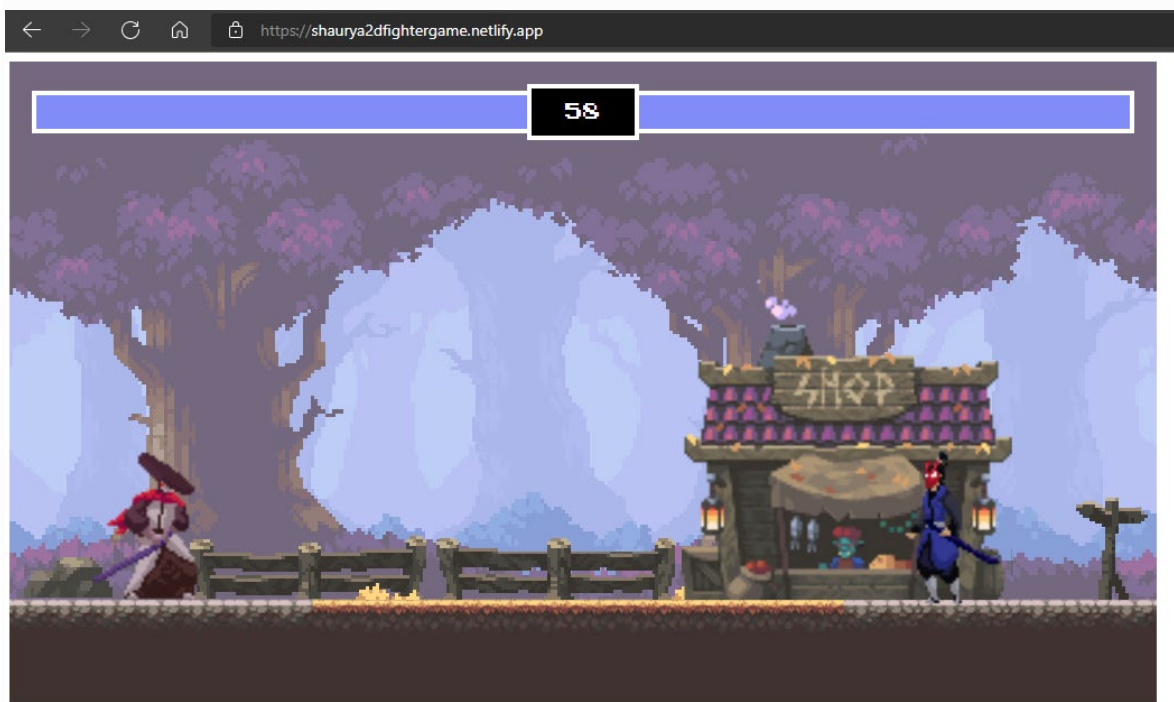


Fig: 4.1: The game hosted on Netlify

We've successfully made a beautiful local multiplayer game hosted online for free that anybody can play at any desktop and enjoy regardless of hardware.

Link: <https://shaurya2dfightergame.netlify.app>

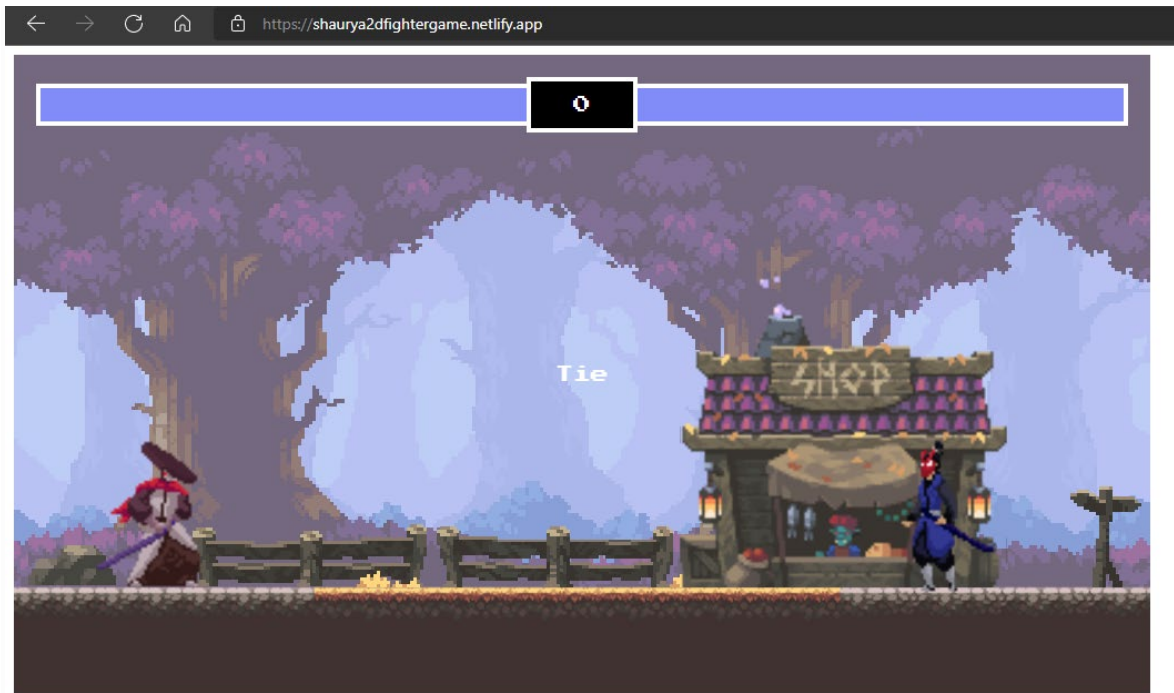


Fig: 4.2 Tie due to timer running out

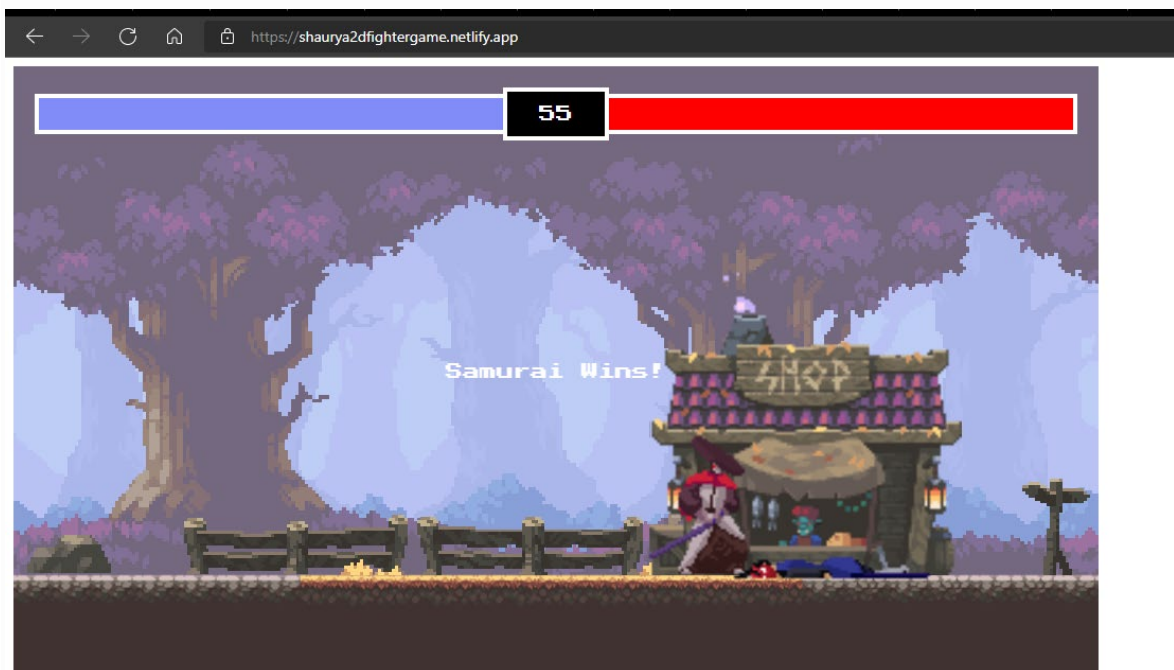


Fig 4.3 Samurai winning by killing his opponent Tenguu

4.1.2 Hosting snowball game on Heroku from github

We connected our github account to Heroku and directly hosted our game on it.

Link: <https://multiplayerbrowsergame.herokuapp.com>

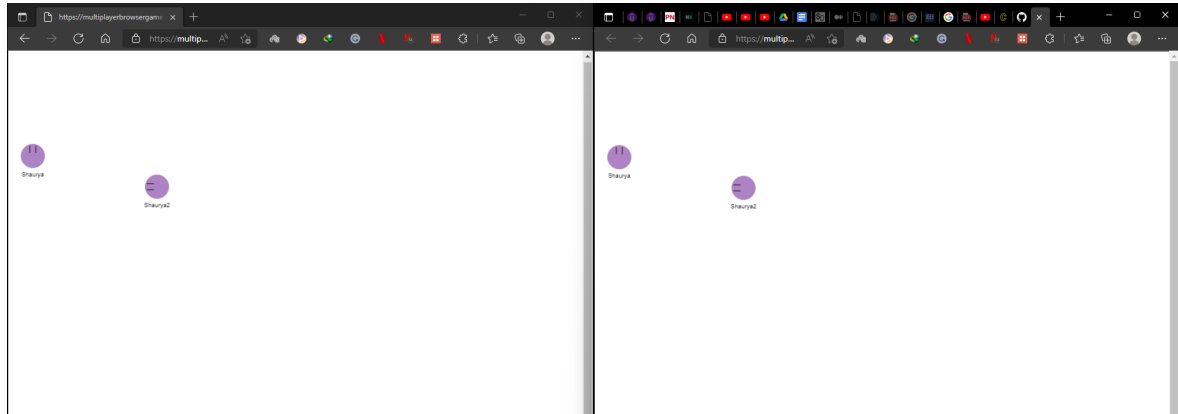


Fig 4.4 Hosted snowball game on Heroku

CHAPTER 5

SUMMARY AND CONCLUSIONS

To conclude, we successfully demonstrated the usage of OOP concepts. We used vanilla JavaScript, HTML Canvas, CSS and Socket library to create two different and unique browser gaming experiences, one specializing in connectivity while the other specialising in game mechanics.

We demonstrate how easy it is to make your own game and host it on the web and also discovered the hidden financial potential behind an untapped industry.

5.1 FUTURE WORK

Project can be extended to various industry level games and also be monetised for financial incentive. It can be easily expanded to larger projects with larger scopes. Web based games have a lot of potential because they are hardware independent and you can play them anywhere you want. Thus, this study concluded how easy game development can be and how simple web hosting can be and demonstrated the financial potential of web-based games like these.

REFERENCES

PROJECT:

- J. Vanhatupa, "On the development of browser games - Technologies of an emerging genre," 2011 7th International Conference on Next Generation Web Services Practices, 2011, pp. 363-368, doi: 10.1109/NWeSP.2011.6088206.
- K. Monu, J. X. Zhang and P. Ralph, "Introduction to the Online Games and Game-Like Systems Minitrack," 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp. 3768-3768, doi: 10.1109/HICSS.2016.469..
- K. Monu, J. X. Zhang and P. Ralph, "Introduction to the Online Games and Game-Like Systems Minitrack," 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp. 3768-3768, doi: 10.1109/HICSS.2016.469.
- www.google.com
- www.geeksforgeeks.com

APPENDIX

SOURCE CODE

Project 1: (2D Fighting Game) - [bridgesams/2D-fighting-game: college project \(github.com\)](https://github.com/bridgesams/2D-fighting-game)

Game Link: <https://shaurya2dfightergame.netlify.app/>

classes.js (contains all our classes):

```
class Sprite {
  constructor({position, imageSrc, scale = 1, framesMax = 1, offset = {x:
0, y:0}}) {
    this.position = position
    this.width = 50
    this.height = 150
    this.image = new Image()
    this.image.src = imageSrc
    this.scale = scale
    this.framesMax = framesMax
    this.framesCurrent = 0
    this.framesElapsed = 0
    this.framesHold = 15
    this.offset = offset
  }

  draw() {
    c.drawImage(
      this.image,
      this.framesCurrent * (this.image.width / this.framesMax),
      0,
      this.image.width / this.framesMax,
      this.image.height,
      this.position.x - this.offset.x,
      this.position.y - this.offset.y,
      (this.image.width / this.framesMax)* this.scale,
      this.image.height * this.scale
    )
  }

  animateFrames(){
    this.framesElapsed++

    if(this.framesElapsed % this.framesHold === 0){
```

```

        if(this.framesCurrent < this.framesMax - 1)
            this.framesCurrent++
        else
            this.framesCurrent = 0
    }
}

update() {
    this.draw()
    this.animateFrames()
}
}

class Fighter extends Sprite{
    constructor({position, velocity, color = 'red', imageSrc, scale = 1,
framesMax = 1, offset = {x: 0, y:0}, sprites, attackBox = {offset:{},width:
undefined, height: undefined}}) {
        super({
            position,
            imageSrc,
            scale,
            framesMax,
            offset
        })

        this.velocity = velocity
        this.width = 50
        this.height = 150
        this.lastKey
        this.attackBox = {
            position: {
                x: this.position.x,
                y: this.position.y
            },
            offset: attackBox.offset,
            width: attackBox.width,
            height: attackBox.height
        }

        this.color = color
        this.isAttacking = false
        this.health = 100

        this.framesCurrent = 0
        this.framesElapsed = 0
        this.framesHold = 15
        this.sprites = sprites

```

```

        this.dead = false

        for (const sprite in this.sprites) {
            sprites[sprite].image = new Image()
            sprites[sprite].image.src = sprites[sprite].imageSrc
        }
        console.log(this.sprites);
    }

    update() {
        this.draw()
        if(!this.dead)
            this.animateFrames()

        //attack boxes
        this.attackBox.position.x = this.position.x +
this.attackBox.offset.x
        this.attackBox.position.y = this.position.y +
this.attackBox.offset.y

        //c.fillRect(this.attackBox.position.x, this.attackBox.position.y,
this.attackBox.width,this.attackBox.height)

        this.position.x += this.velocity.x
        this.position.y += this.velocity.y

        //gravity
        if(this.position.y + this.height + this.velocity.y >= canvas.height
- 96){
            this.velocity.y = 0
            this.position.y = 330
        }
        else
            this.velocity.y += gravity
    }

    attack(){
        this.switchSprite('attack1')
        this.isAttacking = true
    }

    takeHit(){
        if(player.isAttacking)
            this.health -= 20
        else
            this.health -= 15

        if(this.health <= 0)

```

```

        this.switchSprite('death')
    else
        this.switchSprite('takeHit')
    }

    switchSprite(sprite) {
        if (this.image === this.sprites.death.image){
            if (this.framesCurrent === this.sprites.death.framesMax -
1){this.dead = true} return}
            if(this.image === this.sprites.attack1.image && this.framesCurrent <
this.sprites.attack1.framesMax - 1) return

            if(this.image === this.sprites.takeHit.image && this.framesCurrent <
this.sprites.takeHit.framesMax - 1) return

        switch(sprite) {
            case 'idle':
                if(this.image !== this.sprites.idle.image){
                    this.image = this.sprites.idle.image
                    this.framesMax = this.sprites.idle.framesMax
                    this.framesCurrent = 0
                }
                break
            case 'run':
                if(this.image !== this.sprites.run.image){
                    this.image = this.sprites.run.image
                    this.framesMax = this.sprites.run.framesMax
                    this.framesCurrent = 0
                }
                break
            case 'jump':
                if(this.image !== this.sprites.jump.image){
                    this.image = this.sprites.jump.image
                    this.framesMax = this.sprites.jump.framesMax
                    this.framesCurrent = 0
                }
                break
            case 'fall':
                if(this.image !== this.sprites.fall.image){
                    this.image = this.sprites.fall.image
                    this.framesMax = this.sprites.fall.framesMax
                    this.framesCurrent = 0
                }
                break
            case 'attack1':
                if(this.image !== this.sprites.attack1.image){
                    this.image = this.sprites.attack1.image
                    this.framesMax = this.sprites.attack1.framesMax
                    this.framesCurrent = 0

```

```

        }
        break
    case 'takeHit':
        if(this.image !== this.sprites.takeHit.image){
            this.image = this.sprites.takeHit.image
            this.framesMax = this.sprites.takeHit.framesMax
            this.framesCurrent = 0
        }
        break
    case 'death':
        if(this.image !== this.sprites.death.image){
            this.image = this.sprites.death.image
            this.framesMax = this.sprites.death.framesMax
            this.framesCurrent = 0
        }
        break
    }
}
}
}

```

utils.js (All our utility functions):

```

function rectangularCollision({rectangle1,rectangle2}){
    return (
        rectangle1.attackBox.position.x + rectangle1.attackBox.width >=
rectangle2.position.x
        && rectangle1.attackBox.position.x <= rectangle2.position.x +
rectangle2.width
        && rectangle1.attackBox.position.y + rectangle1.attackBox.height >=
rectangle2.position.y
        && rectangle1.attackBox.position.y <= rectangle2.position.y +
rectangle2.height
    )
}

function determineWinner({player,enemy,timerId}){
    clearTimeout(timerId)
    document.querySelector('#displayText').style.display='flex'
    if(player.health === enemy.health){
        document.querySelector('#displayText').innerHTML='Tie'
    } else if(player.health > enemy.health){
        document.querySelector('#displayText').innerHTML='Samurai Wins!'
    } else document.querySelector('#displayText').innerHTML='Tengu Wins!'
}

let timer = 60

```

```

let timerId
function decreaseTimer(){
  if(timer>0){
    timerId=setTimeout(decreaseTimer,1000)
    timer-=1
    document.querySelector('#timer').innerHTML = timer
  }

  if(timer==0){
    document.querySelector('#displayText').style.display='flex'
    determineWinner({player,enemy,timerId})
  }
}

```

index.html:

```

<head>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
  <style>
    * {
      box-sizing: border-box;
      font-family: 'Press Start 2P', cursive;
    }
  </style>
</head>
<body>
  <!--container div-->
  <div style="position:relative; display: inline-block;">
    <!--smaller container div-->
    <div
      style="position:absolute;
      display: flex;
      width: 100%;
      align-items: center;
      padding: 20px;
      ">
      <!-- player health -->
      <div style ='position: relative; width: 100%; display: flex;
justify-content: flex-end; border-top: 4px solid white; border-left: 4px
solid white; border-bottom: 4px solid white;'>
        <div style="background-color: red; height:30px;
width:100%"></div>

```

```

        <div id="playerHealth"
            style="position: absolute;
                background: #818CF8;
                top: 0;
                right: 0;
                bottom: 0;
                width: 100%;
                "
        ></div>
</div>

<!-- clock -->
<div
    id="timer"
    style="background-color: black;
        height: 50px;
        width: 100px;
        flex-shrink: 0;
        display: flex;
        align-items: center;
        justify-content: center;
        color: white;
        border: 4px solid white;
        ">
        10
</div>

<!-- enemy health -->

<div style ='position: relative; width: 100%; border-top: 4px
solid white; border-right: 4px solid white; border-bottom: 4px solid
white;'>
    <div style="background-color: red; height: 30px"></div>
    <div id="enemyHealth"
        style="position: absolute;
            background: #818CF8;
            top: 0;
            right: 0;
            bottom: 0;
            left: 0;
            "
    ></div>
</div>
<div></div>
</div>
    <div id="displayText" style= "position: absolute; color: white; align-
items: center; justify-content: center; top: 0; right: 0; bottom: 0; left: 0;
display: none">Tie</div>
<canvas></canvas>

```

```

    </div>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.9.1/gsap.min.js"
integrity="sha512-
H6cPm97FAsgIKmlBA4s774vqoN24V5gSQL4yBTDOY2su2DeXZVhQPxFk4P6GPdnZqM9fg1G3cMv5
wD7e6cFLZQ==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>

    <script src="js/utility.js"></script>
    <script src="js/classes.js"></script>
    <script src="index.js"></script>
</body>

```

index.js:

```

const canvas = document.querySelector('canvas')
const c = canvas.getContext('2d')

canvas.width = 1024
canvas.height = 576

c.fillRect(0, 0, canvas.width, canvas.height)

const gravity = 0.5

const background = new Sprite({
  position: {
    x: 0,
    y: 0
  },
  imageSrc: './assets/background.png'
})

const shop = new Sprite({
  position: {
    x: 600,
    y: 160
  },
  imageSrc: './assets/shop.png',
  scale: 2.5,
  framesMax: 6
})

const player = new Fighter({
  position: {
    x: 100,

```

```
        y: 0
    },
    velocity: {
        x: 0,
        y: 0
    },
    offset: {
        x:0,
        y:0
    },
    imageSrc: './assets/player/Idle.png',
    framesMax: 8,
    scale: 2.5,
    offset: {
        x:215,
        y: 156
    },
    sprites: {
        idle: {
            imageSrc: './assets/player/Idle.png',
            framesMax: 8
        },
        run: {
            imageSrc: './assets/player/Run.png',
            framesMax: 8
        },
        jump: {
            imageSrc: './assets/player/Jump.png',
            framesMax: 2
        },
        fall:{
            imageSrc: './assets/player/Fall.png',
            framesMax: 2
        },
        attack1: {
            imageSrc: './assets/player/Attack1.png',
            framesMax: 6
        },
        takeHit: {
            imageSrc: './assets/player/Take Hit - white silhouette.png',
            framesMax: 4
        },
        death: {
            imageSrc: './assets/player/Death.png',
            framesMax: 6
        }
    },
    attackBox:{
        offset: {
```



```

    },
    takeHit: {
      imageSrc: './assets/kenji/Take hit.png',
      framesMax: 3
    },
    death: {
      imageSrc: './assets/kenji/Death.png',
      framesMax: 7
    }
  },
  attackBox:{
    offset: {
      x: -170,
      y: 50
    },
    width: 170,
    height: 45
  }
})

```

```

const keys = {
  a: {
    pressed: false
  },
  d: {
    pressed: false
  },
  w: {
    pressed: false
  },
  ArrowLeft: {
    pressed: false
  },
  ArrowRight: {
    pressed: false
  },
  ArrowUp: {
    pressed: false
  }
}

```

```

decreaseTimer()

```

```

function animate() {
  window.requestAnimationFrame(animate)
  c.fillStyle = 'black'
  c.fillRect(0, 0, canvas.width, canvas.height)
  background.update()
  shop.update()
}

```

```

c.fillStyle = 'rgba(255, 255, 255, 0.12)'
c.fillRect(0, 0, canvas.width, canvas.height)
player.update()
enemy.update()

player.velocity.x = 0
enemy.velocity.x = 0

//player movement
if(keys.a.pressed && player.lastKey === 'a' && player.position.x >= 0){
    player.velocity.x = -5
    player.switchSprite('run')
}
else if(keys.d.pressed && player.lastKey === 'd' && player.position.x +
50 <=1024){
    player.velocity.x = 5
    player.switchSprite('run')
} else {
    player.switchSprite('idle')
}

//jumping
if(player.velocity.y < 0) {
    player.switchSprite('jump')
} else if (player.velocity.y > 0) {
    player.switchSprite('fall')
}

//enemy movement
if(keys.ArrowLeft.pressed && enemy.lastKey === 'ArrowLeft' &&
enemy.position.x >= 0){
    enemy.velocity.x = -5
    enemy.switchSprite('run')
}
else if(keys.ArrowRight.pressed && enemy.lastKey === 'ArrowRight' &&
enemy.position.x + 50 <=1024){
    enemy.velocity.x = 5
    enemy.switchSprite('run')
} else {
    enemy.switchSprite('idle')
}

//jumping
if(enemy.velocity.y < 0) {
    enemy.switchSprite('jump')
} else if (enemy.velocity.y > 0) {
    enemy.switchSprite('fall')
}

```

```

//collision detection
if (rectangularCollision({
  rectangle1: player,
  rectangle2: enemy
}) && player.isAttacking && player.framesCurrent === 4)
{
  enemy.takeHit()
  player.isAttacking = false

  gsap.to('#enemyHealth',{
    width: enemy.health + '%'
  })
}

//on miss
if(player.isAttacking && player.framesCurrent === 4){
  player.isAttacking = false
}

if (rectangularCollision({
  rectangle1: enemy,
  rectangle2: player
}) && enemy.isAttacking && enemy.framesCurrent === 2)
{
  player.takeHit()
  enemy.isAttacking = false
  gsap.to('#playerHealth',{
    width: player.health + '%'
  })
}

if(enemy.isAttacking && enemy.framesCurrent === 2){
  enemy.isAttacking = false
}

if(enemy.health <= 0 || player.health <=0){
  determineWinner({player,enemy,timerId})
}
}

animate()

window.addEventListener('keydown', (event) => {
  if(!player.dead){
    switch(event.key){
      case 'd':
        keys.d.pressed = true
        player.lastKey = 'd'
        break
    }
  }
})

```

```

        case 'a':
            keys.a.pressed = true
            player.lastKey = 'a'
            break
        case 'w':
            if(player.velocity.y === 0)
                player.velocity.y = -15
            break
        case ' ':
            player.attack()
            break
    }

    //enemy stuff
    if(!enemy.dead){
        switch(event.key) {
            case 'ArrowRight':
                keys.ArrowRight.pressed = true
                enemy.lastKey = 'ArrowRight'
                break
            case 'ArrowLeft':
                keys.ArrowLeft.pressed = true
                enemy.lastKey = 'ArrowLeft'
                break
            case 'ArrowUp':
                if(enemy.velocity.y === 0)
                    enemy.velocity.y = -15
                break
            case 'ArrowDown':
                enemy.attack()
                break
        }
    }
}
}
console.log(event.key)
})

window.addEventListener('keyup', (event) => {
    switch(event.key){
        case 'd':
            keys.d.pressed = false
            break
        case 'a':
            keys.a.pressed = false
            break
        case 'w':
            keys.w.pressed = false
            break
    }
})

```

```
//enemy stuff
switch(event.key){
  case 'ArrowRight':
    keys.ArrowRight.pressed = false
    break
  case 'ArrowLeft':
    keys.ArrowLeft.pressed = false
    break
  case 'ArrowUp':
    keys.ArrowUp.pressed = false
    break
}
console.log(event.key)
})
```

Project 2:

Snowball Fighter Online Game - Github link:

[bridgesams/multiplayer-browser-game: college project \(github.com\)](https://github.com/bridgesams/multiplayer-browser-game)

Game link: <https://multiplayerbrowsergame.herokuapp.com/>

Index.html:

```
<html>
<head>

<style>

  html, body {
    padding: 0;
    margin: 0;
  }

  canvas {
    width: 100%;
    height: 100%;
    border: 1px solid black;
  }

</style>

</head>
<body>

<canvas></canvas>

<script src="/socket.io/socket.io.js"></script>
<script>

const socket = io()

const canvas = document.getElementsByTagName('canvas')[0]
canvas.width = window.innerWidth
canvas.height = window.innerHeight

const ctx = canvas.getContext('2d')
// ctx.fillRect(50, 50, 100, 100)

const keyboardState = {}
```

```

function renderAvatar (player) {
  if (player.eliminated) return

  ctx.save()
  ctx.translate(player.x, player.y)

  // draw body
  ctx.beginPath()
  ctx.arc(0, 0, 20, 0, 2 * Math.PI)
  ctx.closePath()
  ctx.fillStyle = player.color
  ctx.fill()

  // draw username
  ctx.textAlign = 'center'
  ctx.fillStyle = 'black'
  ctx.fillText(player.username, 0, 34)

  // draw eyes
  // rotate here prior to drawing the eyes,
  // to make the eyes face that the avatar should have
  switch (player.rotation) {
    case FACING_DOWN:
      ctx.rotate(0)
      break
    case FACING_UP:
      ctx.rotate(Math.PI)
      break
    case FACING_LEFT:
      ctx.rotate(Math.PI / 2)
      break
    case FACING_RIGHT:
      ctx.rotate(Math.PI * 1.5)
      break
  }
  ctx.beginPath()
  ctx.moveTo(-5, 5)
  ctx.lineTo(-5, 17)
  ctx.moveTo(5, 5)
  ctx.lineTo(5, 17)
  ctx.stroke()

  ctx.restore()
}

function renderSnowball (snowball) {
  ctx.save()
  ctx.translate(snowball.x, snowball.y)

```

```

    ctx.beginPath()
    ctx.arc(0, 0, 8, 0, 2 * Math.PI)
    ctx.closePath()
    ctx.fillStyle = 'lightblue'
    ctx.fill()

    ctx.restore()
}

const SNOWBALL_SPEED = 5
const PLAYER_SPEED = 3
const FACING_UP = 2
const FACING_DOWN = 0
const FACING_LEFT = 1
const FACING_RIGHT = 3

const gameState = {
  players: [
    {
      username: prompt("What's your username?"),
      playerId: Math.floor(Math.random() * 100000000),
      x: 50, y: 50,
      color: '#ae83c3',
      rotation: FACING_DOWN,
      snowballs: []
    }
  ]
}

const myPlayerId = gameState.players[0].playerId

function render (state) {
  ctx.fillStyle = 'white'
  ctx.fillRect(0, 0, canvas.width, canvas.height)

  state.players.forEach(function (player) {
    renderAvatar(player)

    player.snowballs.forEach(function (snowball) {
      renderSnowball(snowball)
    })
  })
}

const PLAYER_RADIUS = 20
const SNOWBALL_RADIUS = 8

function hitTestPlayerVsPlayer (playerA, playerB) {

```

```

    return Math.sqrt(Math.pow(playerA.x - playerB.x, 2) + Math.pow(playerA.y
- playerB.y, 2)) < (PLAYER_RADIUS * 2)
}
function hitTestPlayerVsSnowball (player, snowball) {
    return Math.sqrt(Math.pow(player.x - snowball.x, 2) + Math.pow(player.y
- snowball.y, 2)) < (PLAYER_RADIUS + SNOWBALL_RADIUS)
}

function logicStep (state) {
    state.players.forEach(player => { // just another way of writing f
        player.snowballs.forEach(snowball => {
            snowball.x += snowball.vx
            snowball.y += snowball.vy

            if (snowball.x < 0 || snowball.x > window.innerWidth ||
                snowball.y < 0 || snowball.y > window.innerHeight) {
                snowball.remove = true
            }
        })
        player.snowballs = player.snowballs.filter(snowball => {
            const shouldBeKept = (snowball.remove !== true)
            return shouldBeKept
        })
    })

    // only moving player one now
    const myPlayer = state.players[0]
    if (!myPlayer.eliminated) {
        if (keyboardState.w) {
            myPlayer.y -= PLAYER_SPEED
            myPlayer.rotation = FACING_UP
        }
        if (keyboardState.s) {
            myPlayer.y += PLAYER_SPEED
            myPlayer.rotation = FACING_DOWN
        }
        if (keyboardState.d) {
            state.players[0].x += PLAYER_SPEED
            myPlayer.rotation = FACING_RIGHT
        }
        if (keyboardState.a) {
            state.players[0].x -= PLAYER_SPEED
            myPlayer.rotation = FACING_LEFT
        }
    }

    // collision algorithm
    state.players.forEach(playerA => {
        playerA.snowballs.forEach(snowball => {

```

```

        state.players.forEach(playerB => {
            if (playerA === playerB) {
                // don't hit yourself
                return
            }

            if (hitTestPlayerVsSnowball(playerB, snowball)) {
                snowball.remove = true
                if (playerB.playerId === myPlayerId) {
                    // others must remove themselves
                    playerB.eliminated = true
                }
            }
        })
    })
})

// remove eliminated players
// state.players = state.players.filter(player => !player.eliminated)

socket.emit('stateUpdate', state.players[0])
}

function gameLoop () {
    requestAnimationFrame(gameLoop)
    logicStep(gameState)
    render(gameState)
}

gameLoop()

document.addEventListener('keydown', function (e) {
    keyboardState[e.key] = true

    if (e.key === ' ') { // spacebar lol

        const myPlayer = gameState.players[0]
        const snowball = {
            x: myPlayer.x, y: myPlayer.y,
            vx: 0, vy: 0
        }

        switch (myPlayer.rotation) {
            case FACING_DOWN:
                snowball.vy = SNOWBALL_SPEED
                break
            case FACING_UP:
                snowball.vy = -SNOWBALL_SPEED

```

```

        break
    case FACING_LEFT:
        snowball.vx = -SNOWBALL_SPEED
        break
    case FACING_RIGHT:
        snowball.vx = +SNOWBALL_SPEED
        break
    }

    myPlayer.snowballs.push(snowball)
}
}))
document.addEventListener('keyup', function (e) {
    keyboardState[e.key] = false
})

socket.on('stateUpdateForwardedByServer', function (player) {
    if (player.playerId === myPlayerId) {
        // ignore own update
        return
    }

    let playerWasFound = false
    for (let i = 0; i < gameState.players.length; ++i) {
        if (gameState.players[i].playerId === player.playerId) {
            gameState.players[i] = player
            playerWasFound = true
            break // don't continue looping, pointless
        }
    }

    if (!playerWasFound) {
        // player we haven't seen before
        gameState.players.push(player)
    }
})
</script>

</body>
</html>

```

server.js:

```

var express = require('express')
var app = express()

```

```

var path = require('path')
var server = require('http').createServer(app)
var io = require('socket.io')(server)
var port = process.env.PORT || 3000

server.listen(port, () => {
  console.log('Server listening at port %d', port)
})

// Routing
app.use(express.static(path.join(__dirname, 'public')))

io.on('connection', (socket) => {
  console.log('got a connection!')
  socket.on('stateUpdate', function (player) {
    console.log('got state update', player)
    io.sockets.emit('stateUpdateForwardedByServer', player)
  })
})

```

package.json(created via node):

```

{
  "name": "multiplayerbrowsergame",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "dependencies": {
    "express": "^4.17.1",
    "socket.io": "^2.3.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/bridgesams/project2.git"
  },
  "author": "Shaurya Sameer",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/bridgesams/project2/issues"
  },
  "homepage": "https://github.com/bridgesams/project2#readme"
}

```

