**CSEE 5590-0002/COMP-SCI 490-0002: Web/Mobile Programming**

**Web Lab - Source Code – Lab1**

**Team id – 4**

**Team Members – Roshna Toke (23)**

**Manaswini(24)**

**Yamini(3)**

**Tasc-2 : Snake Game**

1. app.component.html

<div class="snakegame-containar">  
 <div class="snakegame-header">  
 <h3 class="logo">Snake Game</h3>  
 <div class="scorecard-block">  
 <h3 class="score" [ngClass]="{'new-best-score': nextbestscore}">Score: {{score}}</h3>  
 <h3 class="best-score" [ngClass]="{'new-best-score': nextbestscore}">Best Score: {{Best\_Scor}}</h3>  
 </div>  
 </div>  
 <div class="row" *\*ngFor*="let ***column*** of board; let ***i*** = index;">  
 <div class="column" [ngStyle]="{'background-color': makeColors(***i***, ***j***)}" *\*ngFor*="let ***row*** of ***column***; let ***j*** = index"></div>  
 </div>  
 <div class="start-button" [ngClass]="{'disable-clicks': StartedGame}" (click)="displayMenu()">Start the Game</div>  
 <div class="new-game-menu" *\*ngIf*="CheckMenu">  
 <span class="menu-for-new-game">Mode Selection</span>  
 <div class="start-button new-gbutton" (click)="newGame(***mode***)" *\*ngFor*="let ***mode*** of all\_ModeTypes(all\_ModeType)">{{all\_ModeType[***mode***]}}</div>  
 </div>  
</div>

1. app.component.ts

import { ***Component***, ***HostListener*** } from '@angular/core';  
import { BestScoreHandler } from './app.bestscore.service';  
import { ***COMMANDS***, ***COLORS***, ***BOARD\_LENGTH***, ***GAME\_TYPES*** } from './app.costant';  
  
@Component({  
 selector: 'snake-game',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css'],  
 host: {  
 '(document:keydown)': 'keyEvent($event)'  
 }  
})  
export class AppComponent {  
 private timeinterval: number;  
 private currDirection: number;  
 private default\_mode = 'classic';  
 private gameOver = false;  
  
 public all\_ModeType = ***GAME\_TYPES***;  
 public all\_ModeTypes = ***Object***.keys;  
 public board = [];  
 public obstacles = [];  
 public score = 0;  
 public CheckMenu = false;  
 public StartedGame = false;  
 public nextbestscore = false;  
 public Best\_Scor = this.bestScoreService.retrieve();  
  
 private snake = {  
 direction: ***COMMANDS***.LEFT,  
 parts: [  
 {  
 x: -1,  
 y: -1  
 }  
 ]  
 };  
  
 private fruit = {  
 x: -1,  
 y: -1  
 };  
  
 constructor(  
 private bestScoreService: BestScoreHandler  
 ) {  
 this.setLengthBoard();  
 }  
  
 keyEvent(e: KeyboardEvent) {  
 if (e.keyCode === ***COMMANDS***.LEFT && this.snake.direction !== ***COMMANDS***.RIGHT) {  
 this.currDirection = ***COMMANDS***.LEFT;  
 } else if (e.keyCode === ***COMMANDS***.UP && this.snake.direction !== ***COMMANDS***.DOWN) {  
 this.currDirection = ***COMMANDS***.UP;  
 } else if (e.keyCode === ***COMMANDS***.RIGHT && this.snake.direction !== ***COMMANDS***.LEFT) {  
 this.currDirection = ***COMMANDS***.RIGHT;  
 } else if (e.keyCode === ***COMMANDS***.DOWN && this.snake.direction !== ***COMMANDS***.UP) {  
 this.currDirection = ***COMMANDS***.DOWN;  
 }  
 }  
 makeColors(col: number, row: number): string {  
 if (this.gameOver) {  
 return ***COLORS***.GAME\_OVER;  
 } else if (this.fruit.x === row && this.fruit.y === col) {  
 return ***COLORS***.FRUIT;  
 } else if (this.snake.parts[0].x === row && this.snake.parts[0].y === col) {  
 return ***COLORS***.HEAD;  
 } else if (this.board[col][row] === true) {  
 return ***COLORS***.SNAKEBODY;  
 } else if (this.default\_mode === 'obstacles' && this.checkObstacles(row, col)) {  
 return ***COLORS***.OBSTACLE;  
 }  
  
 return ***COLORS***.BOARD;  
 };  
  
 updatePositions(): void {  
 let newHead = this.repositionHead();  
 let me = this;  
  
 if (this.default\_mode === 'classic' && this.boardCollide(newHead)) {  
 return this.gameOver();  
 } else if (this.default\_mode === 'no\_walls') {  
 this.noWallTransition(newHead);  
 } else if (this.default\_mode === 'obstacles') {  
 this.noWallTransition(newHead);  
 if (this.obstacleCollide(newHead)) {  
 return this.gameOver();  
 }  
 }  
  
 if (this.selfCollide(newHead)) {  
 return this.gameOver();  
 } else if (this.fruitCollide(newHead)) {  
 this.eatFruit();  
 }  
  
 let oldTail = this.snake.parts.pop();  
 this.board[oldTail.y][oldTail.x] = false;  
  
 this.snake.parts.unshift(newHead);  
 this.board[newHead.y][newHead.x] = true;  
  
 this.snake.direction = this.currDirection;  
  
 setTimeout(() => {  
 me.updatePositions();  
 }, this.timeinterval);  
 }  
  
 repositionHead(): any {  
 let newHead = ***Object***.assign({}, this.snake.parts[0]);  
  
 if (this.currDirection === ***COMMANDS***.LEFT) {  
 newHead.x -= 1;  
 } else if (this.currDirection === ***COMMANDS***.RIGHT) {  
 newHead.x += 1;  
 } else if (this.currDirection === ***COMMANDS***.UP) {  
 newHead.y -= 1;  
 } else if (this.currDirection === ***COMMANDS***.DOWN) {  
 newHead.y += 1;  
 }  
  
 return newHead;  
 }  
  
 noWallTransition(part: any): void {  
 if (part.x === ***BOARD\_LENGTH***) {  
 part.x = 0;  
 } else if (part.x === -1) {  
 part.x = ***BOARD\_LENGTH*** - 1;  
 }  
  
 if (part.y === ***BOARD\_LENGTH***) {  
 part.y = 0;  
 } else if (part.y === -1) {  
 part.y = ***BOARD\_LENGTH*** - 1;  
 }  
 }  
  
 addObstacles(): void {  
 let x = this.randomNo();  
 let y = this.randomNo();  
  
 if (this.board[y][x] === true || y === 8) {  
 return this.addObstacles();  
 }  
  
 this.obstacles.push({  
 x: x,  
 y: y  
 });  
 }  
  
 checkObstacles(x, y): boolean {  
 let res = false;  
  
 this.obstacles.forEach((val) => {  
 if (val.x === x && val.y === y) {  
 res = true;  
 }  
 });  
  
 return res;  
 }  
  
 obstacleCollide(part: any): boolean {  
 return this.checkObstacles(part.x, part.y);  
 }  
  
 boardCollide(part: any): boolean {  
 return part.x === ***BOARD\_LENGTH*** || part.x === -1 || part.y === ***BOARD\_LENGTH*** || part.y === -1;  
 }  
  
 selfCollide(part: any): boolean {  
 return this.board[part.y][part.x] === true;  
 }  
  
 fruitCollide(part: any): boolean {  
 return part.x === this.fruit.x && part.y === this.fruit.y;  
 }  
  
 resetFruit(): void {  
 let x = this.randomNo();  
 let y = this.randomNo();  
  
 if (this.board[y][x] === true || this.checkObstacles(x, y)) {  
 return this.resetFruit();  
 }  
  
 this.fruit = {  
 x: x,  
 y: y  
 };  
 }  
  
 eatFruit(): void {  
 this.score++;  
  
 let tail = ***Object***.assign({}, this.snake.parts[this.snake.parts.length - 1]);  
  
 this.snake.parts.push(tail);  
 this.resetFruit();  
  
 if (this.score % 5 === 0) {  
 this.timeinterval -= 15;  
 }  
 }  
  
 gameOver(): void {  
 this.gameOver = true;  
 this.StartedGame = false;  
 let me = this;  
  
 if (this.score > this.Best\_Scor) {  
 this.bestScoreService.store(this.score);  
 this.Best\_Scor = this.score;  
 this.nextbestscore = true;  
 }  
  
 setTimeout(() => {  
 me.gameOver = false;  
 }, 500);  
  
 this.setLengthBoard();  
 }  
  
 randomNo(): any {  
 return ***Math***.floor(***Math***.random() \* ***BOARD\_LENGTH***);  
 }  
  
 setLengthBoard(): void {  
 this.board = [];  
  
 for (let i = 0; i < ***BOARD\_LENGTH***; i++) {  
 this.board[i] = [];  
 for (let j = 0; j < ***BOARD\_LENGTH***; j++) {  
 this.board[i][j] = false;  
 }  
 }  
 }  
  
 displayMenu(): void {  
 this.CheckMenu = !this.CheckMenu;  
 }  
  
 newGame(mode: string): void {  
 this.default\_mode = mode || 'classic';  
 this.CheckMenu = false;  
 this.nextbestscore = false;  
 this.StartedGame = true;  
 this.score = 0;  
 this.currDirection = ***COMMANDS***.LEFT;  
 this.gameOver = false;  
 this.timeinterval = 100;  
 this.snake = {  
 direction: ***COMMANDS***.LEFT,  
 parts: []  
 };  
  
 for (let i = 0; i < 3; i++) {  
 this.snake.parts.push({ x: 8 + i, y: 8 });  
 }  
  
 if (mode === 'obstacles') {  
 this.obstacles = [];  
 let j = 1;  
 do {  
 this.addObstacles();  
 } while (j++ < 9);  
 }  
  
 this.resetFruit();  
 this.updatePositions();  
 }  
}

1. app.component.css

.snakegame-header {  
 color: #fff;  
 padding: 5px 15px 5px 0px;  
 position: relative;  
}  
  
.snakegame-header>.scorecard-block {  
 display: inline-block;  
}  
  
.scorecard-block>.score {  
 position: absolute;  
 right: 8px;  
 top: -5px;  
}  
  
.scorecard-block>.best-score {  
 position: absolute;  
 right: 8px;  
 margin-top: 5px;  
 font-size: 12px;  
}  
  
.snakegame-header>.logo {  
 display: inline-block;  
 padding-left: 15px;  
}  
  
.snakegame-containar {  
 width: 600px; /\*468\*/  
 position: relative;  
 display: block;  
 margin: auto;   
 background-color: #464A43;  
 border-radius: 5px;  
}  
  
.row {  
 height: 26px;  
 margin-left: 15px;  
}  
  
.column {  
 border: 1px solid rgba(97, 131, 138, .1);  
 width: 24px;  
 height: 24px;  
 display: inline-block;  
}  
  
.start-button {  
 padding: 15px;  
 text-align: center;  
 background-color: #464A43;  
 color: white;  
 border-radius: 5px;  
}  
  
.start-button:hover {  
 opacity: 0.65;  
 cursor: pointer;  
}  
  
.start-button.new-gbutton {  
 margin: 0 105px 4px 105px;  
}  
  
.disable-clicks {  
 pointer-events: none;  
}  
  
.new-game-menu {  
 position: absolute;  
 top: 0;  
 width: 100%;  
 bottom: 0;  
 padding: 40% 0;  
 text-align: center;  
 background: rgba(0, 0, 0, 0.15);  
}  
  
.menu-for-new-game {  
 font-size: 1.17em;  
 margin-bottom: 13px;  
 display: block;  
 color: #fff;  
}  
  
.new-best-score {  
 animation: glow .5s infinite alternate;  
}  
  
@keyframes glow {  
 to {  
 text-shadow: 0 0 15px #ffff00;  
 }  
}  
  
@media screen and (max-width: 480px) {  
 .snakegame-containar {  
 width: 100%;  
 }  
 .column {  
 width: 5.55%;  
 height: 0;  
 padding-bottom: 5.06%;  
 box-sizing: border-box;  
 }  
 .row {  
 display: -webkit-box;  
 display: -moz-box;  
 display: -ms-flexbox;  
 display: -webkit-flex;  
 display: flex;  
 height: 0;  
 padding-bottom: 5.5%;  
 }  
 .new-game-menu {  
 width: 100%;  
 top: 8px;  
 padding-top: 30%;  
 font-size: 80%;  
 }  
 .start-button.new-gbutton {  
 margin: 0 15% 1% 15%;  
 }  
 @-moz-document url-prefix() {  
 .column {  
 min-height: 5.5vw;  
 }  
 }  
}

1. app.costant.ts

export const ***BOARD\_LENGTH*** = 22;  
  
export const ***COMMANDS*** = {  
 LEFT: 37,  
 UP: 38,  
 RIGHT: 39,  
 DOWN: 40  
};  
  
export const ***COLORS*** = {  
 GAME\_OVER: '#F70D34',  
 FRUIT: '#E13B21',  
 HEAD: '#2151E1',  
 SNAKEBODY: '#6d94e6',  
 BOARD: '#c4f1ae',  
 OBSTACLE: '#464A43'  
};  
  
export const ***GAME\_TYPES*** = {  
 classic: 'Classic',  
 no\_walls: 'No Walls',  
 obstacles: 'Obstacles'  
};

1. app.module.ts

import { BrowserModule } from '@angular/platform-browser';  
import { ***NgModule*** } from '@angular/core';  
  
import { AppComponent } from './app.component';  
import { BestScoreHandler } from './app.bestscore.service';  
  
@NgModule({  
 declarations: [  
 AppComponent  
 ],  
 imports: [  
 BrowserModule  
 ],  
 providers: [  
 BestScoreHandler  
 ],  
 bootstrap: [  
 AppComponent  
 ]  
})  
export class AppModule { }

1. app.bestscore.service

import { ***Injectable*** } from '@angular/core';  
  
@Injectable()  
export class BestScoreHandler {  
  
 private SnakeGame = 'Snake Game';  
  
 public store(score: number) {  
 ***localStorage***.setItem(this.SnakeGame, ***JSON***.stringify({ 'Best\_Scor': score }));  
 }  
  
 public retrieve() {  
 let storage = this.parse();  
 if (!storage) {  
 this.store(0);  
 storage = this.parse();  
 }  
  
 return storage.Best\_Scor;  
 }  
  
 private parse() {  
 return ***JSON***.parse(***localStorage***.getItem(this.SnakeGame));  
 }  
}