پروژه آزگرافیک

استاد سامعی

دانشگاه علم و فرهنگ (بهمن ۹۸)

دانشجویان: فاطمه سادات بطحائی، پرستو مالکی

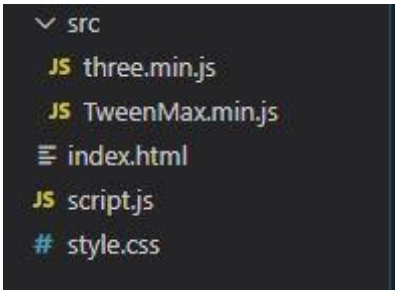
و علیرضا مرشد زاده طهرانی

بازسازی بازی stack

(چیدن خانه ها روی هم)

ساختار کلی پروژه

ساختار کلی پروژه تشکیل شده از یک فایل index.html و style.cssو script.js است.



از دو لایبری three.min.js و TweenMax.min.js برای این پروژه استفاده شده است.

- Index.html

در فایل index ساختار کلی صفحه وب و نحوه قرار گرفتن آن ها همراه با id های مورد نظر برای دسترسی و style دادن نوشته شده است.

index.html

```
<!DOCTYPE html>
<html lang="en" >
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="./style.css">

</head>
<body>

<meta name="viewport" content="width=device-width,user-scalable=no">

<div id="container">
    <div id="game"></div>
    <div id="score">0</div>
    <div id="instructions">Click (or press the spacebar) to place the block</div>
    <div class="game-over">
        <h2>Game Over</h2>
        <p>Try again!</p>
        <p>Click or push spacebar to start again</p>
    </div>
    <div class="game-ready">
        <div id="start-button">Start</div>
        <div></div>
    </div>
</div>


<script src='src\three.min.js'></script>
<script src='src\TweenMax.min.js'></script><script  src="./script.js"></script>

</body>
</html>
```

- Style.css

فایل style شامل کد های css برای زیبا سازی، نحوه نمایش، دادن ویژگی های مختلف به id ها و تگ ها و تغییر وضعیت آن ها در حالت بازی یا باخت است.

نمونه کد:

```css
#container .game-over * {
  -webkit-transition: opacity 0.5s ease, -webkit-transform 0.5s ease;
  transition: opacity 0.5s ease, -webkit-transform 0.5s ease;
  transition: opacity 0.5s ease, transform 0.5s ease;
  transition: opacity 0.5s ease, transform 0.5s ease, -webkit-transform 0.5s ease;
  opacity: 0;
  -webkit-transform: translatey(-50px);
          transform: translatey(-50px);
  color: #333344;
}
#container .game-over h2 {
  margin: 0;
  padding: 0;
  font-size: 40px;
}
#container .game-ready {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: -webkit-box;
  display: flex;
  -webkit-box-orient: vertical;
  -webkit-box-direction: normal;
          flex-direction: column;
  -webkit-box-align: center;
          align-items: center;
  justify-content: space-around;
}
#container .game-ready #start-button {
  -webkit-transition: opacity 0.5s ease, -webkit-transform 0.5s ease;
  transition: opacity 0.5s ease, -webkit-transform 0.5s ease;
  transition: opacity 0.5s ease, transform 0.5s ease;
  transition: opacity 0.5s ease, transform 0.5s ease, -webkit-transform 0.5s ease;
  opacity: 0;
  -webkit-transform: translatey(-50px);
          transform: translatey(-50px);
  border: 3px solid #333344;
  padding: 10px 20px;
  background-color: transparent;
  color: #333344;
```

- Script.js

و در نهایت اصلی ترین قسمت برنامه فایل js است که شامل تمام کد های ساختن صحنه، دوربین ، فانکشن های حرکت و ... می باشد.

توضیحات به صورت کامنت درون برنامه نوشته شده است.

```
use strict";"

console.clear();

var Stage = /** @class */ (function () {

function Stage() {

container //

var _this = this;

this.render = function () {

this.renderer.render(this.scene, this.camera);
```

```
;{
// برای اضافه کردن اشیا در صحنه
this.add = function (elem) {
this.scene.add(elem);
;{
//برای حذف اشیا از صحنه

this.remove = function (elem) {
this.scene.remove(elem);
;{
this.container = document.getElementById('game');
renderer //
web GL //رسم اشیا در قالب
this.renderer = new THREE.WebGLRenderer({
antialias: true,
alpha: false
;({
//به اندازه پنجرمون (عرض وطول) اندازه را تنظیم میکنیم

this.renderer.setSize(window.innerWidth, window.innerHeight);
//برای رنگ پس زمینه مانند ساختار تحت وب رنگ را میدهیم

this.renderer.setClearColor('#D0CBC7', 1);
this.container.appendChild(this.renderer.domElement);
scene //
//برای ساختن صحنه جدید

this.scene = new THREE.Scene();
camera //
//درواقع به renderer میگه با چه نمایی نشون بده

var aspect = window.innerWidth / window.innerHeight;
var d = 20;
//camera تعریف

this.camera = new THREE.OrthographicCamera(-d * aspect, d * aspect, d, -d, -100, 1000);
```

```
//موقعیت دوربین نشون میده
this.camera.position.x = 2;
this.camera.position.y = 2;
this.camera.position.z = 2;
//نقطه تمرکز دوربین را نشون میده
this.camera.lookAt(new THREE.Vector3(0, 0, 0));
light//
//نور با پرتو موازی از منبع مشخص
this.light = new THREE.DirectionalLight(0xffffff, 0.5);
this.light.position.set(0, 499, 0);
this.scene.add(this.light);
//همه صحنه را روشن ولی نه چندان زیاد یعنی نور محیط را مشخص میکند
// مقادیر رنگ و شدت نورا معلوم میکنه
this.softLight = new THREE.AmbientLight(0xffffff, 0.4);
this.scene.add(this.softLight);
window.addEventListener('resize', function () { return _this.onResize(); });
this.onResize();
{
Stage.prototype.setCamera = function (y, speed) {
if (speed === void 0) { speed = 0.3; }
TweenLite.to(this.camera.position, speed, { y: y + 4, ease: Power1.easeInOut });
TweenLite.to(this.camera.lookAt, speed, { y: y, ease: Power1.easeInOut });
;{
Stage.prototype.onResize = function () {
var viewSize = 30;
this.renderer.setSize(window.innerWidth, window.innerHeight);
this.camera.left = window.innerWidth / -viewSize;
this.camera.right = window.innerWidth / viewSize;
this.camera.top = window.innerHeight / viewSize;
this.camera.bottom = window.innerHeight / -viewSize;
this.camera.updateProjectionMatrix();
;{
return Stage;
;(){
var Block = /** @class */ (function () {
function Block(block) {
```

```
// وتنظیم اندازه وموقعیت
this.STATES = { ACTIVE: 'active', STOPPED: 'stopped', MISSED: 'missed' };
this.MOVE_AMOUNT = 12;
//طول وعرض و عمق
this.dimension = { width: 0, height: 0, depth: 0 };
this.position = { x: 0, y: 0, z: 0 };
this.targetBlock = block;
this.index = (this.targetBlock ? this.targetBlock.index : 0) + 1;
this.workingPlane = this.index % 2 ? 'x' : 'z';
this.workingDimension = this.index % 2 ? 'width' : 'depth';
// مجموعه ای از ابعاد از بلوک هدف، و یا پیش فرض.
this.dimension.width = this.targetBlock ? this.targetBlock.dimension.width : 10;
this.dimension.height = this.targetBlock ? this.targetBlock.dimension.height : 2;
this.dimension.depth = this.targetBlock ? this.targetBlock.dimension.depth : 10;
this.position.x = this.targetBlock ? this.targetBlock.position.x : 0;
this.position.y = this.dimension.height * this.index;
this.position.z = this.targetBlock ? this.targetBlock.position.z : 0;
this.colorOffset = this.targetBlock ? this.targetBlock.colorOffset : Math.round(Math.random() *
100);
// رنگ دادن
if (!this.targetBlock) {
this.color = 0x333344;
{
else {
var offset = this.index + this.colorOffset;
var r = Math.sin(0.3 * offset) * 55 + 200;
var g = Math.sin(0.3 * offset + 2) * 55 + 200;
var b = Math.sin(0.3 * offset + 4) * 55 + 200;
this.color = new THREE.Color(r / 255, g / 255, b / 255);
{
// وضعیت ها
this.state = this.index > 1 ? this.STATES.ACTIVE : this.STATES.STOPPED;
// تنظیم جهت
this.speed = -0.1 - (this.index * 0.005);
if (this.speed < -4)
this.speed = -4;
```

this.direction = this.speed;

// ساخت بلوک ها

//برای ساختن بلوک های مکعبی از تابع boxGeometry استفاده کردیم

//طول وعرض وعمق را مشخص کردیم

var geometry = new THREE.BoxGeometry(this.dimension.width, this.dimension.height, this.dimension.depth);

geometry.applyMatrix(new THREE.Matrix4().makeTranslation(this.dimension.width / 2, this.dimension.height / 2, this.dimension.depth / 2));

//متریال را از نوع MeshToonMaterial انتخاب کردیم

this.material = new THREE.MeshToonMaterial({ color: this.color, shading: THREE.FlatShading });

//در واقع geometryوmaterial رو سر هم میکنه

this.mesh = new THREE.Mesh(geometry, this.material);

//موقعیت مکعب ها رو تنظیم میکنیم

this.mesh.position.set(this.position.x, this.position.y + (this.state == this.STATES.ACTIVE ? 0 : 0), this.position.z);

if (this.state == this.STATES.ACTIVE) {

this.position[this.workingPlane] = Math.random() > 0.5 ? -this.MOVE_AMOUNT : this.MOVE_AMOUNT;

{

{

//موقعیت مکعب وقتی برای حرکت از جهت مخالف

Block.prototype.reverseDirection = function () {

this.direction = this.direction > 0 ? this.speed : Math.abs(this.speed);

;{

// دستورات شرطی برای موقعیت های مختلف قرار گرفتن مکعب ها. اگر قسمتی قرار نگرفت اگر کامل روی هم قرار گرفت و اگر کاملا خارج افتاد

Block.prototype.place = function () {

this.state = this.STATES.STOPPED;

var overlap = this.targetBlock.dimension[this.workingDimension] - Math.abs(this.position[this.workingPlane] - this.targetBlock.position[this.workingPlane]);

var blocksToReturn = {

plane: this.workingPlane,

direction: this.direction

;{

if (this.dimension[this.workingDimension] - overlap < 0.3) {

overlap = this.dimension[this.workingDimension];

blocksToReturn.bonus = true;

```
this.position.x = this.targetBlock.position.x;

this.position.z = this.targetBlock.position.z;

this.dimension.width = this.targetBlock.dimension.width;

this.dimension.depth = this.targetBlock.dimension.depth;

{

if (overlap > 0) {

var choppedDimensions = { width: this.dimension.width, height: this.dimension.height, depth:
this.dimension.depth };

choppedDimensions[this.workingDimension] -= overlap;

this.dimension[this.workingDimension] = overlap;

var placedGeometry = new THREE.BoxGeometry(this.dimension.width,
this.dimension.height, this.dimension.depth);

placedGeometry.applyMatrix(new THREE.Matrix4().makeTranslation(this.dimension.width / 2,
this.dimension.height / 2, this.dimension.depth / 2));

var placedMesh = new THREE.Mesh(placedGeometry, this.material);

var choppedGeometry = new THREE.BoxGeometry(choppedDimensions.width,
choppedDimensions.height, choppedDimensions.depth);

choppedGeometry.applyMatrix(new
THREE.Matrix4().makeTranslation(choppedDimensions.width / 2, choppedDimensions.height / 2,
choppedDimensions.depth / 2));

var choppedMesh = new THREE.Mesh(choppedGeometry, this.material);

var choppedPosition = {

x: this.position.x,

y: this.position.y,

z: this.position.z

;{

if (this.position[this.workingPlane] < this.targetBlock.position[this.workingPlane]) {

this.position[this.workingPlane] = this.targetBlock.position[this.workingPlane];

{

else {

choppedPosition[this.workingPlane] += overlap;

{

placedMesh.position.set(this.position.x, this.position.y, this.position.z);

choppedMesh.position.set(choppedPosition.x, choppedPosition.y, choppedPosition.z);

blocksToReturn.placed = placedMesh;

if (!blocksToReturn.bonus)

blocksToReturn.chopped = choppedMesh;

{
```

```
else {
this.state = this.STATES.MISSED;
{
this.dimension[this.workingDimension] = overlap;
return blocksToReturn;
;{
//تعریف زمان و حرکت
Block.prototype.tick = function () {
if (this.state == this.STATES.ACTIVE) {
var value = this.position[this.workingPlane];
if (value > this.MOVE_AMOUNT || value < -this.MOVE_AMOUNT)
this.reverseDirection();
this.position[this.workingPlane] += this.direction;
this.mesh.position[this.workingPlane] = this.position[this.workingPlane];
{
;{
return Block;
;(){
var Game = /** @class */ (function () {
function Game() {
//استیت های مخلفت بازی
var _this = this;
this.STATES = {
LOADING': 'loading','
PLAYING': 'playing','
READY': 'ready','
ENDED': 'ended','
RESETTING': 'resetting"
;{
// دریافت متغیر های مورد نیاز
this.blocks = [];
this.state = this.STATES.LOADING;
this.stage = new Stage();
this.mainContainer = document.getElementById('container');
this.scoreContainer = document.getElementById('score');
this.startButton = document.getElementById('start-button');
```

```
this.instructions = document.getElementById('instructions');
this.scoreContainer.innerHTML = '0';
this.newBlocks = new THREE.Group();
this.placedBlocks = new THREE.Group();
this.choppedBlocks = new THREE.Group();
this.stage.add(this.newBlocks);
this.stage.add(this.placedBlocks);
this.stage.add(this.choppedBlocks);
this.addBlock();
this.tick();
this.updateState(this.STATES.READY);
document.addEventListener('keydown', function (e) {
if (e.keyCode == 32)
this.onAction();_
;({
document.addEventListener('click', function (e) {
this.onAction();_
;({

{
// فانکشن بروزرسانی وضعیت بازی
Game.prototype.updateState = function (newState) {
for (var key in this.STATES)
this.mainContainer.classList.remove(this.STATES[key]);
this.mainContainer.classList.add(newState);
this.state = newState;
;{
// وضعیت های مختلف بازی و فراخوانی فانکشن مورد نیاز
Game.prototype.onAction = function () {
switch (this.state) {
case this.STATES.READY:
this.startGame();
break;
case this.STATES.PLAYING:
this.placeBlock();
break;
```

```
case this.STATES.ENDED:
this.restartGame();
break;
{
;{
// فانکشن شروع بازی
Game.prototype.startGame = function () {
if (this.state != this.STATES.PLAYING) {
this.scoreContainer.innerHTML = '0';
this.updateState(this.STATES.PLAYING);
this.addBlock();
{
;{
// فانکشن شروع دوباره بازی
Game.prototype.restartGame = function () {
var _this = this;
this.updateState(this.STATES.RESETTING);
var oldBlocks = this.placedBlocks.children;
var removeSpeed = 0.2;
var delayAmount = 0.02;
var _loop_1 = function (i) {
TweenLite.to(oldBlocks[i].scale, removeSpeed, { x: 0, y: 0, z: 0, delay: (oldBlocks.length - i) *
delayAmount, ease: Power1.easeIn, onComplete: function () { return
_this.placedBlocks.remove(oldBlocks[i]); } });
TweenLite.to(oldBlocks[i].rotation, removeSpeed, { y: 0.5, delay: (oldBlocks.length - i) *
delayAmount, ease: Power1.easeIn });
;{
for (var i = 0; i < oldBlocks.length; i++) {
loop_1(i);_
{
var cameraMoveSpeed = removeSpeed * 2 + (oldBlocks.length * delayAmount);
this.stage.setCamera(2, cameraMoveSpeed);
var countdown = { value: this.blocks.length - 1 };
TweenLite.to(countdown, cameraMoveSpeed, { value: 0, onUpdate: function () {
_this.scoreContainer.innerHTML = String(Math.round(countdown.value)); } });
this.blocks = this.blocks.slice(0, 1);
setTimeout(function () {
```

```
this.startGame();_

cameraMoveSpeed * 1000); ,{

;{

// فانکشن قرار دادن مکعب ها

Game.prototype.placeBlock = function () {

var _this = this;

var currentBlock = this.blocks[this.blocks.length - 1];

var newBlocks = currentBlock.place();

this.newBlocks.remove(currentBlock.mesh);

if (newBlocks.placed)

this.placedBlocks.add(newBlocks.placed);

if (newBlocks.chopped) {

this.choppedBlocks.add(newBlocks.chopped);

var positionParams = { y: '-=30', ease: Power1.easeIn, onComplete: function () { return
_this.choppedBlocks.remove(newBlocks.chopped); } };

var rotateRandomness = 10;

var rotationParams = {

delay: 0.05,

x: newBlocks.plane == 'z' ? ((Math.random() * rotateRandomness) - (rotateRandomness /
2)) : 0.1,

z: newBlocks.plane == 'x' ? ((Math.random() * rotateRandomness) - (rotateRandomness /
2)) : 0.1,

y: Math.random() * 0.1,

;{

if (newBlocks.chopped.position[newBlocks.plane] >
newBlocks.placed.position[newBlocks.plane]) {

positionParams[newBlocks.plane] = '+=' + (40 * Math.abs(newBlocks.direction));

{

else {

positionParams[newBlocks.plane] = '-=' + (40 * Math.abs(newBlocks.direction));

{

TweenLite.to(newBlocks.chopped.position, 1, positionParams);

TweenLite.to(newBlocks.chopped.rotation, 1, rotationParams);

{

this.addBlock();

;{

Game.prototype.addBlock = function () {
```

```
var lastBlock = this.blocks[this.blocks.length - 1];
if (lastBlock && lastBlock.state == lastBlock.STATES.MISSED) {
return this.endGame();
{
this.scoreContainer.innerHTML = String(this.blocks.length - 1);
var newKidOnTheBlock = new Block(lastBlock);
this.newBlocks.add(newKidOnTheBlock.mesh);
this.blocks.push(newKidOnTheBlock);
this.stage.setCamera(this.blocks.length * 2);
if (this.blocks.length >= 5)
this.instructions.classList.add('hide');
;{
Game.prototype.endGame = function () {
this.updateState(this.STATES.ENDED);
;{
Game.prototype.tick = function () {
var _this = this;
this.blocks[this.blocks.length - 1].tick();
this.stage.render();
requestAnimationFrame(function () { _this.tick(); });
;{
return Game;
;(){
var game = new Game();
```