2Д платформер во Babylon JS

Проектна задача по Напреден веб дизајн

Бојана Боцевска

Индекс бр.221227

Професор: Гоце Арменски

Асистент: Мила Додеска

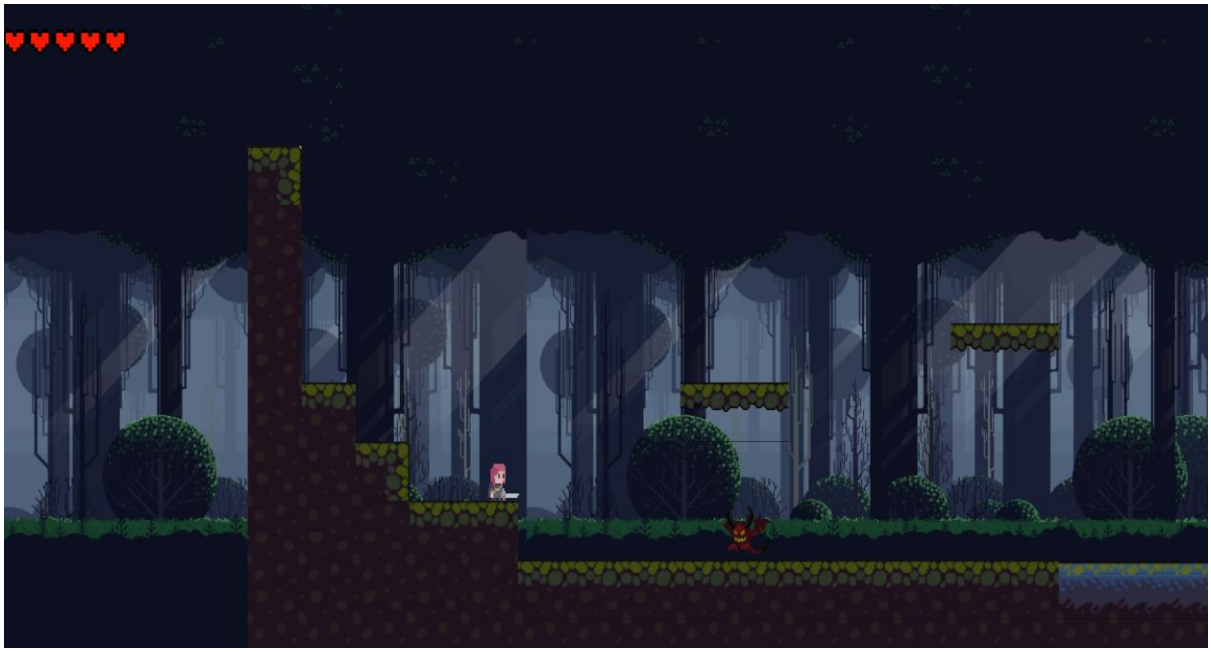
25.08.2025

Факултет за информатички науки и компјутерско инженерство - УКИМ, Скопје

Contents

Вовед

Тема на оваа практична задача е креирање на видео игра со алатката BABYLON JS која може да се игра во веб-прелистувач. При креирање на играта се користени бесплатно достапни елементи од корисници на платформата itch.io. Играта е замислена како 2Д платформер каде треба да одбегнуваш чудовишта за да стигнеш до целта. Тема на оваа практична задача е креирање на видео игра со користење на библиотеката BABYLON JS, која овозможува рендерирање и интерактивност директно во веб-прелистувач. Играта е дизајнирана како 2Д платформер во 3Д сценски простор, каде играчот управува со лик кој мора да се движи по нивоата, да избегнува непријатели и смртоносни пречки и да стигне до крајната цел. За развојот се користени бесплатно достапни спрајтови, текстури и други елементи од корисници на платформата itch.io.



Во процесот на креирање, внимателно е осмислен системот за колизии, а играчот е претставен преку невидлив „hit-box“ кој ја контролира физиката, додека визуелната репрезентација се постигнува преку спрајтови и анимации. Проектот ги прикажува основните принципи на 2Д платформер игра: движење, колизии, анимации и кориснички интерфејс, овозможувајќи интерактивно и динамично искуство за играчот.

Сегменти од код

Поставување на сцената

За правилно функционирање на играта, најпрво се дефинираат сцената и моторот. Моторот на Babylon.js ја обработува графиката и ја прикажува на HTML „canvas“ елемент кој е специјално поставен во документот. Сцената се иницијализира со бела боја на позадина, а во неа се додава хемисферично светло кое обезбедува правилно осветлување на сите 3Д објекти. Осветлувањето е важно бидејќи без него објектите би биле темни и непрепознатливи.

```
const canvas = document.getElementById("renderCanvas");
const engine = new BABYLON.Engine(canvas, true);
```

Камерата што се користи е ортографска камера. За разлика од перспективните камери, ортографската не додава перспектива, туку го прикажува просторот „рамно“, со што се симулира 2Д изглед. Ова е особено важно бидејќи иако сите објекти во сцената се 3Д, играта има карактеристики на 2Д платформер. Камерата е позиционирана паралелно со играчот и ја следи неговата позиција со мало задоцнување, што создава мазен ефект на движење. Големината на видното поле на камерата се дефинира преку параметарот orthoSize, а дополнително е овозможено прилагодување според односот на ширина и висина на прозорецот, со што се обезбедува правилно прикажување без деформации. Позадината на сцената се збогатува со „Layer“ објект кој прикажува статична слика (на пример шума), и со тоа се создава чувство на атмосфера и длабочина.

```
scene = new BABYLON.Scene(engine);
scene.clearColor = new BABYLON.Color3.White();

camera = new BABYLON.FreeCamera("camera1", new BABYLON.Vector3(0, 0, -10), scene);
camera.mode = BABYLON.Camera.ORTHOGRAPHIC_CAMERA;
camera.setTarget(BABYLON.Vector3.Zero());
// Define orthographic size
let ratio = engine.getRenderWidth() / engine.getRenderHeight();
let orthoSize = 6; // Adjust this number for zoom
camera.orthoLeft   = -orthoSize * ratio;
camera.orthoRight  =  orthoSize * ratio;
camera.orthoTop    =  orthoSize;
camera.orthoBottom = -orthoSize +1;
```

Играч

Откако е креирана камерата следен е самиот играч. За да може играчот да има колизии со елемнти од нивото, сите интерактибилни елементи, како и самиот играч, се тро-димензионални објекти. Играчот во сцената е дефиниран со два слоја. Првиот е 3Д коцка која е невидлива за корисникот, но има важна улога во механиката на играта. Таа служи за пресметка на колизии и интеракции со останатите елементи на сцената. Вториот слој е спрајт, кој визуелно го претставува карактерот на екранот. Овој спрајт е

слика што може да се анимира преку промена на фрејмови, со што се постигнува впечаток на движење и активности како одење или скокање.

```
//player box
chara = BABYLON.MeshBuilder.CreateBox("boxy", {width:0.8, height:1, depth:0.5}, scene);
//player sprite
const spriteManagerPlayer = new BABYLON.SpriteManager("playerManager", "resources/character.png", 1,
    { width: 225, height: 150 }, scene, BABYLON.Texture.NEAREST_SAMPLINGMODE);
playerSprite = new BABYLON.Sprite("pp", spriteManagerPlayer);
playerSprite.size = 1;
playerSprite.cellIndex = 0;

// Create a material for the box
boxMat = new BABYLON.StandardMaterial("boxMat", scene);
boxMat.diffuseTexture = new BABYLON.Texture("resources/chara.png", scene);
boxMat.diffuseTexture.hasAlpha = true;
chara.material = boxMat;
chara.position.x = 1;
chara.isVisible = false;
playerSprite.position = chara.position;
return scene;
```

Потребата од оваа комбинација се јавува затоа што спрајтовите во Babylon.js немаат вградена колизија, додека 3Д објектите ја поддржуваат. Со поставување на невидлива кутија за колизии и над неа видлив спрајт, се добива точна механика за интеракција, а истовремено изгледот останува дводимензионален како што е замислена играта.

Креирање и структура на нивоа

Секое ниво од играта е дефинирано во посебна JSON датотека која содржи дводимензионална матрица со броеви. При стартување на играта, главниот код вчитува соодветна датотека според избраното ниво.
Функцијата createTileGrid(scene, levelMap) ја обработува оваа матрица и за секој број создава соодветен објект во сцената. Матрицата се состои од редови и колони каде што секоја вредност претставува различен тип на елемент. Повеќето броеви се поврзани со различни плочки (земја, трева, вода), но постојат и специјални броеви:

- Број 2 означува непријател. Наместо плочка, во таа позиција се повикува функцијата createEnemy, која генерира непријател и го додава во глобалната низа window.enemies.
- Број 8 означува крајна точка на нивото. На таа позиција се додава анимирано знаме со спрајт кој се репродуцира во бесконечна анимација.

Секоја плочка која се создава добива материјал со текстура поврзана од ресурсната папка на проектот. Дополнително, при генерирање се пресметува точната позиција на објектот во координатниот систем за да се обезбеди правилно подредување без изместувања. Со ова функцијата createTileGrid ја враќа низата tiles, која го содржи целиот распоред на плочки за тековното ниво.

```
tile.material = mat;
tile.position.x = col * tileSize;
tile.position.y = yOffset + (rows - 1 - row) * tileSize;
tile.position.z = 0;
tiles.push(tile);
```

Кориснички интерфејс

Покрај сцена, играта содржи и кориснички интерфејс кој е изграден со библиотеката Babylon GUI. Интерфејсот се состои од панел кој прикажува срца што го претставуваат бројот на животи на играчот. Панелот е реализиран преку StackPanel кој овозможува хоризонтално или вертикално подредување на елементите. Во овој случај, срцата се прикажани хоризонтално во горниот лев агол на екранот.

```
// Health UI
window.guiTex = BABYLON.GUI.AdvancedDynamicTexture.CreateFullscreenUI("UI");
window.heartsPanel = new BABYLON.GUI.StackPanel();
window.heartsPanel.isVertical = false;
window.heartsPanel.horizontalAlignment = BABYLON.GUI.Control.HORIZONTAL_ALIGNMENT_LEFT;
window.heartsPanel.verticalAlignment = BABYLON.GUI.Control.VERTICAL_ALIGNMENT_TOP;
window.heartsPanel.left = "0px";
window.heartsPanel.top = "0px";
window.heartsPanel.paddingTop = "-450px";
guiTex.addControl(window.heartsPanel);
if (typeof buildHearts === "function") {buildHearts(6);}
```

Интерфејсот е фиксиран и независен од камерата, што значи дека останува на истата позиција без разлика на тоа како камерата се движи низ нивото. Ова е важно бидејќи обезбедува играчот секогаш да има видлив преглед на својот статус.

Главен циклус и логика на играта

Функционирањето на играта е овозможено преку главниот рендеринг циклус engine.runRenderLoop. Овој циклус се извршува континуирано и во секоја итерација ги извршува следните задачи:

- Го ажурира движењето на играчот според внесот од тастатура, преку функцијата updateMovement.
- Ја проверува интеракцијата со плочките, непријателите и специјалните елементи од нивото.
- Ги ажурира сите непријатели преку нивните функции за движење.
- Ја поместува камерата така што постепено се приближува кон позицијата на играчот, со што се добива ефект на мазно следење.
- Ја рендерира сцената за повторно прикажување на екранот.

Со ваквата структура, циклусот овозможува играта да биде интерактивна и непрекинато да реагира на сите активности на играчот и елементите во светот.

Со оваа организација на кодот се постигнува јасна поделба на логиката: сцената и камерата ја дефинираат визуелната поставеност, JSON датотеките ја обезбедуваат структурата на нивото, плочките и непријателите ја градат интерактивната средина, корисничкиот интерфејс го прикажува статусот на играчот, а главниот циклус овозможува динамично и непрекинато функционирање на целата игра.

Непријатели

Непријателите во играта се посебни објекти кои се креираат преку функцијата createEnemy. Сите непријатели се чуваат во глобалната низа window.enemies. Во текот на играта тие постојано се ажурираат преку функцијата enemyMovement, која им овозможува движење и интеракција со играчот. Овој пристап со централизирана низа на непријатели е ефикасен бидејќи овозможува сите да се обработуваат систематски во рамките на главниот циклус, без потреба од индивидуални повици за секој поединечно. Непријателите исто како карактерот на иргачот се составени од невидлива коцка и спрајт на неа за да може да има интеракција со играчот.

Карактеристики на играчот

Предходно беше спомнато дека карактерот на играчот има колизии со елементи од нивото. Во moveSet.js се сместени сите функции што овозможуваат карактерот да постои во светот. Најпрво се поставени потребните променливи и константи за состојбата на играчот како што се неговите состојби, животи на располагање, брзини на движење, почента позиција (во случај да треба играчот да се врати назад на почеток на нивото) и слично.

```
let grounded = false;
let vertical = 0;
const speed = 0.06;
const gravity = -0.003;
const jumpHeight = 0.13;
const floor = -70;
let playerHP = 5;
const maxHP = 5;
const respawnPoint = new BABYLON.Vector3(1, 1, 0);
let canTakeDamage = true;
const damageCooldown = 1000;
let currentState = "idle";
```

Функцијата setupPlayer(box, tiles, scene) се повикува пред да започне рендерирањето и ја подготвува сцената за правилно ракување со колизии. Најпрво се овозможува Babylon системот за колизии (scene.collisionsEnabled = true), без кој ниту еден објект не би можел да се судира со друг. Потоа, на играчот му се активира можноста за судири (box.checkCollisions = true), но истовремено се оневозможува можноста да биде „pickable“ (isPickable = false), за да не може зраците за детекција (raycasts) да се одбиваат од него самиот.

За да се добие попрецизна колизија, на играчот не му се користи оригиналната форма, туку се задава елипсоиден hit-box преку параметарот box.ellipsoid. Овој хит-бокс е со радиус од 0.4 по X и Z-оска и висина од 0.5 по Y-оска. Дополнително, целата елипса е поместена нагоре со box.ellipsoidOffset = new BABYLON.Vector3(0, 0.05, 0), за да биде центрирана околу ликот наместо да потоне под подлогата.

Потоа, сите плочки кои се дел од нивото се конфигурираат да учествуваат во колизии (tile.checkCollisions = true). За разлика од играчот, тие треба да бидат „pickable“, бидејќи зраците кои се користат за проверка на контакт со земјата треба да можат да ги детектираат. Ако некоја плочка нема име, таа автоматски добива „tile“ како ознака. Со оваа конфигурација, играчот може реално да стои и да се движи врз подлогата, да се судира со ѕидови или други плочки, а истовремено системот може со точни зраци да проверува дали играчот се наоѓа на земја или е во воздух.

```
function setupPlayer(box, tiles, scene) {

    scene.collisionsEnabled = true;
    const halfHeight = 0.5;
    box.checkCollisions = true;
    box.isPickable = false;
    box.ellipsoid = new BABYLON.Vector3(0.4, 0.5, 0.4);
    box.ellipsoidOffset = new BABYLON.Vector3(0, 0.05, 0);

    for (let tile of tiles) {
        tile.checkCollisions = true;
        tile.isPickable = true;
        if (!tile.name) tile.name = "tile";
    }
}
```

Следна е функцијата setPlayerState(newState, playerSprite), која е одговорна за најголем дел од анимациите на карактерот. Според тоа од каде е повикана се чита што е сместено во newState и се повикуваа соодветната анимација.

```
function setPlayerState(newState, playerSprite, tiles) {
    if (currentState === newState) return;
    currentState = newState;

    switch (newState) {
        case "idle":
            playerSprite.playAnimation(6, 9, true, 160);
            break;
        case "run":
            playerSprite.playAnimation(0, 5, true, 100);
            break;
        case "jump":
            playerSprite.playAnimation(12, 14, false, 120);
            break;
        case "dead":
            playerSprite.playAnimation(36, 40, false, 150);
            break;
    }
}

if (moveX !== 0) {
    setPlayerState("run", playerSprite);
} else {
    setPlayerState("idle", playerSprite);
    box.moveWithCollisions(new BABYLON.Vector3(0, -0.01, 0));
}
```

Во циклусот за рендерирање постојано се повикува функцијата updateMovement (inputMap, chara, scene , playerSprite, tiles); која ги контролира сите акции на карактерот. Играчот го контролира каракерот стандардно со wad копчињата, каде „a“ и „d“ се за движење лево и десно, а „w“ е за скокање. Позицијата на спрајтот секогаш се синхронизира со hit-box објектот за да се прикажува точно движењето на ликот. Вертикалното движење дополнително се ажурира со гравитација, со што се симулира паѓање. За да се спречи играчот да пропадне низ мапата, од карактерот се испраќаат зраци надолу кои проверуваат дали има подлога под него. Ако има контакт, ликот се смета за „приземјен“.

```
playerSprite.position = box.position;
let moveX = 0;
if (inputMap["a"] || inputMap["A"] || inputMap["ArrowLeft"]) {
    moveX = -speed;
    playerSprite.invertU = true;
}
if (inputMap["d"] || inputMap["D"] || inputMap["ArrowRight"]) {
    moveX =  speed;
    playerSprite.invertU = false;
}
if ((inputMap["w"] || inputMap["W"] || inputMap["ArrowUp"]) && grounded) {
    vertical = jumpHeight;
    grounded = false;
    setPlayerState("jump", playerSprite);
}
```

При проверките, ако ликот застане врз плочка од тип 7, инстантно го елиминира играчот, а ако застане на плочка од тип 8, веднаш се пренасочува кон страната за завршување на нивото. Дополнително, постои и безбедносна проверка – ако играчот пропадне под одредена вредност на „y“, се ресетира неговата позиција на подот за да не излезе целосно од сцената.

```
box.moveWithCollisions(new BABYLON.Vector3(moveX, 0, 0));
box.moveWithCollisions(new BABYLON.Vector3(0, vertical, 0));

const feetY = box.position.y + box.ellipsoidOffset.y - box.ellipsoid.y;
const rayOrigin = new BABYLON.Vector3(box.position.x, feetY + 0.06, box.position.z);
const ray = new BABYLON.Ray(rayOrigin, new BABYLON.Vector3(0, -1, 0), 0.12);
const pick = scene.pickWithRay(ray, (mesh) =>
    mesh !== box && mesh.isPickable && mesh.name && mesh.name.startsWith("tile"));

if (pick.hit && vertical <= 0) {
    const hitTile = pick.pickedMesh;
    // Check if it's lava
    if (hitTile.tileType === 7) {
        takeDamage(5, box, playerSprite);
    }
    if (hitTile.tileType === 8) {
        window.location.href = "end.html";
    }
```

Последниот сегмент од функцијата разгледува две сценарија кога играчот ќе наиде на непријател. За да го убиеш непријателот треба да скокнеш на него. При проверка на позициите на играчот и непријателот ако нивните гранични линии се преклопуваат и играчот се наоѓа над непријателот се смета дека непријателот е убиен, ако не се исполнува тој услов се смета дека играчот е повреден.

```
if (window.enemies && window.enemies.length > 0) {
    for (let i = window.enemies.length - 1; i >= 0; i--) {
        let enemy = window.enemies[i];

        const playerBox = box.getBoundingInfo().boundingBox;
        const enemyBox = enemy.getBoundingInfo().boundingBox;

        const playerMin = playerBox.minimumWorld;
        const playerMax = playerBox.maximumWorld;

        const enemyMin = enemyBox.minimumWorld;
        const enemyMax = enemyBox.maximumWorld;

        const collision =
            playerMax.x > enemyMin.x &&
            playerMin.x < enemyMax.x &&
            playerMax.y > enemyMin.y &&
            playerMin.y < enemyMax.y;

        if (collision) {
            // Player jumps on top of enemy
            if (vertical < 0 && box.position.y > enemy.position.y + 0.2) {
                enemy.dispose();
                window.enemies.splice(i, 1);
                vertical = jumpHeight / 2;
                grounded = false;
                playerSprite.playAnimation(18,21, false, 80);
            } else {
                takeDamage(1, box, playerSprite);
            }
        }
    }
}
```

Во случај играчот да е повреден се повикува функцијата takeDamage(amount, box, playerSprite). Најпрво кодот проверува дали играчот може да биде повреден, со што се спречува континуирано одземање на животи преку краток „cooldown“. Ако може да биде повреден, се намалува бројот на животи и се проверува дали играчот е мртов. Доколку бројот на животи е нула или помал, играчот умира, се враќа на почетната позиција (respawnPoint) и му се ресетираат состојбите за движење и гравитација. Ако пак играчот преживее, се одбива наназад (со зголемување на вертикалната вредност) и може да продолжи со играта. На крај секогаш се повикува „buildHearts“ за да се ажурира графичкиот приказ на животите.

```
function takeDamage(amount, box, playerSprite) {
    if (!canTakeDamage) return;

    canTakeDamage = false;
    setTimeout(() => { canTakeDamage = true; }, damageCooldown);
    playerHP -= amount;
    playerSprite.playAnimation(30, 32, false, 80);
    if (playerHP <= 0) {
        setPlayerState("dead", playerSprite);
        playerHP = maxHP;
        box.position = respawnPoint.clone();
        vertical = 0;
        grounded = true;

        console.log("Player died! Respawned at start.");
    } else {
        vertical = jumpHeight / 2;
        grounded = false;
    }
    // Update UI
    buildHearts(playerHP);
}
```

Функцијата buildHearts (hp) го управува корисничкиот интерфејс за животите. Прво ги брише претходните икони од панелот, а потоа повторно ги генерира сите срциња во зависност од максималниот број животи. За секој живот се прикажува полна или празна икона, со што играчот секогаш има точен визуелен приказ на моменталните животи со кои располага.

```
function buildHearts(hp) {
    heartsPanel.clearControls();
    heartImages = [];
    for (let i = 0; i < maxHP; i++) {
        const heart = new BABYLON.GUI.Image("heart" + i,
            i < hp ? "resources/hearts_full.png" : "resources/hearts_empty.png"
        );
        heart.width = "40px";
        heart.height = "40px";
        heartImages.push(heart);
        heartsPanel.addControl(heart);
    }
}
```

Анимација на карактерот

Функцијата playAnimation(startFrame, endFrame, loop, delay) овозможува прикажување на секвенца од слики (рамки) од спрајт листот за да се добие анимација. Аргументите `startFrame` и `endFrame` го одредуваат опсегот на рамки што ќе се прикажат, додека која слика е во која рамка се одредува кога ќе се дефинира спрајтот. За карактерот на играчот, датотеката на спрајт листата има големина 1350x1200 пиксели и е поделена на 48 рамнки. Параметарот „loop“ дефинира дали анимацијата ќе се повторува бесконечно или ќе заврши по еднаш извршување. Последниот аргумент, „delay“, ја задава брзината на анимацијата, односно за колку време (во милисекунди) ќе се смени секоја рамка. На овој начин преку една функција се контролираат сите можни анимации на ликот, како трчање, скокање, мирување или примање удар.

Користени материјали

Спрајт лист за карактерот на играчот - https://edermunizz.itch.io/pixel-art-rpg-character-creator

Сликата за позадината - https://edermunizz.itch.io/free-pixel-art-forest

Спрајт за непријателите - https://xzany.itch.io/flying-demon-2d-pixel-art

Спрајтови за останатите елементи од нивото - https://craftpix.net/freebies/free-swamp-game-tileset-pixel-art/?srsltid=AfmBOoq1LHjE6uyJ_frZJaGCgSRPG77iuTcbGmL6aDChdMG2IF_NodgS

Елементи користени за елементите од менито - https://crusenho.itch.io/complete-ui-book-styles-pack