Απαλλακτική Εργασία Τεχνολογίες Ανάπτυξης Ηλεκτρονικών Παιχνιδιών

Χρήστος Γκουτζηγιάννης - Π18030

Χρήστος Λέσκος - Π18093

Εισαγωγή

Για την απαλλακτική εργασία μας αποφασίσαμε να υλοποιήσουμε ένα πιστό αντίγραφο της πρώτη πίστα του Super Mario Bros. Συγκεκριμένα υλοποιήσαμε όλα τα animations, τους εχθρούς, ακριβώς το ίδιο επίπεδο με όλα τα assets και την λειτουργικότητα των power ups.

Περιγραφή του Προβλήματος

Το ζητούμενο της εργασίας ήταν η υλοποίηση ενός 2D ή 3D παιχνιδιού στο περιβάλλον Unity3D. Αποφασίσαμε πως θα ήταν ενδιαφέρον να υλοποιήσουμε ένα 2D παιχνίδι κυρίως επειδή έχουμε εκτεθεί στην ανάπτυξη 3D παιχνιδιού μέσω του μαθήματος Εικονική Πραγματικότητα. Επίσης το Super Mario Bros, παρότι παλιό παιχνίδι, παρουσιάζει ιδιαίτερο ενδιαφέρον στην υλοποίηση του, καθώς έχει αρκετές λειτουργίες που είναι περίπλοκες. Τέλος θέλαμε πολύ να αναβιώσουμε ένα κλασικό παιχνίδι της παιδικής μας ηλικίας.

Αναλυτικη Παρουσίαση Φάσεων Ανάπτυξης

Ανάλυση

Στην φάση της Ανάλυσης προσπαθήσαμε να συλλέξουμε και να καταγράψουμε όσες περισσότερες πληροφορίες μπορούμε για το παιχνίδι και την λειτουργικότητα του. Συγκεκριμένα μας ενδιέφερε το design του επιπέδου, η αλληλεπίδραση του παίκτη(Mario) με τους αντιπάλους και οι χειρισμοί και ο τρόπος κίνησης του χαρακτήρα. Τις παραπάνω πληροφορίες αντλήσαμε από βίντεο στο YouTube αλλά κυρίως από το Super Mario Wiki. Στο Super Mario Wiki βρήκαμε όλες τις λεπτομέρειες για την λειτουργικότητα αλλά και φωτογραφίες από το πρώτο επίπεδο.

Σχεδίαση

Στην φάση της σχεδίασης χωρίσαμε την λειτουργικότητα στα διαφορετικα GameObjects(Mario, Koopa, Goomba, Mystery Block) και βρήκαμε όλα τα απαραίτητα assets στο OpenGameArt. Επίσης σχεδιάσαμε τα διαφορετικά Scripts και την λειτουργικότητα που θα έπρεπε να παρέχουν.

Υλοποίηση

Κατα την υλοποίηση χρησιμοποιήσαμε όλες τις φωτογραφίες και τα assets που είχαμε συλλέξει έτσι ώστε να σχεδιάσουμε το επίπεδο όσο πιο πιστά στο αρχικό γίνεται. Επίσης υλοποιήσαμε όλα τα scripts σε C# για την κίνηση του παίκτη, την κίνηση των αντιπάλων, τα animations αλλά και διάφορα στοιχεία του περιβάλλοντος όπως τα mystery boxes και τα power ups.

Πηγαίος Κώδικας

Πίνακας Scripts

AnimatedSprite.cs	Δέχεται ως input τα διαφορετικά sprites ενός αντιπάλου και το επιθυμητό frame rate, είναι υπεύθυνο για την αλλαγή των sprites έτσι ώστε να λειτουργούν τα animations.
BlockCoin.cs	Υπεύθυνο για το animation και την κίνηση ενός coin όταν ο Mario χτυπάει ένα mystery box.
BlockHit.cs	Διαχειρίζεται το collision μεταξύ του Mario και ενός block.
BlockItem.cs	Υπεύθυνο για την κίνηση ενός block όταν το χτυπάει ο Mario.
DeathAnimation.cs	Δέχεται ως input το sprite που παίζει μετά το θάνατο οποιουδήποτε χαρακτήρα και παίζει το animation.
DeathBarrier.cs	Αφού παίζει το death animation, ο χαρακτήρας κινείται προς τα κάτω εκτός της κάμερας. Αν είναι ο Mario ξεκινάει ξανά το επίπεδο, αν είναι κάποιος αντίπαλος καταστρέφεται.
EntityMovement.cs	Διαχειρίζεται την κίνηση των αυτόνομων αντιπάλων. Συγκεκριμένα ξεκινάει την κίνηση

	τους όταν γίνουν ορατοί από την κάμερα.
Extensions.cs	Utility functions που χρησιμοποιούμε στα υπόλοιπα Scripts.
FlagPole.cs	Διαχειρίζεται το collision του παίκτη και ξεκινάει το animation στο τέλος του παιχνιδιού όταν ο παίκτης ολοκληρώσει το επίπεδο.
GameManager.cs	Μετράει τις ζωές και τα coins που έχει ο παίκτης και φορτώνει το επίπεδο.
Goomba.cs	Υλοποιεί την λειτουργικότητα του αντιπάλου Goomba. Συγκεκριμένα αν έρθει σε επαφή με τον Mario του αφαιρεί μια ζωή, αλλά αν ο Mario πέσει πάνω του τον σκοτώνει.
Koopa.cs	Υλοποιεί την λειτουργικότητα του αντιπάλου Koopa. Συγκεκριμένα αν έρθει σε επαφή με τον Mario του αφαιρεί μια ζωή, αλλά αν ο Mario πέσει πάνω του τον σκοτώνει και εξαπολύει ένα κέλυφος το οποίο σκοτώνει άλλους αντιπάλους.
Pipe.cs	Υλοποιεί την είσοδο του Mario στους σωλήνες.
Player.cs	Ελέγχει την παρουσίαση του Mario. Δηλαδή αν ο Mario δεχθεί ένα χτύπημα από αντίπαλο μικραίνει ενώ, αν πάρει ένα power up μεγαλώνει.
PlayerMovement.cs	Διαχειρίζεται την κίνηση του Mario.
PlayerSpriteRenderer.cs	Διαχειρίζεται τα animations του Mario.
PowerUp.cs	Διαχειρίζεται τα διαφορετικά τύπου power ups.
SideScrolling.cs	Κεντράρει την κάμερα στον Mario.

Ανάλυση Σημαντικών Scripts

PlayerMovement.cs

```
using UnityEngine;

[RequireComponent(typeof(Rigidbody2D))]
public class PlayerMovement : MonoBehaviour
{
    private new Camera camera;
    private new Rigidbody2D rigidbody;
    private new Collider2D collider;

    private Vector2 velocity;
    private float inputAxis;

    public float moveSpeed = 8f;
    public float maxJumpHeight = 5f;
    public float maxJumpTime = 1f;
    public float jumpForce => (2f * maxJumpHeight) / (maxJumpTime / 2f);
    public float gravity => (-2f * maxJumpHeight) / Mathf.Pow(maxJumpTime / 2f, 2f);

    public bool grounded { get; private set; }
    public bool jumping { get; private set; }
    public bool running => Mathf.Abs(velocity.x) > 0.25f || Mathf.Abs(inputAxis) > 0.25f;
    public bool sliding => (inputAxis > 0f && velocity.x < 0f) || (inputAxis < 0f && velocity.x > 0f);
    public bool falling => velocity.y < 0f && !grounded;

    private void Awake()
    {
        camera = Camera.main;
        rigidbody = GetComponent<Rigidbody2D>();
        collider = GetComponent<Collider2D>();
    }

    private void OnEnable()
    {
        rigidbody.isKinematic = false;
        collider.enabled = true;
        velocity = Vector2.zero;
        jumping = false;
    }

    private void OnDisable()
    {
        rigidbody.isKinematic = true;
        collider.enabled = false;
        velocity = Vector2.zero;
        jumping = false;
    }

    private void Update()
    {
        HorizontalMovement();

        grounded = rigidbody.Raycast(Vector2.down);

        if (grounded) {
            GroundedMovement();
        }

        ApplyGravity();
    }

    private void FixedUpdate()
    {
        // move mario based on his velocity
        Vector2 position = rigidbody.position;
        position += velocity * Time.fixedDeltaTime;

        // clamp within the screen bounds
        Vector2 leftEdge = camera.ScreenToWorldPoint(Vector2.zero);
        Vector2 rightEdge = camera.ScreenToWorldPoint(new Vector2(Screen.width, Screen.height));
        position.x = Mathf.Clamp(position.x, leftEdge.x + 0.5f, rightEdge.x - 0.5f);

        rigidbody.MovePosition(position);
    }

    private void HorizontalMovement()
    {
        // accelerate / decelerate
        inputAxis = Input.GetAxis("Horizontal");
        velocity.x = Mathf.MoveTowards(velocity.x, inputAxis * moveSpeed, moveSpeed * Time.deltaTime);

        // check if running into a wall
        if (rigidbody.Raycast(Vector2.right * velocity.x)) {
            velocity.x = 0f;
        }

        // flip sprite to face direction
        if (velocity.x > 0f) {
            transform.eulerAngles = Vector3.zero;
        } else if (velocity.x < 0f) {
            transform.eulerAngles = new Vector3(0f, 180f, 0f);
        }
    }

    private void GroundedMovement()
    {
        // prevent gravity from infinitly building up
        velocity.y = Mathf.Max(velocity.y, 0f);
        jumping = velocity.y > 0f;

        // perform jump
        if (Input.GetButtonDown("Jump"))
        {
            velocity.y = jumpForce;
            jumping = true;
        }
    }

    private void ApplyGravity()
    {
        // check if falling
        bool falling = velocity.y < 0f || !Input.GetButton("Jump");
        float multiplier = falling ? 2f : 1f;

        // apply gravity and terminal velocity
        velocity.y += gravity * multiplier * Time.deltaTime;
        velocity.y = Mathf.Max(velocity.y, gravity / 2f);
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.layer == LayerMask.NameToLayer("Enemy"))
        {
            // bounce off enemy head
            if (transform.DotTest(collision.transform, Vector2.down))
            {
                velocity.y = jumpForce / 2f;
                jumping = true;
            }
        }
        else if (collision.gameObject.layer != LayerMask.NameToLayer("PowerUp"))
        {
            // stop vertical movement if mario bonks his head
            if (transform.DotTest(collision.transform, Vector2.up)) {
                velocity.y = 0f;
            }
        }
    }

}
```

Το συγκεκριμένο Script είναι ίσως το πιο σημαντικό, καθώς δέχεται το input του χρήστη και υλοποιεί την κίνηση του Mario. Αρχικά ορίζουμε κάποιες δημόσιες μεταβλητές όπου ορίζουμε την ταχύτητα κίνησης, τη δύναμη του jump, την επιτάχυνση και τον χρόνο που ο Mario βρίσκεται στον αέρα. Επίσης ορίζουμε boolean μεταβλητές με τις οποίες ελέγχουμε την κατάσταση στην οποία βρίσκεται ο Mario(αν είναι στον αέρα, στο έδαφος, κτλ.). Κατά την αρχικοποίηση παίρνουμε references για τα διάφορα Components που θα χρειαστούμε, όπως την κάμερα και το RigidBody2D. Στην μέθοδο Update(), χρησιμοποιούμε τις μεθόδους HorizontalMovement(), ApplyGravity() και ελέγχουμε αν ο Mario βρίσκεται στο έδαφος. Στην μέθοδο HorizontalMovement() παίρνουμε το input του χρήστη και αλλάζουμε αντίστοιχα την επιτάχυνση του Mario. Στην μέθοδο ApplyGravity() εφαρμόζουμε μια δύναμη προς τα κάτω η οποία παίζει τον ρόλο της βαρύτητας και περιορίζουμε την μέγιστη ταχύτητα του Mario. Παρότι στις παραπάνω μεθόδους αλλάζουμε την μεταβλητή velocity, αυτό δεν επιφέρει κάποια αλλαγή στον Mario. Συγκεκριμένα επειδή χρησιμοποιούμε το Component RgidBody2D, αυτό σημαίνει ότι ο Mario υπάγεται στους κανόνες του physics engine του Unity. Το Unity επιβάλλει οποιαδήποτε αλλαγή που επηρεάζει τα physics του παιχνιδιού να γίνεται στην μέθοδο FixedUpdate(), έτσι ώστε να διασφαλίσει ότι τα physics δουλεύουν σταθερά, ανεξάρτητα από το frame rate του παιχνιδιού. Έτσι στην μέθοδο FixedUpdate() εφαρμόζουμε τις παραπάνω αλλαγές στο RigidBody 2D Component του Mario. Τέλος στη μέθοδο OnCollisonEnter2D() ελέγχουμε για collison με αντιπάλους ή power ups. Η κίνηση, το collision detection και το animation των αντιπάλων γίνεται με παρόμοιο τρόπο και για αυτό δεν το παρουσιάζουμε αναλυτικά εδώ.

GameManager.cs

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set;
}
    public int world { get; private set; }
    public int stage { get; private set; }
    public int lives { get; private set; }
    public int coins { get; private set; }

    private void Awake()
    {
        if (Instance != null) {
            DestroyImmediate(gameObject);
        } else {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
    }

    private void OnDestroy()
    {
        if (Instance == this) {
            Instance = null;
        }
    }

    private void Start()
    {
        Application.targetFrameRate = 60;

        NewGame();
    }

    public void NewGame()
    {
        lives = 3;
        coins = 0;

        LoadLevel(1, 1);
    }

    public void GameOver()
    {

        NewGame();
    }

    public void LoadLevel(int world, int stage)
    {
        this.world = world;
        this.stage = stage;

        SceneManager.LoadScene($"{world}-{stage}");
    }

    public void NextLevel()
    {
        LoadLevel(world, stage + 1);
    }

    public void ResetLevel(float delay)
    {
        Invoke(nameof(ResetLevel), delay);
    }

    public void ResetLevel()
    {
        lives--;

        if (lives > 0) {
            LoadLevel(world, stage);
        } else {
            GameOver();
        }
    }

    public void AddCoin()
    {
        coins++;

        if (coins == 100)
        {
            coins = 0;
            AddLife();
        }
    }

    public void AddLife()
    {
        lives++;
    }

}
```

Ο GameManager υλοποιεί το Singleton pattern, έτσι ώστε να διασφαλίσουμε ότι υπάρχει ένα μονάδικο instance του GameManager. Σε αυτό Script δηλώνουμε μεταβλητές(lives, coins) και μεθόδους AddCoin(), AddLife() για να παρακολουθούμε πόσες ζωές και πόσα coins έχει ο Mario. Επίσης έχουμε τις μεθόδους NewGame(), GameOver() και ResetLevel() για να φορτώνουμε το επίπεδο. Τέλος υλοποιήσαμε κάποιες μεθόδους για να φορτώνουμε επόμενα επίπεδα, έτσι ώστε να τα φορτώνουμε αν τα υλοποιήσουμε στο μέλλον.

FlagPole.cs

```
using System.Collections;
using UnityEngine;

public class FlagPole : MonoBehaviour
{
    public Transform flag;
    public Transform poleBottom;
    public Transform castle;
    public float speed = 6f;
    public int nextWorld = 1;
    public int nextStage = 1;

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Player"))
        {
            StartCoroutine(MoveTo(flag, poleBottom.position));
            StartCoroutine(LevelCompleteSequence(other.transform));
        }
    }

    private IEnumerator LevelCompleteSequence(Transform player)
    {
        player.GetComponent<PlayerMovement>().enabled = false;

        yield return MoveTo(player, poleBottom.position);
        yield return MoveTo(player, player.position + Vector3.right);
        yield return MoveTo(player, player.position + Vector3.right + Vector3.down);
        yield return MoveTo(player, castle.position);

        player.gameObject.SetActive(false);

        yield return new WaitForSeconds(2f);

        GameManager.Instance.LoadLevel(nextWorld, nextStage);
    }

    private IEnumerator MoveTo(Transform subject, Vector3 position)
    {
        while (Vector3.Distance(subject.position, position) > 0.125f)
        {
            subject.position = Vector3.MoveTowards(subject.position, position, speed * Time.deltaTime);
            yield return null;
        }

        subject.position = position;
    }

}
```

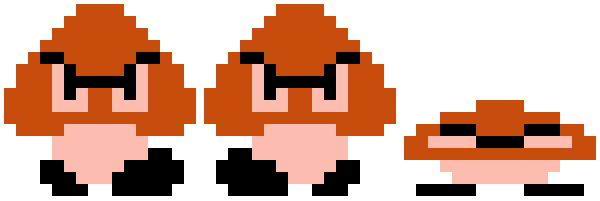
Το FlagPole.cs διαχειρίζεται το τέλος του επιπέδου και για αυτό επιλέξαμε να το δείξουμε αναλυτικότερα. Αρχικά έχουμε υλοποιήση την μέθοδο MoveTo, η οποία δέχεται ως input ένα GameObject και ένα position και σταδιακά μετακινεί το GameObject στο παραπάνω position. Έτσι όταν ανιχνεύσουμε collision με τον παίκτη στη μέθοδο OnTriggerEnter2D() μετακινούμε την σημαία προς τα κάτω και τον Mario στο κάστρο με τη μέθοδο MoveTo(),

Animations

Τα animations που χρησιμοποιήθηκαν για τους αντιπάλους είναι αρκετά απλά καθώς κουνιούνται σε μια προκαθορισμένη πορεία, χωρίς το input του χρήστη. Συγκεκριμένα για κάθε αντίπαλο έχουμε τρία animations:

- Walk 1.
- Walk 2.
- Flat.

Εναλλάσσοντας τα Walk 1 και 2 δίνουμε την εντύπωση ότι οι αντίπαλοι περπατάνε και όταν ο Mario πατήσει πάνω τους δείχνουμε το Flat animation για να δείξουμε στον παίκτη ότι ο εχθρός πέθανε. Παρακάτω βλέπουμε τα animations ενός αντιπάλου.



Παραπάνω ενδιαφέρον παρουσιάζουν τα animations του Mario. Αρχικά όλα τα animations του Mario χωρίζονται σε τρεις κατηγορίες:

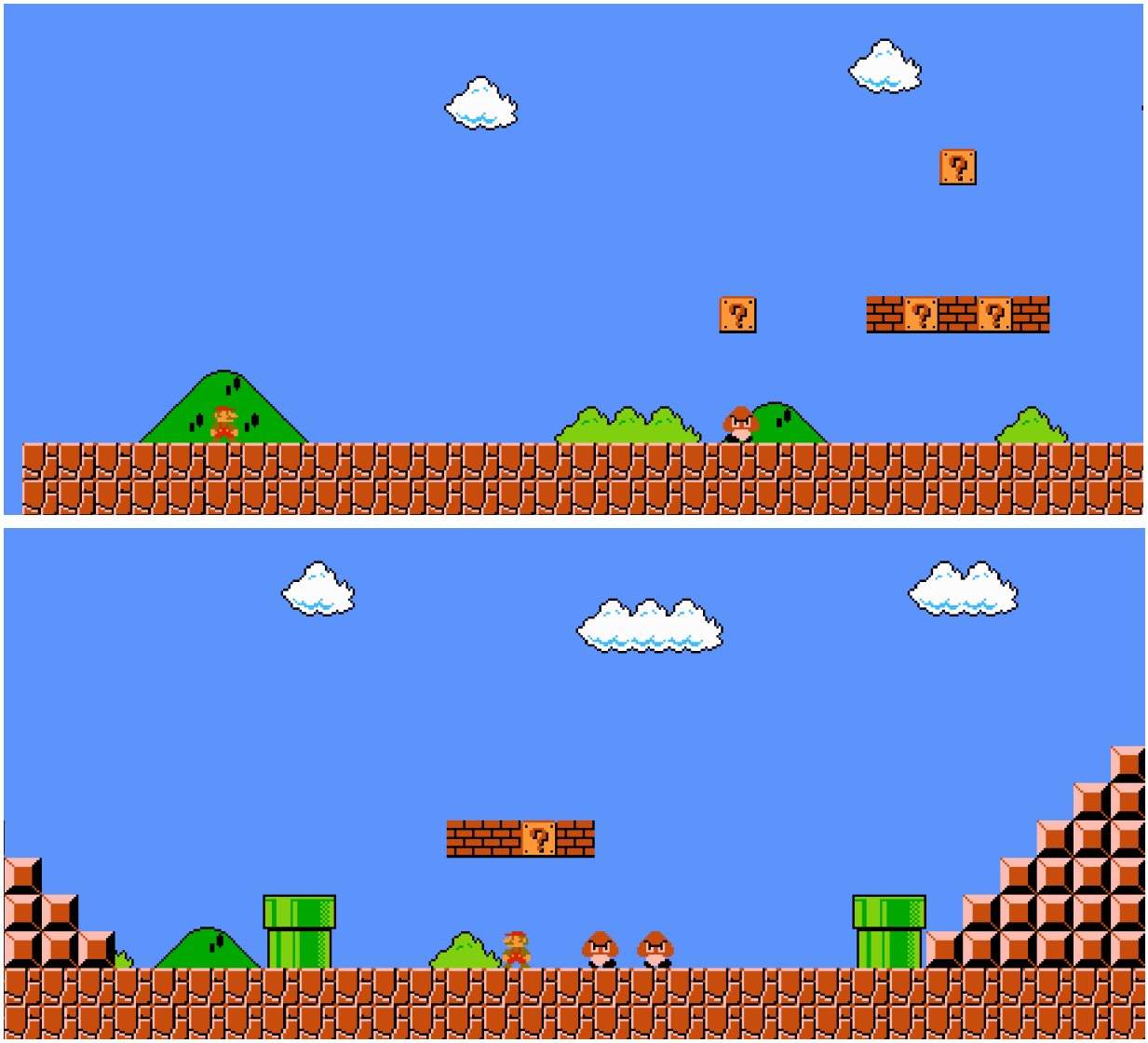
- Small.
- Normal.
- Big.

Εναλλάσσοντας μεταξύ των τριών παραπάνω κατηγοριών μεγαλώνουμε τον Mario όταν παίρνει ένα power up και τον μικραίνουμε όταν δέχεται ένα χτύπημα.
Για τον Mario έχουμε τα παρακάτω animations:

- Idle(ακινησία)
- Run 1, 2 & 3.
- Death (animation όταν χάσει όλες τις ζωές).
- Jump.
- Slide(όταν ο Mario τρέχει γρήγορα και προσπαθεί να αλλάξει κατεύθυνση).

Τέλος έχουμε και άλλα μικρά animations, όπως τα animations για τα power ups, την είσοδο του Mario σε έναν σωλήνα και την ολοκλήρωση του επιπέδου όταν ο Mario φτάνει στη σημαία.

Screenshots από την εφαρμογή



Δυσκολίες, μελλοντικές επεκτάσεις και συμπεράσματα

Σίγουρα το δυσκολότερο κομμάτι κατα την υλοποίηση ήταν ο χειρισμός του Mario και η αλληλεπίδραση του με το περιβάλλον. Συγκεκριμένα αντί να χρησιμοποιήσουμε πολλά built in κομμάτια που παρέχει το Unity, τα υλοποιήσαμε from scratch έτσι ώστε να έχουμε όσο περισσότερο έλεγχο γίνεται. Έτσι μετά από αρκετό πειραματισμό καταφέραμε να αποδώσουμε σχετικά καλά την κινητικότητα του Mario. Πιθανές μελλοντικέ επεκτάσεις θα ήταν η υλοποίηση παραπάνω επιπέδων, διαφορετικών τύπου αντιπάλων ή υλοποίηση μουσικής και sound effects.