ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΑΝΘΡΩΠΟΥ & ΥΠΟΛΟΓΙΣΤΗ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΙΚΟΝΙΚΟ ΓΥΜΝΑΣΤΗΡΙΟ

ΤΕΧΝΙΚΟ ΕΓΧΕΙΡΙΔΙΟ

ΣΤΟΙΧΕΙΑ ΜΕΛΩΝ ΟΜΑΔΑΣ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΑΘΑΝΑΣΙΟΥ ΒΑΣΙΛΕΙΟΣ ΕΥΑΓΓΕΛΟΣ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ : 19390005
ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΡΟΜΑΝΙΟΥΚ ΒΙΚΤΩΡ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ : 713242017024
ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΠΕΤΡΟΠΟΥΛΟΣ ΠΑΝΑΓΙΩΤΗΣ

ΑΛΛΗΛΕΠΙΔΡΑΣΗ ΑΝΘΡΩΠΟΥ & ΥΠΟΛΟΓΙΣΤΗ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ : 20390188
ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΘΕΟΧΑΡΗΣ ΓΕΩΡΓΙΟΣ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ : 19390283

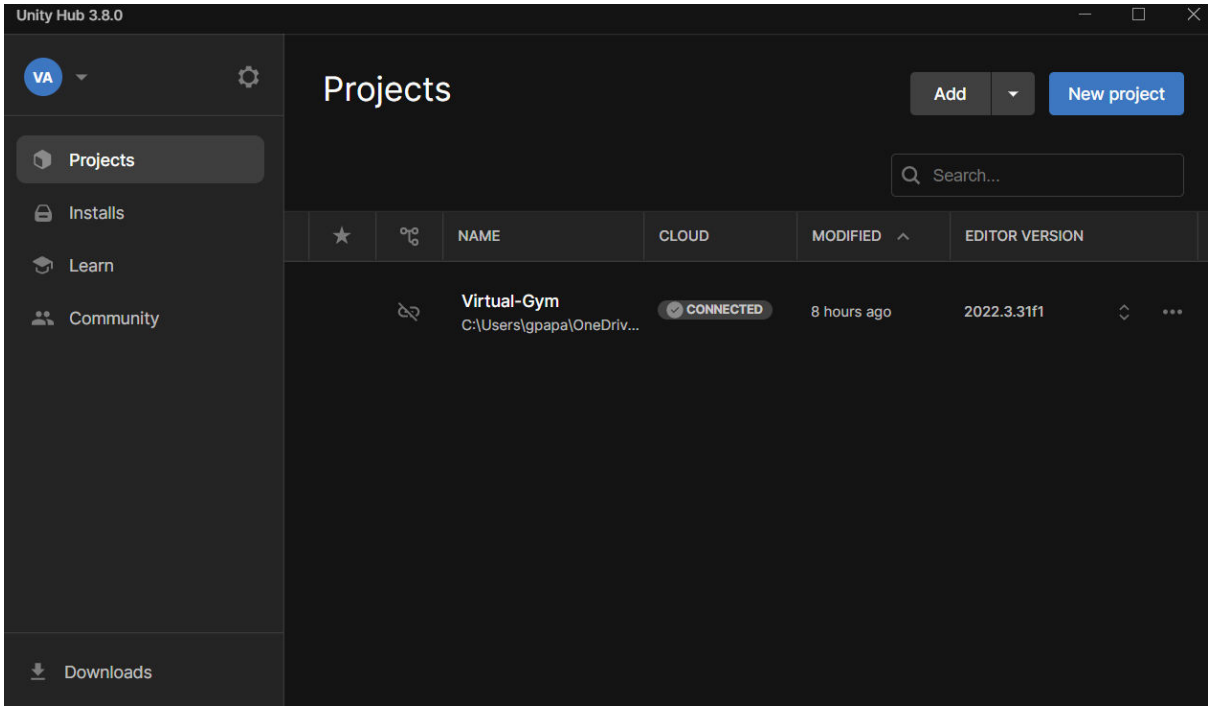
ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Το Unity είναι μία πολυδιάστατη πλατφόρμα ανάπτυξης λογισμικού που χρησιμοποιείται κυρίως για τη δημιουργία παιχνιδιών και διαδραστικού περιεχομένου σε διάφορες πλατφόρμες. Με χρήση γλωσσών προγραμματισμού όπως η C#, οι developers μπορούν να δημιουργήσουν παιχνίδια με υψηλή ποιότητα και απλότητα. Παρέχεται επίσης ένα ευέλικτο περιβάλλον ανάπτυξης, με εργαλεία που επιτρέπουν τη δημιουργία γραφικών, ήχων, φυσικών χαρακτηριστικών και άλλων στοιχείων που απαιτούνται για τα παιχνίδια.

Οδηγίες Εγκατάστασης

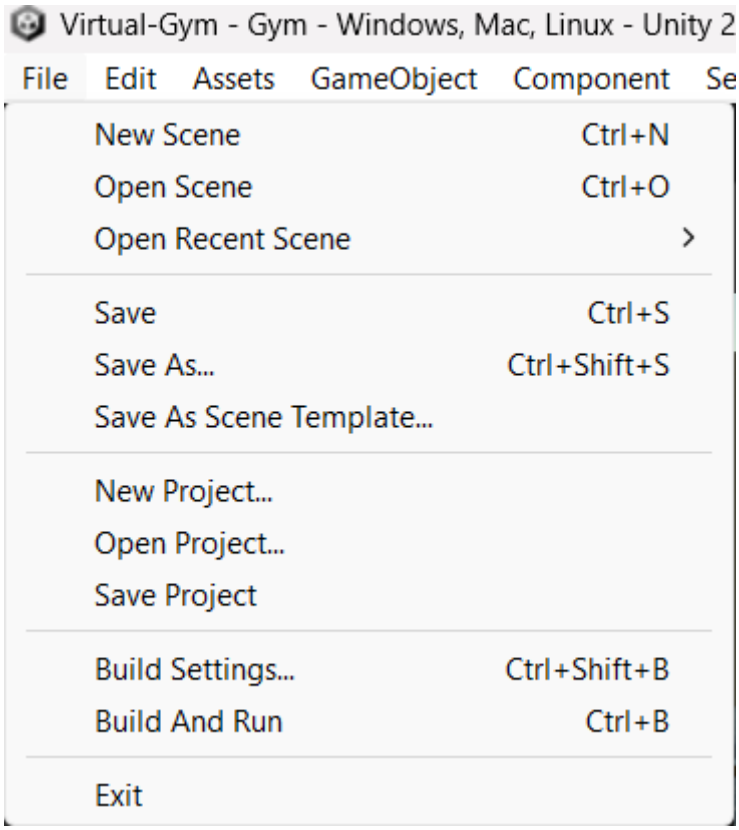
Πρώτα κάντε προσθήκη την εφαρμογή στο Unity Hub πατώντας το κουμπί Add.



Εικόνα 1. Unity Hub

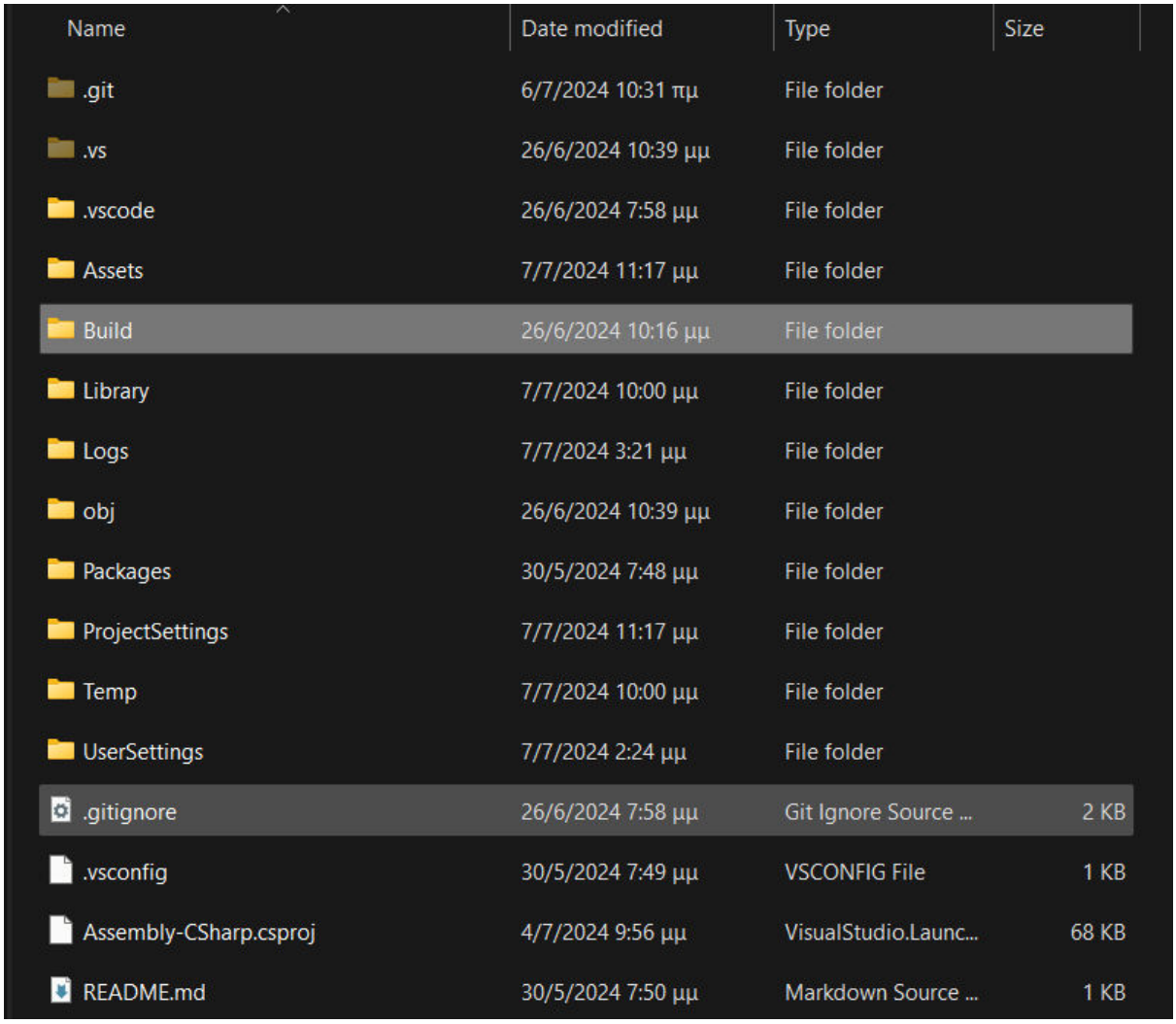
Έπειτα πατώντας Ctrl+B ξεκινάτε την διαδικασία Build της εφαρμογής για να παράγετε το εκτελέσιμο αρχείο.

Εικόνα 2. Unity



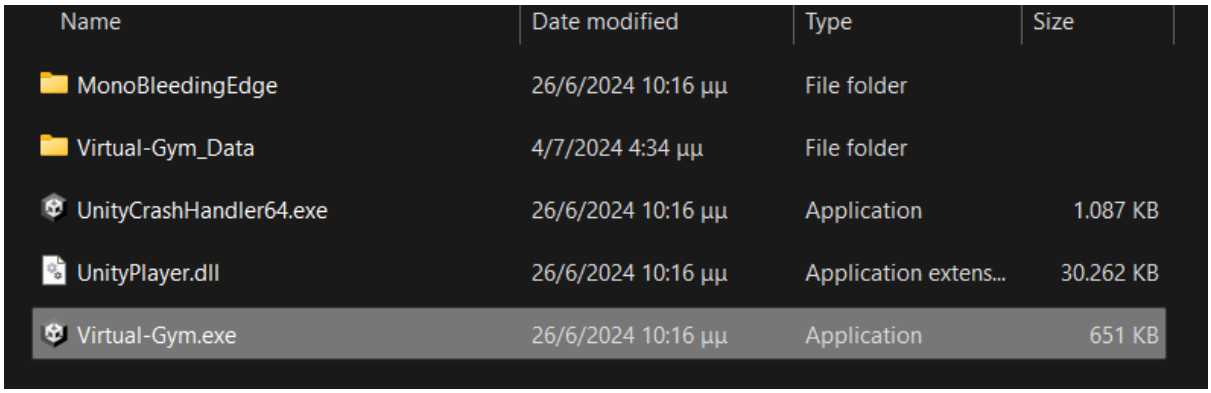
Εικόνα 3. Build and Run

Πηγαίνοντας στον file explorer και στα αρχεία του project, πηγαίνετε στον Build Folder.



Εικόνα 4. Φάκελος του project

Πατάτε το Virtual-Gym.exe και ξεκινάτε την εφαρμογή.



Εικόνα 5. Φάκελος του εκτελέσιμου

Περιγραφή

Στα πλαίσια της εργασίας αναπτύχθηκε ένα εικονικό γυμναστήριο μέσω Unity (2022.3.31f1) το οποίο δίνει την δυνατότητα στον χρήστη να αλληλεπιδρά και να εκτελέσει διάφορες λειτουργίες, όπως ασκήσεις σε όργανα γυμναστικής, επικοινωνία με γυμναστή κλπ.

Τα αρχεία κώδικα της εφαρμογής βρίσκονται επίσης στο παρακάτω αποθετήριο

https://github.com/Human-Computer-Interaction/Virtual-Gym

Αναλυτική Παρουσίαση Λειτουργιών

Οι λειτουργίες παρουσιάζονται αναλυτικά στο συνοπτικό εγχειρίδιο χρήστη (User-Manual.pdf)

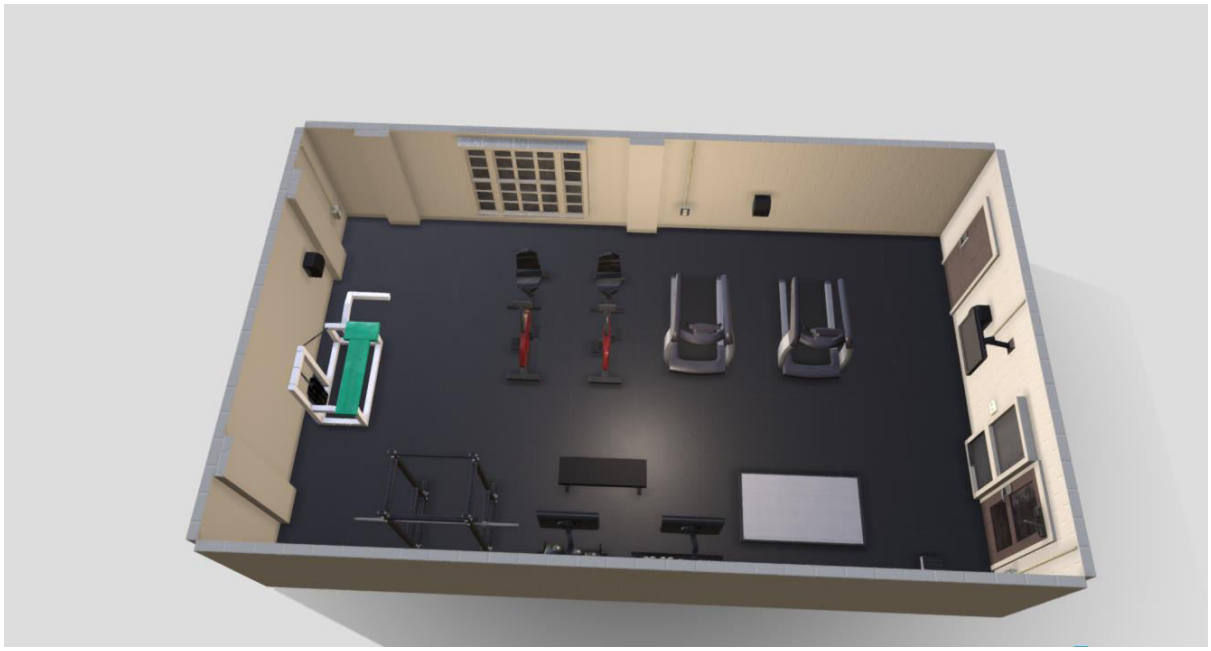
Σχεδιασμός και Υλοποίηση Γραφικών

Για την υλοποίηση των γραφικών, κατεβάσαμε τα assets από το ίντερνετ δωρεάν και τα προσθέσαμε στον φάκελο του project. Αναλυτικά έχουμε τα εξής assets:

Χώρος Γυμναστηρίου

Ο χώρος του γυμναστηρίου αντλήθηκε από την ιστοσελίδα “sketchfab” η οποία περιέχει πληθώρα μοντέλων. Το μοντέλο του γυμναστηρίου υπήρχε έτοιμο , ωστόσο κάναμε τις κατάλληλες παραμετροποιήσεις έτσι ώστε να εξυπηρετεί το σκοπό μας. Για παράδειγμα, στο αρχικό μοντέλο τα components του γυμναστηρίου ήταν πολύ κοντά μεταξύ τους με αποτέλεσμα να προκαλούν πολλαπλά collisions . Καταφέραμε να το παραμετροποιήσουμε επειδή κάθε αντικείμενο στο χώρο είναι “ξεχωριστό” και όχι ενιαίο.

Πριν:



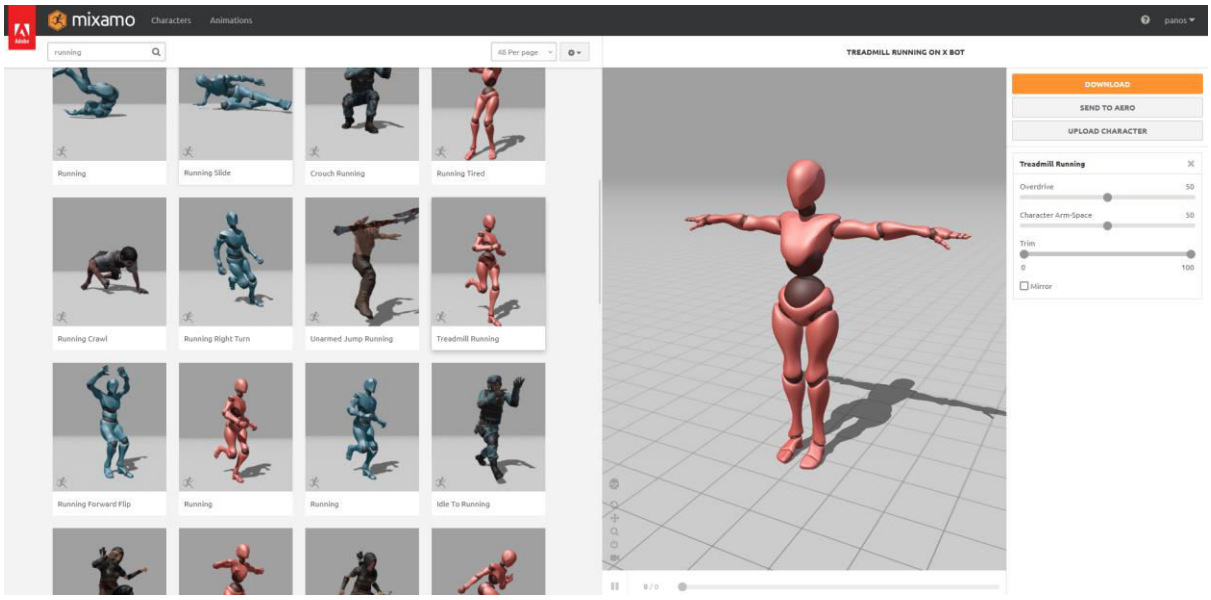
Μετά:

Πηγή:

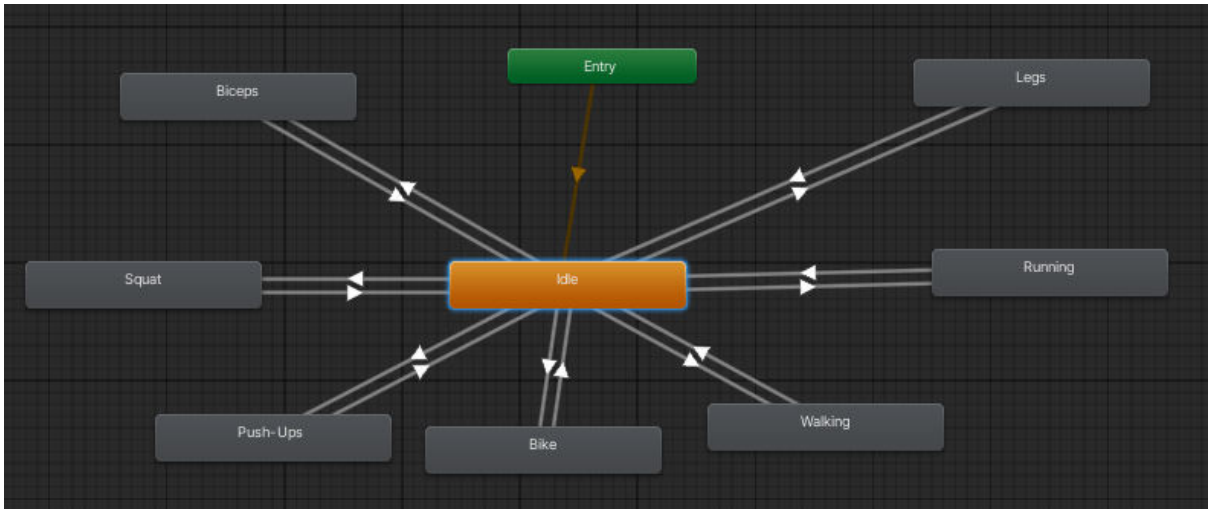
https://sketchfab.com/3d-models/modular-gym-86e8cc43d76c44e3925e625fdab155f3

Αθλητής

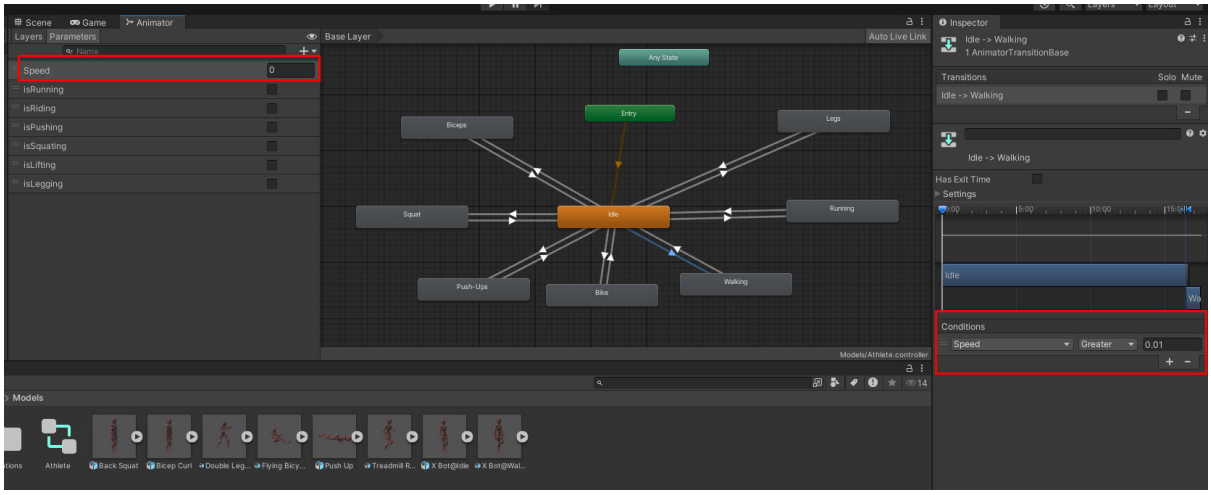
Τόσο το μοντέλο του αθλητή όσο και τα animations των καταστάσεων idle,walking και των ασκήσεων γυμναστικής αντλήθηκαν από τον ιστότοπο “Mixamo”.



Την υλοποίηση των animation στις διαφορετικές καταστάσεις που μπορεί να βρεθεί ο παίκτης τις υλοποιήσαμε μέσω του Unity και συγκεκριμένα μέσω του Unity animator.



Κάθε κόμβος του animation αποτελεί μια διαφορετική κατάσταση. Η μετάβαση από έναν κόμβο στον άλλο γίνεται με την αλλαγή της μεταβλητής “κατάστασης”. Για παράδειγμα όταν ο χρήστης δεν κινείται βρίσκεται μόνιμα στην κατάσταση idle. Ένας παράγοντας μετάβασης είναι όταν ο χρήστης πατάει το πλήκτρο w άρα μετακινεί τον παίκτη , δηλαδή αυξάνει την ταχύτητά του. Επομένως όταν ο αθλητής έχει ταχύτητα < 0.1 άρα δεν κινείται είναι στην κατάσταση animation idle ενώ όταν πατάει το πλήκτρο w μεταβαίνει στην κατάσταση walking :



Το ίδιο ακριβώς συμβαίνει και με τα υπόλοιπα animation μόνο που σε αυτά έχουμε χρησιμοποιήσει boolean μεταβλητή για το αν βρίσκονται ή όχι στην σωστή θέση-στιγμή-κατάσταση για να πρέπει να ενεργοποιηθούν. Για παράδειγμα οταν ο παίκτης κάνει collide με τον διάδρομο τότε η global μεταβλητή του animation is Walking αλλάζει από false σε true άρα έχουμε και μετάβαση στην κατάσταση walking. Όταν κάνει exit το collision τότε η μεταβλητή γίνεται false.

Ο παρακάτω κώδικας για τον αθλητή αλλάζει την τιμή του speed αναλόγως αν περπατάει η οχι (δηλαδή πατάει w)

```
void Update()

    {



        isGrounded = Physics.CheckSphere(groundCheck.position,
groundDistance, groundMask);



        if (isGrounded && velocity.y < 0)

        {

            velocity.y = -2f;

        }

        float x = Input.GetAxis("Horizontal");

        float z = Input.GetAxis("Vertical");



        Vector3 move = new Vector3(x, 0, z);
```

```
        if (move == Vector3.zero)

        {

            animator.SetFloat("Speed", 0);

        }

        else

        {

            animator.SetFloat("Speed", 1f);

        }



        move = transform.right * x + transform.forward * z;



        controller.Move(speed * Time.deltaTime * move);



    }
```

Ενώ στον παρακάτω ελέγχει τα collisions για να ορίσει ποιο animation πρέπει να τρέξει αρα και σε ποιά κατάσταση πρέπει να μεταβεί :

```
private void OnTriggerEnter(Collider other)

    {

        if (other.gameObject.CompareTag("Treadmills") |
other.gameObject.CompareTag("Bike") |
other.gameObject.CompareTag("Push-Ups")

            | other.gameObject.CompareTag("Bar") |
other.gameObject.CompareTag("Weights") |
other.gameObject.CompareTag("Leg Extension"))

        {


playerCameraRotation.GetComponentInChildren<MouseLook>().enabled =
false;
```

```
            camera.transform.position = new
Vector3(camera.transform.position.x, camera.transform.position.y + 1f,
camera.transform.position.z);

            camera.transform.rotation = Quaternion.Euler(90f, 0f, 0f);

            if (other.gameObject.CompareTag("Treadmills"))

            {

                animator.SetBool("isRunning", true);



            }

            else if (other.gameObject.CompareTag("Bike"))

            {

                animator.SetBool("isRiding", true);

            }

            else if (other.gameObject.CompareTag("Push-Ups"))

            {

                animator.SetBool("isPushing", true);

                print(animator.GetBool("isPushing"));

            }

            else if (other.gameObject.CompareTag("Bar"))

            {

                animator.SetBool("isSquating", true);

            }

            else if (other.gameObject.CompareTag("Weights"))

            {

                animator.SetBool("isLifting", true);

            }

            else if (other.gameObject.CompareTag("Leg Extension"))

            {
```

```
                animator.SetBool("isLegging", true);

            }

        }

    }



    private void OnTriggerExit(Collider other)

    {

        if (other.gameObject.CompareTag("Treadmills") |
other.gameObject.CompareTag("Bike") |
other.gameObject.CompareTag("Push-Ups")

        | other.gameObject.CompareTag("Bar") |
other.gameObject.CompareTag("Weights") |
other.gameObject.CompareTag("Leg Extension"))

        {


playerCameraRotation.GetComponentInChildren<MouseLook>().enabled =
true;

            camera.transform.position = new
Vector3(camera.transform.position.x, camera.transform.position.y - 1f,
camera.transform.position.z);

            camera.transform.rotation = Quaternion.Euler(-90f, 0f, 0f);

            if (other.gameObject.CompareTag("Treadmills"))

            {

                animator.SetBool("isRunning", false);



            }

            else if (other.gameObject.CompareTag("Bike"))

            {

                animator.SetBool("isRiding", false);

            }
```

```
            else if (other.gameObject.CompareTag("Push-Ups"))
            {
                animator.SetBool("isPushing", false);
            }
            else if (other.gameObject.CompareTag("Bar"))
            {
                animator.SetBool("isSquating", false);
            }
            else if (other.gameObject.CompareTag("Weights"))
            {
                animator.SetBool("isLifting", false);
            }
            else if (other.gameObject.CompareTag("Leg Extension"))
            {
                animator.SetBool("isLegging", false);
            }
        }

    }
```

Γυμναστής

Ισχύει ότι ισχύει και για τον αθλητή μόνο που έχει αλλαχτεί το χρώμα σε μπλέ. Επίσης βρίσκεται μονίμως στην κατασταση idle

Διατροφολόγος

Ισχύει ότι ισχύει και για τον αθλητή μόνο που έχει αλλαχτεί το χρώμα σε πράσινο. Επίσης βρίσκεται μονίμως στην κατασταση idle

Αυτόματος Πωλητής

Αντλήθηκε από το unity assets store .

Πηγή:

https://assetstore.unity.com/packages/3d/props/interior/snack-machines-3517

Σκηνή Αρχικού Μενού

Η πρώτη σκηνή παρουσιάζει το αρχικό μενού με 2 κουμπιά “Play” και “Quit”. Ουσιαστικά, ο χρήστης έχει την δυνατότητα να επιλέξει να παίξει το παιχνίδι ή να το κλείσει. Το background του μενού περιλαμβάνει μία εικόνα γυμναστηρίου, ενώ τα κουμπιά έχουν δικό τους σχεδιαστικό. Πρόκειται δηλαδή, για ένα αντικείμενο Canvas.

Όσον αφορά τις λειτουργίες των κουμπιών “Play” και “Quit”, ενεργοποιούνται με τη μέθοδο onClick() όπου καλούνται οι συναρτήσεις StartGame() σε περίπτωση που ο χρήστης πατήσει το κουμπί “Play” και StopGame() σε περίπτωση που ο χρήστης πατήσει το κουμπί “Quit”.

Οι συναρτήσεις StartGame() και StopGame() αποτελούν μέθοδοι της κλάσης StartStopGame που βρίσκεται στο παρακάτω πηγαίο αρχείο

[1] StartStop.cs

```
using UnityEngine;

using UnityEngine.SceneManagement;

public class StartStopGame : MonoBehaviour

{

    public void StartGame()

    {

        SceneManager.LoadScene("Gym");

        Time.timeScale = 1;

    }

    public void StopGame()

    {

        Application.Quit();

    }

}
```

Σκηνή Γυμναστηρίου

Αθλητής (Athlete)

Ο αθλητής είναι ο χρήστης που αλληλεπιδρά με το εικονικό γυμναστήριο. Ο έλεγχος του υλοποιείται με δύο scripts που χειρίζονται την κάμερα του παιχνιδιού και την κίνηση του παίκτη αντίστοιχα.

Κάμερα

Η κάμερα του παιχνιδιού είναι τοποθετημένη στο κεφάλι του χρήστη που σημαίνει ότι η κίνηση του είναι First-Person. Με άλλα λόγια η όψη του χρήστη στο παιχνίδι είναι από τα ίδια του τα μάτια, χωρίς να φαίνεται το μοντέλο του στην όψη (Third-Person). Η κάμερα του παίκτη ελέγχεται από το script MouseLook.cs

[2] MouseLook.cs

```
using UnityEngine;


public class MouseLook : MonoBehaviour

{

    public float mouseSensitivity = 100f;


    public Transform playerBody;


    private float xRotation = 0f;


    void Start()

    {

        Cursor.lockState = CursorLockMode.Locked;

        playerBody.rotation = Quaternion.Euler(0, 90f, 0);

    }


    void Update()
```

```
    {

        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity *
Time.deltaTime;

        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity *
Time.deltaTime;



        xRotation -= mouseY;

        xRotation = Mathf.Clamp(xRotation, -90f, 90f);



        transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);

        playerBody.Rotate(Vector3.up * mouseX);

    }

}
```

Ορίζουμε το mouse sensitivity στο 100, όπου καθορίζει το πόσο γρήγορες θα είναι οι κινήσεις του ποντικιού. Επίσης, ορίζουμε και τις κινήσεις του ποντικιού τόσο στον οριζόντιο άξονα X όσο και στον κάθετο άξονα Y. Όσον αφορά, την κίνηση του ποντικιού στον άξονα Z, ορίζεται από την συνάρτηση Quaternion.Euler() η οποία χειρίζεται την περιστροφή του παίκτη και στους 3 άξονες.

Κίνηση

Το script AthleteMovement.cs χειρίζεται την κίνηση και τα κινούμενα σχέδια του χρήστη-αθλητή. Χρησιμοποιεί έναν CharacterController για την κίνηση, επιτρέποντας στον χαρακτήρα να κινείται με βάση την είσοδο του παίκτη, και ελέγχει αν στον χαρακτήρα ασκούνται τα φαινόμενα βαρύτητας. Το script διαχειρίζεται διάφορα animations (τρέξιμο,ώθηση κ.λπ.) που ενεργοποιούνται από αλληλεπιδράσεις με διάφορα όργανα άσκησης χρησιμοποιώντας τις μεθόδους OnTriggerEnter και OnTriggerExit. Η θέση και η περιστροφή της κάμερας προσαρμόζονται όταν ο χαρακτήρας αλληλεπιδρά με αυτά τα αντικείμενα για να παρέχει την κατάλληλη οπτική ανατροφοδότηση. Η μέθοδος Update ελέγχει διαρκώς την είσοδο του παίκτη για την κίνηση του χαρακτήρα και ενημερώνει αναλόγως την κατάσταση των κινούμενων σχεδίων, εξασφαλίζοντας μια ομαλή και ευέλικτη εμπειρία παιχνιδιού.

[3] AthleteMovement.cs

```
using System;

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.EventSystems;


public class AthleteMovement : MonoBehaviour




{

    [SerializeField]

    private CharacterController controller;


    [SerializeField]

    private float speed = 1f;


    [SerializeField]

    private Transform groundCheck;

    [SerializeField]

    private float groundDistance = 0.0f;

    private LayerMask groundMask;


    private Vector3 velocity;

    private bool isGrounded;


    private Animator animator;
```

```
    [SerializeField] new Camera camera;


    [SerializeField] GameObject Treadmill1;

    [SerializeField] GameObject Treadmill2;


    private GameObject playerCameraRotation;


    private void OnTriggerEnter(Collider other)

    {

        if (other.gameObject.CompareTag("Treadmills") |
other.gameObject.CompareTag("Bike") | other.gameObject.CompareTag("Push-Ups")

            | other.gameObject.CompareTag("Bar") |
other.gameObject.CompareTag("Weights") | other.gameObject.CompareTag("Leg
Extension"))

        {

            playerCameraRotation.GetComponentInChildren<MouseLook>().enabled =
false;

            camera.transform.position = new Vector3(camera.transform.position.x,
camera.transform.position.y + 1f, camera.transform.position.z);

            camera.transform.rotation = Quaternion.Euler(90f, 0f, 0f);

            if (other.gameObject.CompareTag("Treadmills"))

            {

                animator.SetBool("isRunning", true);



            }

            else if (other.gameObject.CompareTag("Bike"))

            {

                animator.SetBool("isRiding", true);

            }

            else if (other.gameObject.CompareTag("Push-Ups"))

            {
```

```
                animator.SetBool("isPushing", true);

                print(animator.GetBool("isPushing"));

            }

            else if (other.gameObject.CompareTag("Bar"))

            {

                animator.SetBool("isSquating", true);

            }

            else if (other.gameObject.CompareTag("Weights"))

            {

                animator.SetBool("isLifting", true);

            }

            else if (other.gameObject.CompareTag("Leg Extension"))

            {

                animator.SetBool("isLegging", true);

            }

        }

    }


    private void OnTriggerExit(Collider other)

    {

        if (other.gameObject.CompareTag("Treadmills") |
other.gameObject.CompareTag("Bike") | other.gameObject.CompareTag("Push-Ups")

        | other.gameObject.CompareTag("Bar") |
other.gameObject.CompareTag("Weights") | other.gameObject.CompareTag("Leg
Extension"))

        {

            playerCameraRotation.GetComponentInChildren<MouseLook>().enabled =
true;

            camera.transform.position = new Vector3(camera.transform.position.x,
camera.transform.position.y - 1f, camera.transform.position.z);

            camera.transform.rotation = Quaternion.Euler(-90f, 0f, 0f);

            if (other.gameObject.CompareTag("Treadmills"))
```

```
            {
                animator.SetBool("isRunning", false);


            }
            else if (other.gameObject.CompareTag("Bike"))
            {
                animator.SetBool("isRiding", false);
            }
            else if (other.gameObject.CompareTag("Push-Ups"))
            {
                animator.SetBool("isPushing", false);
            }
            else if (other.gameObject.CompareTag("Bar"))
            {
                animator.SetBool("isSquating", false);
            }
            else if (other.gameObject.CompareTag("Weights"))
            {
                animator.SetBool("isLifting", false);
            }
            else if (other.gameObject.CompareTag("Leg Extension"))
            {
                animator.SetBool("isLegging", false);
            }
        }

    }



    void Start()
```

```
    {


        animator = GetComponentInChildren<Animator>();

        groundCheck = GameObject.Find("Ground-Check").GetComponent<Transform>();

        controller = GetComponent<CharacterController>();

        playerCameraRotation = GameObject.Find("Athlete");


    }


    void Update()

    {


        isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance,
groundMask);


        if (isGrounded && velocity.y < 0)

        {

            velocity.y = -2f;

        }

        float x = Input.GetAxis("Horizontal");

        float z = Input.GetAxis("Vertical");


        Vector3 move = new Vector3(x, 0, z);

        if (move == Vector3.zero)

        {

            animator.SetFloat("Speed", 0);

        }

        else

        {

            animator.SetFloat("Speed", 1f);
```

```
        }


        move = transform.right * x + transform.forward * z;


        controller.Move(speed * Time.deltaTime * move);


    }


}
```

Γυμναστής (Fitness Instructor)

Η επικοινωνία του χρήστη με τον γυμναστή υλοποιείται στο script NPCInteraction.cs. Όταν ο παίκτης βρίσκεται κοντά στον NPC (γυμναστή) και πατάει το πλήκτρο "E", εμφανίζεται ένα πάνελ διαλόγου που εμφανίζει μηνύματα από τον NPC. Το σενάριο διαχειρίζεται διαφορετικές ακολουθίες διαλόγου ανάλογα με την πρόοδο του παίκτη, οι οποίες ελέγχονται από το κύριο script GameManager. Διευκολύνει επίσης την εισαγωγή των χαρακτηριστικών του παίκτη, τα οποία χρησιμοποιούνται για τη δημιουργία εξατομικευμένων προγραμμάτων γυμναστικής με βάση τον ΔΜΣ (Δείκτης Μάζας Σώματος) του παίκτη. Ο NPC παρέχει συγκεκριμένες ρουτίνες άσκησης ανάλογα με τη φυσική κατάσταση του παίκτη (λιποβαρής, φυσιολογικός, υπέρβαρος, παχύσαρκος), καθοδηγώντας τον παίκτη σε διάφορες δραστηριότητες στο γυμναστήριο.

[4] NPCInteraction,cs

```
using System;

using System.Collections;

using System.Collections.Generic;

using TMPro;

using UnityEngine;


public class NPCInteraction : MonoBehaviour

{
```

```
    public GameObject dialoguePanel;
    public TextMeshProUGUI dialogueText;
    public GameObject continueButton;
    public GameObject characteristicsInputPanel;
    public GameObject gameManagerComponent;
    private GameManager gameManager;
    private GameObject playerCameraRotation;
    private GameObject playerCamera;


    private List<string> dialogue;
    private int index;
    public float wordSpeed;
    public bool playerIsClose;
    private bool hasCharacteristics = false;
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Athlete"))
        {
            playerIsClose = true;
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.CompareTag("Athlete"))
        {
            playerIsClose = false;
            ZeroText();
        }
    }
    void Start()
```

```
    {
        playerCameraRotation = GameObject.Find("Athlete");
        playerCamera = GameObject.Find("Camera");
        gameManager = gameManagerComponent.GetComponent<GameManager>();


        dialogue = new List<string>
        {
            "Welcome to PADA Gym!",
            "Next, you're going to sign up your characteristics form.",
        };
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E) && playerIsClose)
        {
            playerCameraRotation.GetComponentInChildren<MouseLook>().enabled =
false;
            Cursor.lockState = CursorLockMode.Confined;
            playerCamera.transform.rotation = Quaternion.Euler(new Vector3(12f,
88f, 0f));
            if (dialoguePanel.activeInHierarchy)
            {
                ZeroText();
            }
            else
            {
                dialoguePanel.SetActive(true);
                StartCoroutine(Typing());
            }
        }
        if (dialogueText.text == dialogue[index])
```

```
        {
            continueButton.SetActive(true);
        }
        if (gameManager.TasksAreFinished())
        {
            dialogue = new List<string>
                {
                    "You have finished your exercise. Good Job!",
                    "Now you have to eat some food. Ask the nutritionist for more
info about the food.",
                    "Good luck!",
                };
        }
        if (gameManager.athleteBoughtAllItems)
        {
            dialogue = new List<string>
                {
                    "Seems like you have also finished your nutrition program.
Good Job!",
                    "Thanks for using PADA Gym."
                };
        }
    }
    public void ZeroText()
    {
        dialogueText.text = "";
        index = 0;
        dialoguePanel.SetActive(false);
        playerCameraRotation.GetComponentInChildren<MouseLook>().enabled = true;
        Cursor.lockState = CursorLockMode.Locked;
    }
```

```
    public void NextLine()

    {

        continueButton.SetActive(false);

        if (index < dialogue.Count - 1)

        {

            index++;

            dialogueText.text = "";

            StartCoroutine(Typing());

        }

        else if (index == dialogue.Count - 1)

        {

            dialoguePanel.SetActive(false);

            if (!hasCharacteristics)

                characteristicsInputPanel.SetActive(true);

            playerIsClose = false;

        }

    }

    public void PassInputsToGameManager()

    {

        var inputsFields =
characteristicsInputPanel.GetComponentsInChildren<TMP_InputField>();

        gameManager.playerStats.Weight =
Math.Abs(float.Parse(inputsFields[0].text));

        gameManager.playerStats.Age = Math.Abs(int.Parse(inputsFields[1].text));

        gameManager.playerStats.Height =
Math.Abs(float.Parse(inputsFields[2].text)) / 100f;

        gameManager.InitStats();

        gameManager.ActivatePanel();

        characteristicsInputPanel.SetActive(false);

        inputsFields[0].text = "";

        inputsFields[1].text = "";
```

```
        inputsFields[2].text = "";

        hasCharacteristics = true;

        GymPlanBasedOnBMI();

        gameManager.hasSpokenToNpc = true;

    }


    public void GymPlanBasedOnBMI()

    {

        dialogue.Clear();

        gameManager.CalculateBMI(gameManager.playerStats);

        string BodyType = gameManager.BodyTypeBasedOnBmi();

        switch (BodyType)

        {

            case "Underweight":

                {

                    gameManager.physicalCondition = "Underweight";

                    dialogue.Add("You are 'Underweight' so I created a gym program
for you.");

                    dialogue.Add("This program will help you gain muscle, but you
will have to eat more.");

                    dialogue.Add("Ask the nutritionist for more information about
your food program.");

                    dialogue.Add("Your personalized plan is ready.");

                    dialogue.Add("10' of Treadmill, 20' Bar, 10' of Squats, 10' of
Leg Extensions, 10' of Dumbells.");

                    dialogue.Add("Good luck!");

                    gameManager.equipmentUse.TreadmillUse = 10f;

                    gameManager.equipmentUse.BarUse = 20f;

                    gameManager.equipmentUse.LegExtensionUse = 10f;

                    gameManager.equipmentUse.DumbellsUse = 10f;

                    gameManager.equipmentUse.BikeUse = 0f;
```

```
                    gameManager.equipmentUse.MatUse = 0f;

                    gameManager.tasksToFinish.Add("Treadmill", 10f);

                    gameManager.tasksToFinish.Add("Bar", 20f);

                    gameManager.tasksToFinish.Add("LegExtension", 10f);

                    gameManager.tasksToFinish.Add("Dumbells", 10f);

                    break;

                }

            case "Normal":

                {

                    gameManager.physicalCondition = "Normal";

                    dialogue.Add("Your physical characteristics are 'Normal'.");

                    dialogue.Add("This program will help you maintain your
weight.");

                    dialogue.Add("Ask the nutritionist for more information about
your food program.");

                    dialogue.Add("Your personalized plan is ready.");

                    dialogue.Add("10' of Treadmill, 10' Bicycle, 20' Bar, 10' of
Squats, 10' of Leg Extensions.");

                    dialogue.Add("Good luck!");

                    gameManager.equipmentUse.TreadmillUse = 10f;

                    gameManager.equipmentUse.BikeUse = 10f;

                    gameManager.equipmentUse.BarUse = 20f;

                    gameManager.equipmentUse.LegExtensionUse = 10f;

                    gameManager.equipmentUse.DumbellsUse = 0f;

                    gameManager.equipmentUse.MatUse = 0f;

                    gameManager.tasksToFinish.Add("Treadmill", 10f);

                    gameManager.tasksToFinish.Add("Bike", 10f);

                    gameManager.tasksToFinish.Add("Bar", 20f);

                    gameManager.tasksToFinish.Add("LegExtension", 10f);

                    break;

                }
```

```
            case "Overweight":
                {
                    gameManager.physicalCondition = "Overweight";
                    dialogue.Add("You are 'Overweight' so I created a gym program
for you.");
                    dialogue.Add("This program will help you to lose weight, but
you will have to eat less.");
                    dialogue.Add("Ask the nutritionist for more information about
your food program.");
                    dialogue.Add("Your personalized plan is ready.");
                    dialogue.Add("20' of Treadmill, 20' Bycicle, 20' Bar, 10' of
Squats, 10' of Leg Extensions.");
                    dialogue.Add("Good luck!");
                    gameManager.equipmentUse.TreadmillUse = 20f;
                    gameManager.equipmentUse.BikeUse = 20f;
                    gameManager.equipmentUse.BarUse = 20f;
                    gameManager.equipmentUse.LegExtensionUse = 10f;
                    gameManager.equipmentUse.DumbellsUse = 0f;
                    gameManager.equipmentUse.MatUse = 0f;
                    gameManager.tasksToFinish.Add("Treadmill", 20f);
                    gameManager.tasksToFinish.Add("Bike", 20f);
                    gameManager.tasksToFinish.Add("Bar", 20f);
                    gameManager.tasksToFinish.Add("LegExtension", 10f);
                    break;
                }
            case "Obese":
                {
                    gameManager.physicalCondition = "Obese";
                    dialogue.Add("You are 'Obese' so I created a gym program for
you.");
                    dialogue.Add("This program will help you lose weight, but you
will have to eat less.");
```

```
                    dialogue.Add("Ask the nutritionist for more information about
your food program.");

                    dialogue.Add("Your personalized plan is ready.");

                    dialogue.Add("20' of Treadmill, 20' Bycicle, 20' Bar, 10' of
Push Ups, 10' of Leg Extensions, 10 Dumbells.");

                    dialogue.Add("Good luck");

                    gameManager.equipmentUse.TreadmillUse = 20f;

                    gameManager.equipmentUse.BikeUse = 20f;

                    gameManager.equipmentUse.BarUse = 20f;

                    gameManager.equipmentUse.LegExtensionUse = 10f;

                    gameManager.equipmentUse.DumbellsUse = 10f;

                    gameManager.equipmentUse.MatUse = 10f;

                    gameManager.tasksToFinish.Add("Treadmill", 20f);

                    gameManager.tasksToFinish.Add("Bike", 20f);

                    gameManager.tasksToFinish.Add("Bar", 20f);

                    gameManager.tasksToFinish.Add("LegExtension", 10f);

                    gameManager.tasksToFinish.Add("Dumbells", 10f);

                    gameManager.tasksToFinish.Add("Mat", 10f);


                    break;

                }



        }

        dialogueText.text = dialogue[0];

        dialoguePanel.SetActive(true);

    }


    IEnumerator Typing()

    {

        foreach (char letter in dialogue[index].ToCharArray())
```

```
        {
            dialogueText.text += letter;
            yield return new WaitForSeconds(wordSpeed);
        }
    }
}
```

Διατροφολόγος (Nutritionist)

Η επικοινωνία του χρήστη με τον διατροφολόγο υλοποιείται στο script NutritionistInteraction.cs. Η φιλοσοφία είναι ίδια με το script NPCInteraction.cs που διαχειρίζεται την επικοινωνία του χρήστη με τον γυμναστή. Η διαφορά είναι στους διαλόγους, όπου ο διατροφολόγος βγάζει προγράμματα διατροφής ανάλογα την φυσική κατάσταση του χρήστη. Η φυσική κατάσταση αποθήκευεται στην μεταβλητή "physical_condition" του κυρίου αντικειμένου ελέγχου GameManager, όπου παίρνει τιμή μόνο εφόσον ο χρήστης επικοινωνήσει με τον γυμναστή για να εισάγει τα στοιχεία του (μάζα, ηλικία, ύψος).

[5] NutritionistInteraction.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;


public class NutrionistInteraction : MonoBehaviour
{
    public GameObject dialoguePanel;
    public TextMeshProUGUI dialogueText;
    public GameObject continueButton;
    public GameObject gameManagerComponent;
    private GameManager gameManager;
    private GameObject playerCameraRotation;
    private GameObject playerCamera;
```

```
public GameObject getProgramButton;
private List<string> dialogue;
private int index;
public float wordSpeed;
public bool playerIsClose;


private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Athlete"))
    {
        playerIsClose = true;
    }
}


private void OnTriggerExit(Collider other)
{
    if (other.gameObject.CompareTag("Athlete"))
    {
        playerIsClose = false;
        ZeroText();
    }
}


void Start()
{
    playerCameraRotation = GameObject.Find("Athlete");
    playerCamera = GameObject.Find("Camera");
    gameManager = gameManagerComponent.GetComponent<GameManager>();
```

```
        if (gameManager == null)
        {
            return;
        }
        InitializeDialogue();
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E) && playerIsClose)
        {
            HandleInteraction();
        }
        if (dialogueText.text == dialogue[index])
        {
            continueButton.SetActive(true);
        }
        if (gameManager.athleteBoughtAllItems)
        {
            dialogue = new List<string>
                {
                    "You have finished your nutrition program. Good Job!",
                    "Thanks for using PADA Gym."
                };
        }
    }

    private void HandleInteraction()
    {
        playerCameraRotation.GetComponentInChildren<MouseLook>().enabled = false;
```

```
        Cursor.lockState = CursorLockMode.Confined;

        playerCamera.transform.rotation = Quaternion.Euler(new Vector3(12f, 88f,
0f));



        if (dialoguePanel.activeInHierarchy)

        {

            ZeroText();

        }

        else

        {

            dialoguePanel.SetActive(true);

            StartCoroutine(Typing());

        }

    }



    private void InitializeDialogue()

    {

        if (gameManager.physicalCondition == "Unknown")

        {

            dialogue = new List<string>

            {

                "Welcome to PADA Gym!",

                "You should talk to the fitness instructor to check your physical
condition."

            };

        }

    }



    public void ZeroText()

    {

        dialogueText.text = "";
```

```
        index = 0;
        dialoguePanel.SetActive(false);
        playerCameraRotation.GetComponentInChildren<MouseLook>().enabled = true;
        Cursor.lockState = CursorLockMode.Locked;
    }

    public void NextLine()
    {
        continueButton.SetActive(false);
        if (gameManager.hasSpokenToNpc)
        {
            PassInputsToGameManager();
        }
        if (index < dialogue.Count - 1)
        {
            index++;
            dialogueText.text = "";
            StartCoroutine(Typing());
        }
        else if (index == dialogue.Count - 1)
        {
            dialoguePanel.SetActive(false);
            playerIsClose = false;
        }
    }

    public void PassInputsToGameManager()
    {
        gameManager.ActivatePanel();
        FoodProgramBasedOnPhysicalCondition();
```

```
    }


    public void FoodProgramBasedOnPhysicalCondition()
    {
        dialogue.Clear();


        switch (gameManager.physicalCondition)
        {
            case "Underweight":
                dialogue.Add("You are 'Underweight' so I created a food program
for you.");
                dialogue.Add("The fitness instructor gave you a program that will
help you gain muscle, but you will have to eat more.");
                dialogue.Add("Let me show you what to buy from the vending
machine.");
                dialogue.Add("Your food program is ready.");
                dialogue.Add("1 protein, 1 creatine, 1 dipping belt");
                dialogue.Add("Goal: Significant weight gain and stamina
improvement.");
                dialogue.Add("It will cost you 75$.");
                gameManager.itemsToBuyFromVendingMachine["Protein"] = 1;
                gameManager.itemsToBuyFromVendingMachine["Creatine"] = 1;
                gameManager.itemsToBuyFromVendingMachine["DippingBelt"] = 1;
                break;


            case "Normal":
                dialogue.Add("You are 'Normal' so I created a food program for
you.");
                dialogue.Add("The fitness instructor gave you a program that will
help you maintain your weight.");
                dialogue.Add("Let me show you what to buy from the vending
machine.");
                dialogue.Add("Your food program is ready.");
```

```
                dialogue.Add("1 energy drink, 1 protein, 1 pair of gloves");

                dialogue.Add("Goal: Moderate weight gain and stamina
improvement.");

                dialogue.Add("It will cost you 35$.");

                gameManager.itemsToBuyFromVendingMachine["EnergyDrink"] = 1;

                gameManager.itemsToBuyFromVendingMachine["Protein"] = 1;

                gameManager.itemsToBuyFromVendingMachine["Gloves"] = 1;

                break;


            case "Overweight":

                dialogue.Add("You are 'Overweight' so I created a food program for
you.");

                dialogue.Add("The fitness instructor gave you a program that will
help you to lose weight, but you will have to eat less.");

                dialogue.Add("Let me show you what to buy from the vending
machine.");

                dialogue.Add("Your food program is ready.");

                dialogue.Add("2 energy drinks, 1 pair of gloves");

                dialogue.Add("Goal: Minimal weight gain but significant stamina
improvement.");

                dialogue.Add("It will cost you 20$.");

                gameManager.itemsToBuyFromVendingMachine["EnergyDrink"] = 2;

                gameManager.itemsToBuyFromVendingMachine["Gloves"] = 1;

                break;


            case "Obese":

                dialogue.Add("You are 'Obese' so I created a food program for
you.");

                dialogue.Add("The fitness instructor gave you a program that will
help you to lose weight, but you will have to eat less.");

                dialogue.Add("Let me show you what to buy from the vending
machine.");

                dialogue.Add("Your food program is ready.");
```

```
                dialogue.Add("1 energy drink, 1 pair of gloves");

                dialogue.Add("Goal: Minimal weight gain and minimal stamina
improvement.");

                dialogue.Add("It will cost you 15$.");

                gameManager.itemsToBuyFromVendingMachine["EnergyDrink"] = 1;

                gameManager.itemsToBuyFromVendingMachine["Gloves"] = 1;

                break;


            default:

                dialogue.Add("You need to check your physical condition with the
fitness instructor first.");

                break;

        }

        gameManager.hasSpokenToNpc = false;

        dialogueText.text = dialogue[0];

        continueButton.SetActive(true);

    }


    IEnumerator Typing()

    {

        foreach (char letter in dialogue[index].ToCharArray())

        {

            dialogueText.text += letter;

            yield return new WaitForSeconds(wordSpeed);

        }

    }

    public void AthleteHasInteractedWithNpc()

    {

        if (gameManager.hasSpokenToNpc == true)

        {

            getProgramButton.SetActive(true);
```

```
            continueButton.SetActive(false);

        }

    }

}
```

Αυτόματος Πωλητής (Vending Machine)

Το script PurchaseGymProducts επιτρέπει στον παίκτη να αλληλεπιδράσει με τον αυτόματο πωλητή στο γυμναστήριο για να αγοράσει προϊόντα γυμναστικής και προγράμματα διατροφής. Το script διαχειρίζεται τα πάνελ του UI, ελέγχει αν είναι κοντά ο παίκτης και ενημερώνει τα στατιστικά και το απόθεμα του παίκτη κατά την αγορά. Διασφαλίζει ότι ο παίκτης μπορεί να αγοράσει αντικείμενα μόνο αν έχει αρκετά χρήματα και ότι τα κουμπιά για την αγορά αντικειμένων ενεργοποιούνται ή απενεργοποιούνται με βάση τα υπόλοιπα αντικείμενα προς αγορά.

[6] PurchaseProducts

```
using System.Collections;

using System.Linq;

using UnityEngine;

using UnityEngine.UI;


public class PurchaseGymProducts : MonoBehaviour

{

    [SerializeField]

    public GameObject gameManagerComponent;


    [SerializeField]

    public GameObject vendingMachinePanel;


    private bool playerIsClose;

    [SerializeField]
```

```
    public GameObject energyItemsPanel;

    [SerializeField]

    public GameObject fitnessItemsPanel;

    [SerializeField]

    public GameObject energyDrinksButton;

    [SerializeField]

    public GameObject fitnessProductButton;

    [SerializeField]

    public GameObject exitButton;

    [SerializeField]

    public GameObject moneyWarning;

    [SerializeField]

    public Button buyEnergyDrinkButton;

    [SerializeField]

    public Button buyProteinButton;

    [SerializeField]

    public Button buyCreatineButton;

    [SerializeField]

    public Button buyGlovesButton;

    [SerializeField]

    public Button buyDippingBeltButton;

    private GameManager gameManager;


    private GameObject player;

    void Start()

    {

        player = GameObject.Find("Athlete");

        gameManager = gameManagerComponent.GetComponent<GameManager>();

    }
```

```
    public void DisplayVendingMachinePanel()
    {
        vendingMachinePanel.SetActive(true);
        player.GetComponentInChildren<MouseLook>().enabled = false;
        Cursor.lockState = CursorLockMode.Confined;

    }

    public void DisplayEnergyItems()
    {
        energyDrinksButton.SetActive(false);
        fitnessProductButton.SetActive(false);
        exitButton.SetActive(false);
        energyItemsPanel.SetActive(true);
    }

    public void DisplayFitnessItems()
    {
        energyDrinksButton.SetActive(false);
        fitnessProductButton.SetActive(false);
        exitButton.SetActive(false);
        fitnessItemsPanel.SetActive(true);
    }

    public void ReturnToCaregorySelection()
    {
        energyItemsPanel.SetActive(false);
        fitnessItemsPanel.SetActive(false);
        energyDrinksButton.SetActive(true);
        fitnessProductButton.SetActive(true);
```

```
        exitButton.SetActive(true);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Athlete"))
        {
            playerIsClose = true;
        }
    }
    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject.CompareTag("Athlete"))
        {
            playerIsClose = false;
            vendingMachinePanel.SetActive(false);
            player.GetComponentInChildren<MouseLook>().enabled = true;
            Cursor.lockState = CursorLockMode.Locked;
        }
    }

    public void BuyEnergyDrink()
    {
        if (gameManager.playerStats.Money >= 5f)
        {
            gameManager.playerStats.Money -= 5f;
            gameManager.playerStats.Weight += 0.2f;
            gameManager.playerStats.Stamina += 1.5f;
            gameManager.InitStats();
            gameManager.playerStats.Invetory["EnergyDrink"] += 1;
```

```
            gameManager.itemsToBuyFromVendingMachine["EnergyDrink"] -= 1;
            gameManager.athleteBoughtAllItems = BoughtAllItems();
        }
        else
        {
            StartCoroutine(DisplayMoneyWarning());
        }
    }

    public void BuyProtein()
    {
        if (gameManager.playerStats.Money >= 20f)
        {
            gameManager.playerStats.Money -= 20f;
            gameManager.playerStats.Weight += 0.5f;
            gameManager.playerStats.Stamina += 1f;
            gameManager.InitStats();
            gameManager.playerStats.Invetory["Protein"] += 1;
            gameManager.itemsToBuyFromVendingMachine["Protein"] -= 1;
            gameManager.athleteBoughtAllItems = BoughtAllItems();
        }
        else
        {
            StartCoroutine(DisplayMoneyWarning());
        }
    }

    public void BuyCreatine()
    {
        if (gameManager.playerStats.Money >= 30f)
```

```
        {
            gameManager.playerStats.Money -= 30f;
            gameManager.playerStats.Weight += 1f;
            gameManager.playerStats.Stamina += 0.5f;
            gameManager.InitStats();
            gameManager.playerStats.Invetory["Creatine"] += 1;
            gameManager.itemsToBuyFromVendingMachine["Creatine"] -= 1;
            gameManager.athleteBoughtAllItems = BoughtAllItems();

        }
        else
        {
            StartCoroutine(DisplayMoneyWarning());
        }
    }
    public void BuyGloves()
    {
        if (gameManager.playerStats.Money >= 10f)
        {
            gameManager.playerStats.Money -= 10f;
            gameManager.playerStats.Stamina += 0.1f;
            gameManager.playerStats.Weight += 0.3f;
            gameManager.InitStats();
            gameManager.playerStats.Invetory["Gloves"] += 1;
            gameManager.itemsToBuyFromVendingMachine["Gloves"] -= 1;
            gameManager.athleteBoughtAllItems = BoughtAllItems();
        }
        else
        {
            StartCoroutine(DisplayMoneyWarning());
```

```
        }
    }
    public void BuyDippingBelt()
    {
        if (gameManager.playerStats.Money >= 25f)
        {
            gameManager.playerStats.Money -= 25f;
            gameManager.playerStats.Stamina += 0.25f;
            gameManager.playerStats.Weight += 0.75f;
            gameManager.InitStats();
            gameManager.playerStats.Invetory["Belt"] += 1;
            gameManager.itemsToBuyFromVendingMachine["DippingBelt"] -= 1;
            gameManager.athleteBoughtAllItems = BoughtAllItems();

        }
        else
        {
            StartCoroutine(DisplayMoneyWarning());
        }

    }
    public void ExitSnackMachine()
    {
        vendingMachinePanel.SetActive(false);
        player.GetComponentInChildren<MouseLook>().enabled = true;
        Cursor.lockState = CursorLockMode.Locked;
    }
    private IEnumerator DisplayMoneyWarning()
    {
```

```
        moneyWarning.SetActive(true);

        yield return new WaitForSeconds(2f);

        moneyWarning.SetActive(false);


    }

    public void Update()

    {

        if (Input.GetKeyDown(KeyCode.E) && playerIsClose)

        {

            DisplayVendingMachinePanel();

        }

        CanBuyEnergyDrink();

        CanBuyProtein();

        CanBuyCreatine();

        CanBuyGloves();

        CanBuyDippingBelt();


    }

    public void CanBuyEnergyDrink()

    {

        gameManager.itemsToBuyFromVendingMachine.TryGetValue("EnergyDrink", out
int energyDrinksToBuy);

        if (energyDrinksToBuy == 0)

        {

            buyEnergyDrinkButton.interactable = false;

        }

        else buyEnergyDrinkButton.interactable = true;

    }

    public void CanBuyProtein()

    {
```

```
        gameManager.itemsToBuyFromVendingMachine.TryGetValue("Protein", out int
proteinToBuy);

        if (proteinToBuy == 0)

        {

            buyProteinButton.interactable = false;

        }

        else buyProteinButton.interactable = true;

    }

    public void CanBuyCreatine()

    {

        gameManager.itemsToBuyFromVendingMachine.TryGetValue("Creatine", out int
creatineToBuy);

        if (creatineToBuy == 0)

        {

            buyCreatineButton.interactable = false;

        }

        else buyCreatineButton.interactable = true;

    }

    public void CanBuyGloves()

    {

        gameManager.itemsToBuyFromVendingMachine.TryGetValue("Gloves", out int
glovesToBuy);

        if (glovesToBuy == 0)

        {

            buyGlovesButton.interactable = false;

        }

        else buyGlovesButton.interactable = true;

    }

    public void CanBuyDippingBelt()

    {
```

```
        gameManager.itemsToBuyFromVendingMachine.TryGetValue("DippingBelt", out
int beltToBuy);

        if (beltToBuy == 0)

        {

            buyDippingBeltButton.interactable = false;

        }

        else buyDippingBeltButton.interactable = true;

    }

    public bool BoughtAllItems()

    {

        return gameManager.itemsToBuyFromVendingMachine.All(item => item.Value ==
0);

    }




}
```

Έξοδος Παιχνιδιού (Exit Door)

Το script ExitDoor διαχειρίζεται την αλληλεπίδραση του παίκτη με μια πόρτα εξόδου, επιτρέποντάς του να μεταβεί από την τρέχουσα σκηνή στην σκηνή "Μενού". Ελέγχει αν ο παίκτης βρίσκεται κοντά στην πόρτα και ανταποκρίνεται στην είσοδο του παίκτη για να ενεργοποιήσει την αλλαγή σκηνής.

[7] ExitDoor.cs

```
using UnityEngine;

using UnityEngine.SceneManagement;



public class ExitDoor : MonoBehaviour

{

    private bool playerNearDoor;
```

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Athlete"))
        {
            playerNearDoor = true;
        }
    }
    private void OnTriggerExit(Collider other)
    {

        if (other.gameObject.CompareTag("Athlete"))
        {
            playerNearDoor = false;
        }
    }
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.E) && playerNearDoor)
        {
            Cursor.lockState = CursorLockMode.Confined;
            SceneManager.LoadSceneAsync("Menu");
            Destroy(GameManager.manager);
        }
    }
}
```

Βασικό Μενού Ελέγχου (Game Manager)

Η βασικότερη οντότητα του παιχνιδιού. Ελέγχει τα πάντα .

Αρχικά περιέχει τα χαρακτηριστικά του χρήστη :

```
public class PlayerStats

{



    public Dictionary<string, int> Invetory;

    public float Weight;

    public float Stamina;

    public float Money;

    public float Height;

    public int Age;



    public float CaloriesBurned()

    {

        return 0.0f;

    }

    public PlayerStats(float weight, float stamina, float money, float
height, int age)

    {

        this.Weight = weight;

        this.Stamina = stamina;

        this.Money = money;

        this.Height = height;
```

```
        this.Age = age;

        Invetory = new Dictionary<string, int>();

    }

    public PlayerStats()

    {

        Invetory = new Dictionary<string, int>();

    }

    public PlayerStats(int money, int stamina)

    {

        this.Money = money;

        this.Stamina = stamina;

        Invetory = new Dictionary<string, int>();

    }

    public override string ToString()

    {

        return $"Weight: {Weight}, Stamina: {Stamina}, Money: {Money},
Height: {Height}, Age: {Age}";

    }


}
```

Ελέγχει αν ο χρήστης έχει τελειώσει με τις ασκήσεις που του έχει αναθέσει ο γυμναστής και τα χρονικά timers .

Ενδεικτικά για την μπάρα :

```
  if (AthleteCollider != null && BarCollider != null)

        {

            if (AthleteCollider.bounds.Intersects(BarCollider.bounds))

            {

                updateGeneralTimer(ref GeneralTimer);
```

```
                checkExerciseComplition(ref equipmentTimers.BarTimer,
equipmentUse.BarUse, out isBarFinished);

                TimeSpan timeSpan =
TimeSpan.FromSeconds(equipmentTimers.BarTimer);

                ScreenTimer.text = string.Format("{0:ss\\:ff}",
timeSpan); // show seconds and ms

                if (isBarFinished)

                {

                    bonusMoney = rand.Next(5, 15);

                    playerStats.Money += bonusMoney;

                    InitStats();

                    equipmentUse.BarUse = -2;

                    tasksToFinish["Bar"] = -2;

                }

                else if (equipmentUse.BarUse == -2)

                {

                    barFinished.SetActive(true);

                }

            }

            else barFinished.SetActive(false);

        }
```

Υπολογίζει το bmi που με βάση αυτό ορίζει ο γυμναστής και ο διατροφολόγος το πρόγραμμα που πρέπει να του δωθεί.

```
    public float CalculateBMI(PlayerStats playerStats)

    {

        BMI = playerStats.Weight / (playerStats.Height *
playerStats.Height);

        return BMI;

    }
```

Είναι υπεύθυνο να αλλάζει το UI των χαρακτηριστικών του χρήστη με βάση τα inputs του ,να ανοίγει το βοηθητικό UI και το inventory .

```
public void ActivatePanel()
{
    PlayerStatsPanel.SetActive(true);
}
public void HelpPanelActivate()
{
    if (Input.GetKeyDown(KeyCode.F1))
    {
        helpPanel.SetActive(!helpPanel.activeSelf);
    }
}
public void InventoryPanelActivate()
{
    if (Input.GetKeyDown(KeyCode.I))
    {
        UpdateInventoryValues();
        inventoryPanel.SetActive(!inventoryPanel.activeSelf);
    }
}
```

Να υποστηρίζει την λειτουργία του pause

```
public void PausePanelActivate()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
```

```
            PauseGame();

            pausePanel.SetActive(true);

        }

    }

    public void ResumeGameButton()

    {

        pausePanel.SetActive(false);

        UnpauseGame();

    }

    public void ExitGameButton()

    {

        SceneManager.LoadSceneAsync("Menu");

        Destroy(GameManager.manager);

    }
```

Τέλος , περιέχει ένα γενικό timer του παιχνιδιού ετσι ώστε να υπάρχει αλληλεπίδραση με τα χαρακτηριστικά του χρήστη (π.χ. λαμβάνει χρήματα κάθε 10 δευτερόλεπτα ή αυξάνεται το stamina του)

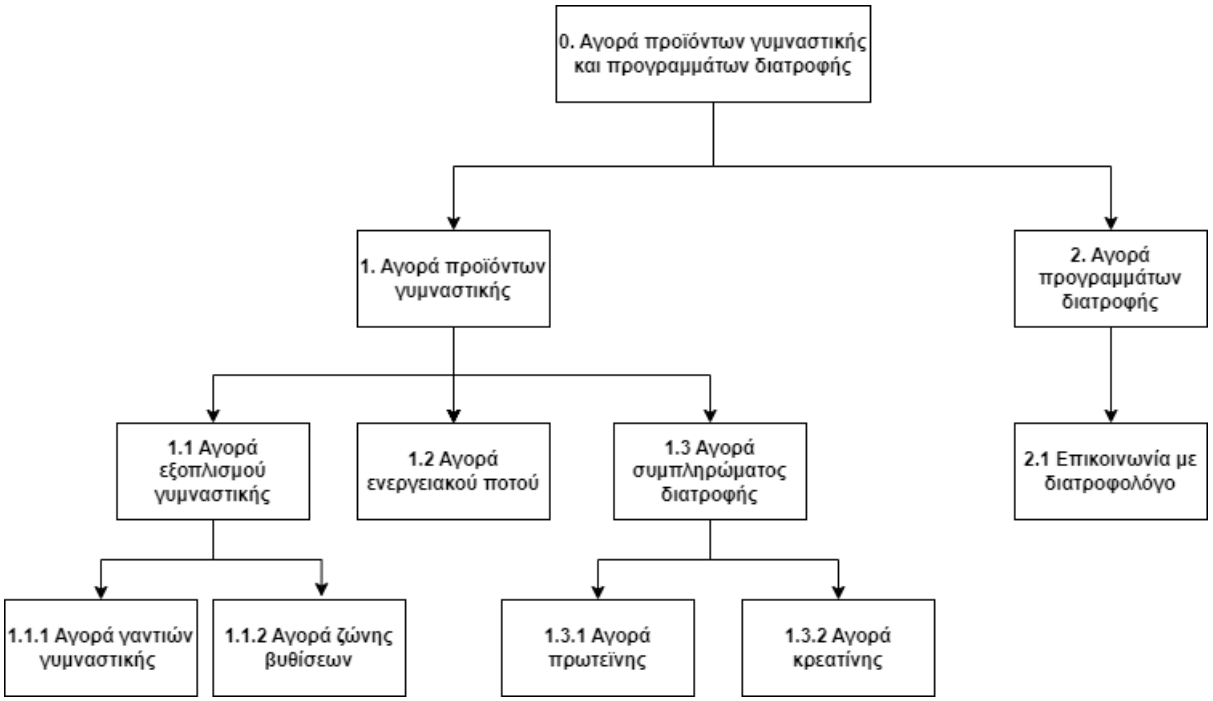
```
public void updateGeneralTimer(ref float Timer)

    {

        Timer += Time.deltaTime;

        int generalTimerToInt = (int)Timer;

        if (generalTimerToInt / 10 > 0)

        {

            playerStats.Money += 1;

            if (playerStats.Weight != 0)

                playerStats.Weight -= 0.1f;

            playerStats.Stamina += 0.1f;

            InitStats();
```

```
            Timer = 0;

        }

    }
```

Online-Help

Ο χρήστης μπορεί να δει πληροφορίες για τις διαθέσιμες υπηρεσίες της εφαρμογής του εικονικού γυμναστηρίο πατώντας “F1” και βλέποντας το User-Manual

Αγορά προϊόντων γυμναστικής και προγραμμάτων διατροφής



Το διάγραμμα απεικονίζει τη διαδικασία αγοράς προϊόντων γυμναστικής και προγραμμάτων διατροφής μέσα στο παιχνίδι. Ξεκινώντας από την κεντρική κατηγορία, ο παίκτης έχει δύο κύριες επιλογές: την αγορά προϊόντων γυμναστικής και την αγορά προγραμμάτων διατροφής. Στην πρώτη κατηγορία, η αγορά προϊόντων γυμναστικής περιλαμβάνει την αγορά εξοπλισμού, όπως γάντια και ζώνες, την αγορά ενεργειακών ποτών, και την αγορά συμπληρωμάτων διατροφής, όπως πρωτεΐνη και κρεατίνη. Στη δεύτερη κατηγορία, η αγορά προγραμμάτων διατροφής περιλαμβάνει την επικοινωνία με τον διατροφολόγο. Το διάγραμμα παρέχει μια σαφή δομή των διαθέσιμων επιλογών και των υποκατηγοριών τους,

βοηθώντας τον παίκτη να κατανοήσει και να πλοηγηθεί εύκολα στις δυνατότητες του παιχνιδιού.

Σας ευχαριστούμε για την προσοχή σας