HUMAN & COMPUTER INTERACTION

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

VIRTUAL GYM

TECHNICAL MANUAL

TEAM MEMBER DETAILS

NAME : ATHANASIOU VASILEIOS EVANGELOS
STUDENT ID: 19390005
NAME : ROMANIUK VIKTOR
STUDENT ID : 713242017024
NAME : PETROPOULOS PANAGIOTIS
STUDENT ID : 20390188
NAME : THEOCHARIS GEORGIOS
STUDENT ID : 19390283

CONTENTS

Introduction

Unity is a cross-platform software development platform primarily used for creating games and interactive content across multiple platforms. By using programming languages like C#, developers can create games with high quality and simplicity. A flexible development environment is also provided, with tools that allow the creation of graphics, sounds, physics and other elements required for games.

Installation Instructions

First add the app to Unity Hub by pressing the Add button .

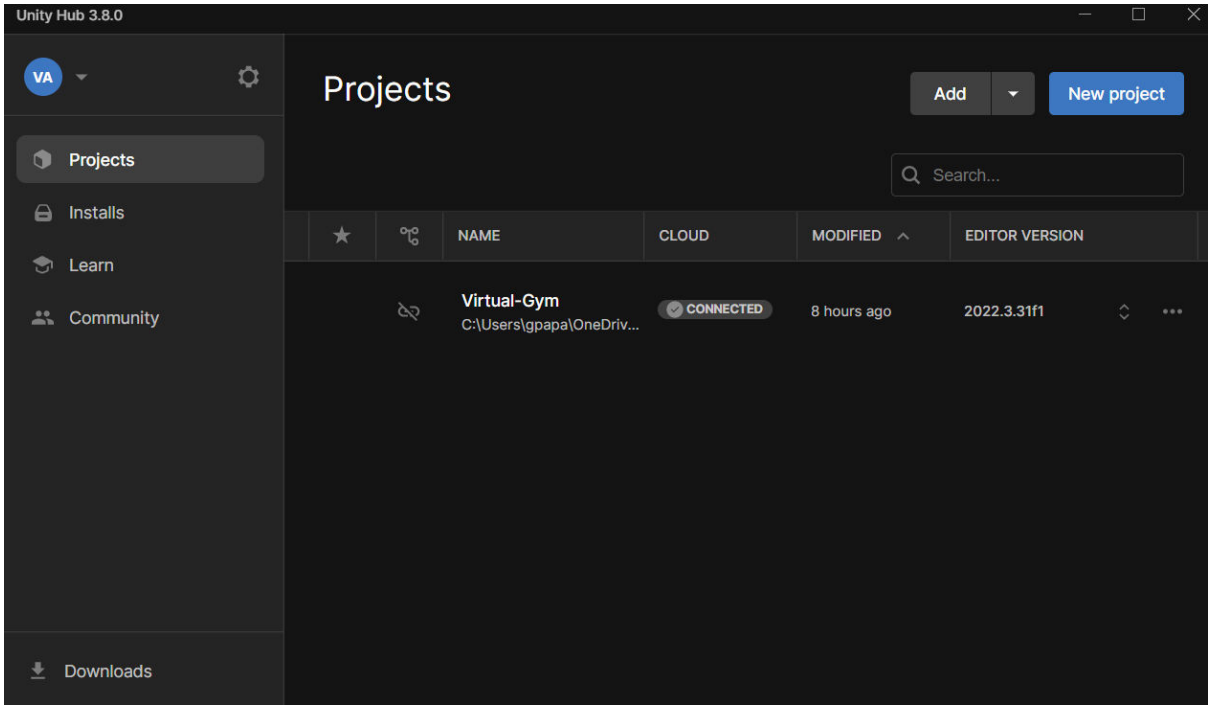


Figure 1. Unity Hub

Then by pressing Ctrl + B you start the Build process of the application to generate the executable file.

Figure 2 . Unity

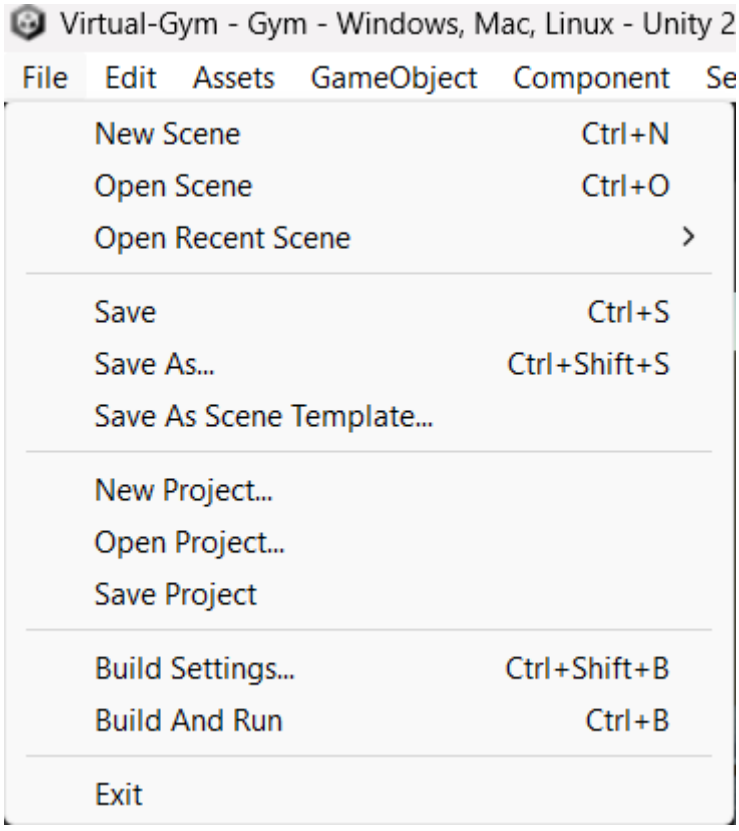


Figure 3. Build and Run

Going in file explorer and at archives of the project, go in the Build Folder.

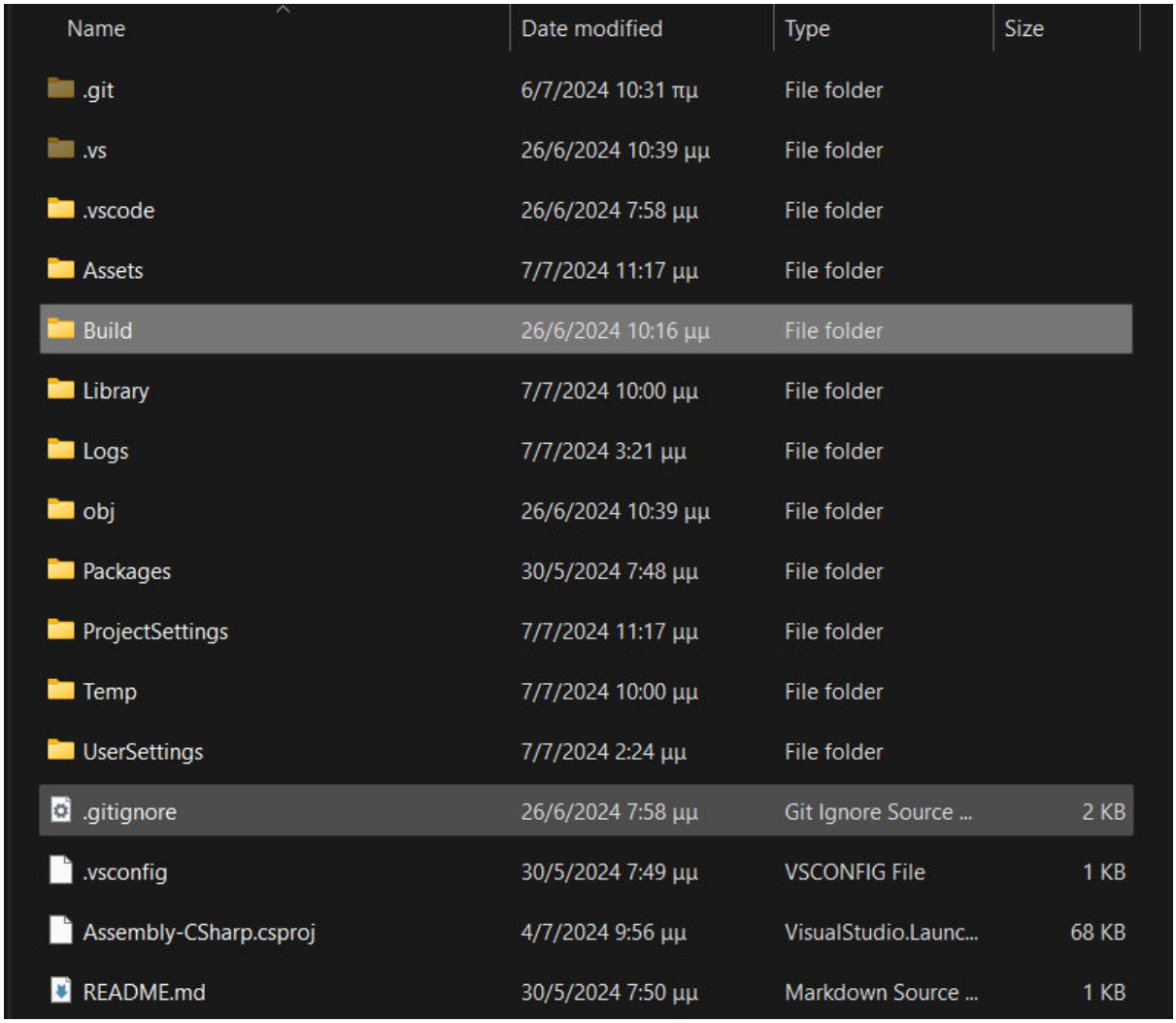


Figure 4. Project folder

Click on Virtual - Gym . exe and start the application.

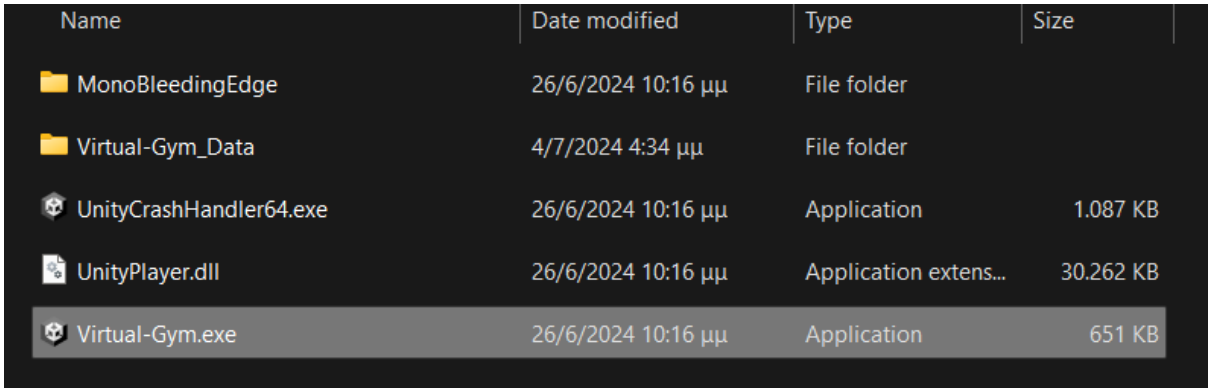


Figure 5. Executable folder

Description

As part of the work, a virtual gym was developed through Unity (2022.3.31f1) which enables the user to interact and perform various functions, such as exercises on gym equipment, communication with a trainer, etc.

The application code files are also located in the repository below

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

Detailed Presentation of Functions

The functions are presented in detail in the concise user manual (User-Manual.pdf)

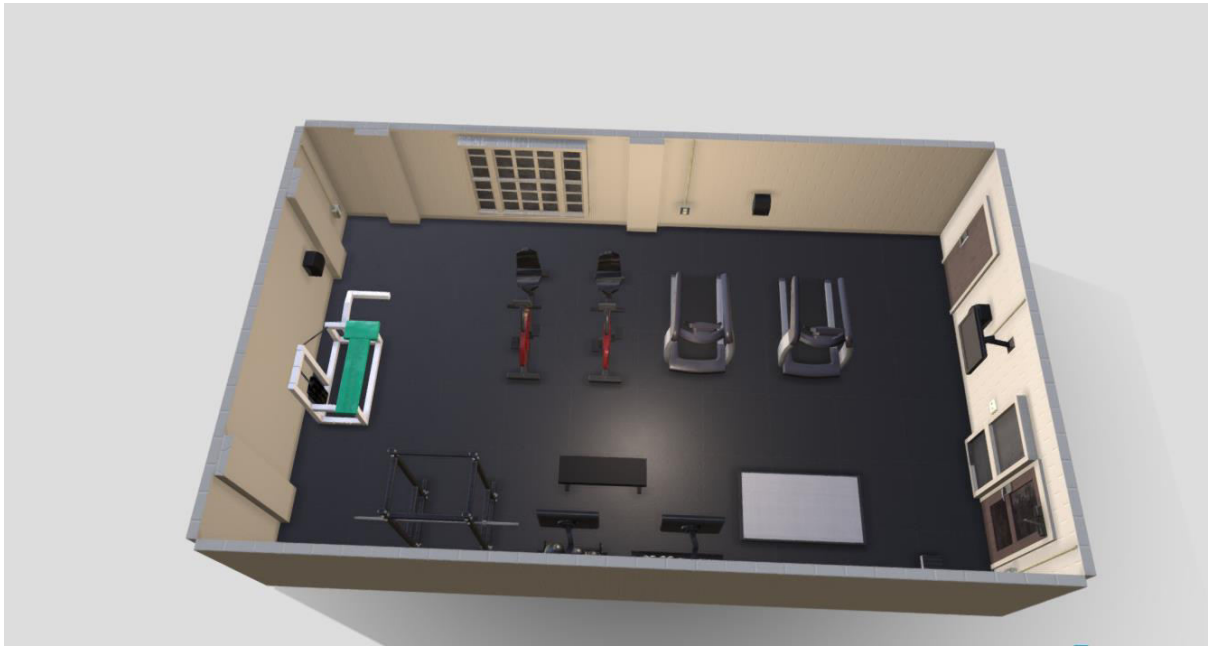
Design and Implementation of Graphics

To implement the graphics, we downloaded the assets from the internet for free and added them to the project folder. In detail, we have the following assets:

Gym area

The gym area was taken from the website "sketchfab" which contains many models. The gym model was ready, however we made the appropriate parameters so that it served our purpose. For example, in the original model, the components of the gym were very close to each other, causing multiple collisions. We were able to parameterize it because each object in the space is "separate" and not unified.

Before:



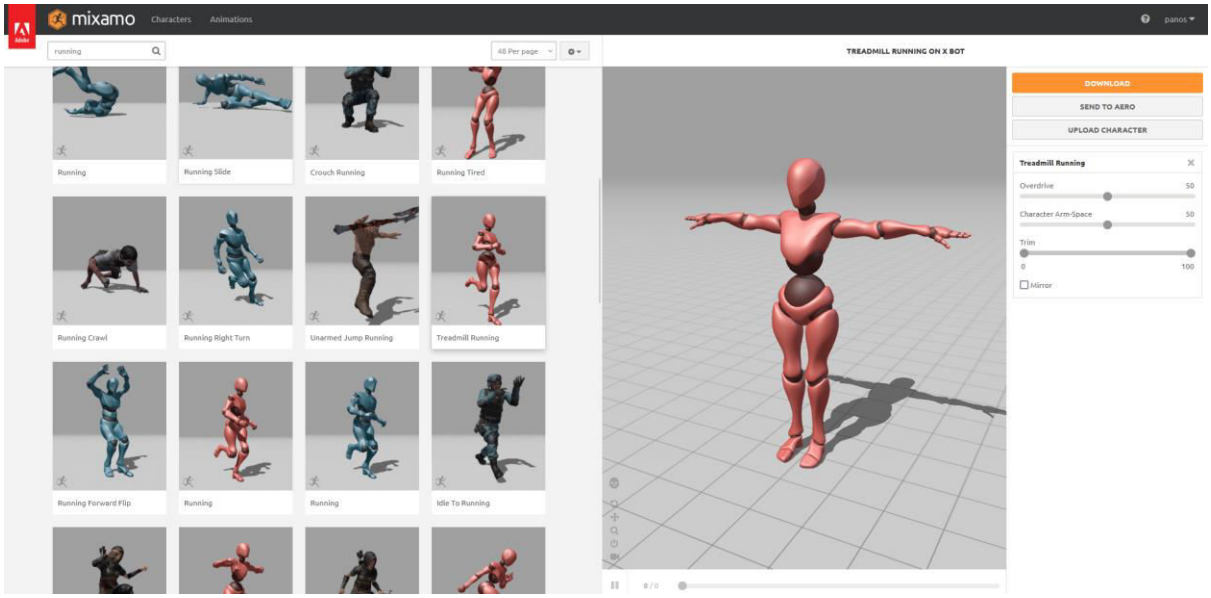
After:

Source:

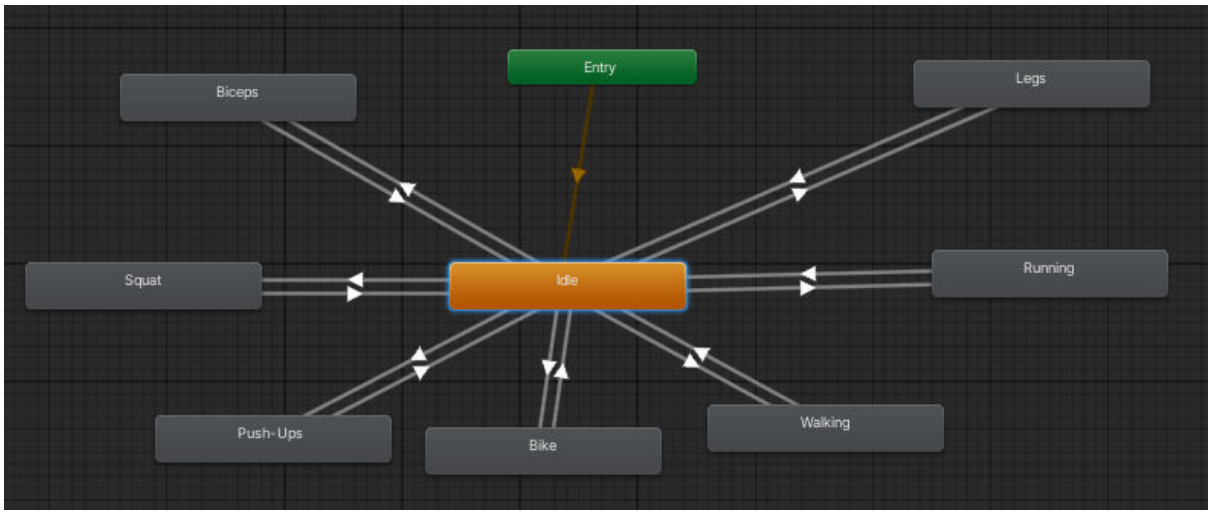
https://sketchfab.com/3d-models/modular-gym-86e8cc43d76c44e3925e625fdab155f3

Athlete

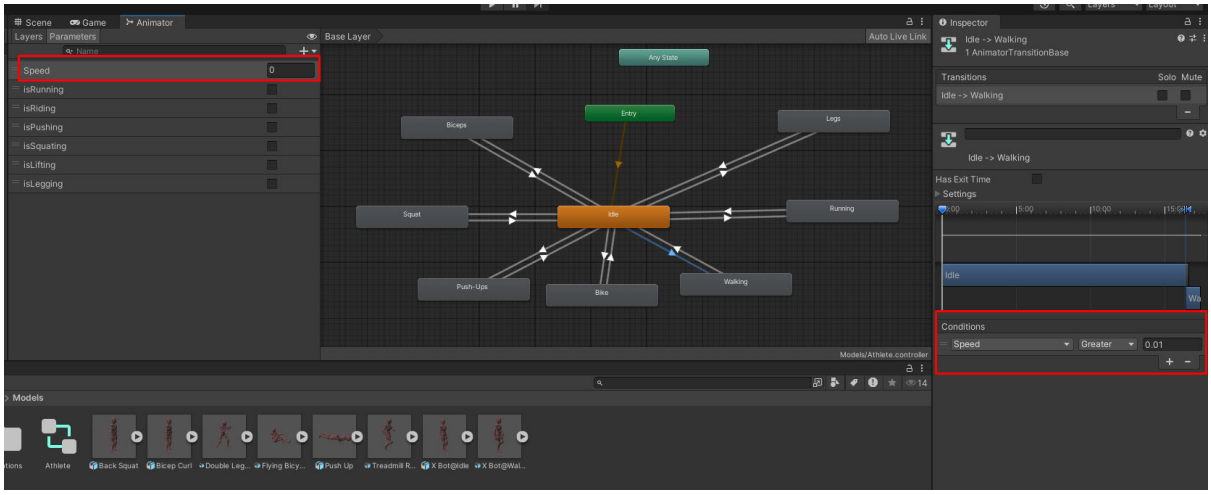
Both the model of the athlete and the animations of the idle, walking and fitness exercises were taken from the “Mixamo” website.



We implemented the animations in the different situations that the player can find themselves in through Unity, specifically through the Unity animator.



Each node of the animation is a different state. The transition from one node to another is done by changing the “state” variable. For example, when the user is not moving, he is permanently in the idle state. A transition factor is when the user presses the w key it moves the player i.e. increases their speed. Therefore, when the athlete has a speed < 0.1, so he is not moving, he is in the animation idle state, while when he presses the w key, he switches to the walking state:



The exact same thing happens with the rest of the animations, only that in them we have used a boolean variable for whether or not they are in the right position-moment-situation for them to be activated. For example, when the player collides with the corridor, then the global variable of the animation is Walking changes from false to true, so we also have a transition to the walking state. When the collision exits then the variable becomes false.

The following code for the athlete changes the speed value depending on whether he is walking or not (ie pressing 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 );

    }
```

While in the following it checks the collisions to define which animation should run and which state it should go to:

```
private void OnTriggerEnter ( Collider others )
{
        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 ( "isSquatting" , 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 others )

{

        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 ( "isSquatting" , false );
}
            else if ( other . gameObject . CompareTag ( "Weights" ))
{
                animator . SetBool ( "isLifting" , false );
}
            else if ( other . gameObject . CompareTag ( "Leg Extension"
))
            {
                animator . SetBool ( "isLegging" , false );
}
}


}
```

Trainer

The same applies to the athlete only that the color has been changed to blue. It is also permanently in the idle state

Nutritionist

The same applies to the athlete only that the color has been changed to green. It is also permanently in the idle state

Vending machine

Pulled from the unity assets store.

Source:

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

Home Menu Scene

The first scene shows the initial menu with 2 buttons “Play” and “Quit”. Essentially, the user has the option to choose to play the game or close it. The background of the menu includes an image of a gym, while the buttons have their own design. That is, it is a Canvas object.

As for the functions of the “Play” and “Quit” buttons, they are activated by the onClick() method where the functions StartGame() are called in case the user presses the “Play” button and StopGame() in case the user presses the “Quit” button.

The StartGame() and StopGame() functions are methods of the StartStopGame class found in the source file below

[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();

    }

}
```

Gym scene

Athlete

The athlete is the user who interacts with the virtual gym. Its control is implemented with two scripts that handle the game camera and the player's movement respectively.

Camera

The game's camera is mounted on the user's head, which means that his movement is First-Person. In other words, the view of the user in the game is from his own eyes, without showing his model in the view (Third-Person). The player's camera is controlled by the MouseLook.cs script

[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);

    }

}
```

We set the mouse sensitivity to 100, where it determines how fast the mouse movements will be. We also define the movements of the mouse in both the horizontal X-axis and the vertical Y-axis. As for the movement of the mouse in the Z-axis, it is defined by the function Quaternion.Euler() which handles the rotation of the player in all 3 axes.

Move

The AthleteMovement.cs script handles the movement and animations of the user-athlete. It uses a CharacterController for movement, allowing the character to move based on player input, and controls whether gravity effects are applied to the character. The script manages various animations (running, pushing, etc.) that are triggered by interactions with various exercise instruments using the OnTriggerEnter and OnTriggerExit methods. Camera position and rotation adjust when the character interacts with these objects to provide appropriate visual feedback. The Update method constantly checks the player's input for character movement and updates the animation state accordingly, ensuring a smooth and flexible gameplay experience.

[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("isSquatting", 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("isSquatting", 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);

    }
```

```
}
```

Fitness Instructor

The user's communication with the trainer is implemented in the NPCInteraction.cs script. When the player is near the NPC (gym) and presses the "E" key, a dialog panel appears showing messages from the NPC. The script manages different dialogue sequences depending on the player's progress, which are controlled by the main GameManager script. It also makes it easy to enter player characteristics, which are used to create personalized fitness programs based on the player's BMI (Body Mass Index). The NPC provides specific exercise routines based on the player's physical condition (underweight, normal, overweight, obese), guiding the player through various gym activities .

[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
{
dialogPanel.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;
dialogPanel.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)
{
```

```
dialogPanel.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)
```

```
{

"Underweight" case:

{

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);

breaks?

}

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);

breaks?

}

"Overweight" case:

{

gameManager.physicalCondition = "Overweight";

dialogue.Add("You are 'Overweight' 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' of Bicycle, 20' of 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);

breaks?

}

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);


breaks?

}




}

dialogueText.text = dialogue[0];

dialogPanel.SetActive(true);

}


IEnumerator Typing()

{

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

{

dialogueText.text += letter;

yield return new WaitForSeconds(wordSpeed);

        }

}

}
```

Nutritionist

The user's communication with the nutritionist is implemented in the NutritionistInteraction.cs script. The philosophy is the same as the NPCInteraction.cs script that manages the user's communication with the trainer. The difference is in the dialogues, where the nutritionist produces nutrition programs according to the physical condition of the user. The physical condition is stored in the “physical_condition” variable of the main GameManager control object, where it only gets a value if the user contacts the trainer to enter their information (mass, age, height).

[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

{

dialogPanel.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;
dialogPanel.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)

{

dialogPanel.SetActive(false);

playerIsClose = false;

}

}


public void PassInputsToGameManager()

{

gameManager.ActivatePanel();

FoodProgramBasedOnPhysicalCondition();

}


public void FoodProgramBasedOnPhysicalCondition()

{

dialogue.Clear();


switch (gameManager.physicalCondition)

{

"Underweight" case:

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;

breaks?


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;

breaks?


"Overweight" case:

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 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;
```

```
breaks?


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 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;

breaks?


default:

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

breaks?

}

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);

        }

}

}
```

Vending Machine

The PurchaseGymProducts script allows the player to interact with the vending machine at the gym to purchase gym products and diet plans. The script manages the UI panels, checks if the player is nearby, and updates the player's stats and inventory when purchased. It ensures that the player can only buy items if they have enough money, and that buttons to buy items are enabled or disabled based on the remaining items to buy.

[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.Inventory["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.Inventory["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.Inventory["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.Inventory["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.Inventory["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);

    }




}
```

Exit Door

The ExitDoor script manages the player's interaction with an exit door, allowing them to switch from the current scene to the "Menu" scene. Checks if the player is near the door and responds to player input to trigger the scene change.

[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);
}
}
}
```

Basic Menu Control (Game Manager)

The most basic entity of the game. He controls everything.

Initially it contains the characteristics of the user :

```
public class PlayerStats

{



    public Dictionary < string , int > Inventory ;

    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 ;

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

}
```

```
    public PlayerStats ()
{
        Inventory = new Dictionary < string , int >();
}
    public PlayerStats ( int money , int stamina )
{
        this . Money = money
        this . Stamina = stamina
        Inventory = new Dictionary < string , int >();
}
    public override string toString ()
{
        return $"Weight: { Weight }, Stamina: { Stamina }, Money: {
Money }, Height: { Height }, Age: { Age }" ;
    }

}
```

It checks if the user has finished the exercises assigned by the trainer and the timers.

Indicative for her bar :

```
  if ( AthleteCollider != null && BarCollider != )
{
            if ( AthleteCollider . bounds . Intersects ( BarCollider .
bounds ))
{
                updateGeneralTimer ( ref GeneralTimer );
                checkExerciseCompletion ( 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 );

        }
```

It calculates the bmi on the basis of which the trainer and the nutritionist define the program that should be given to him.

```
    public float CalculateBMI ( PlayerStats playerStats )

{

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

        return BMI ?

}
```

It is responsible for changing the UI of the user's features based on his inputs, opening the auxiliary UI and the 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
);
        }
}
```

To support the pause function

```
    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 );
    }
```

Finally, it contains a general timer of the game so that there is interaction with the user's characteristics (e.g. he receives money every 10 seconds or his stamina increases)

```
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 ;
```

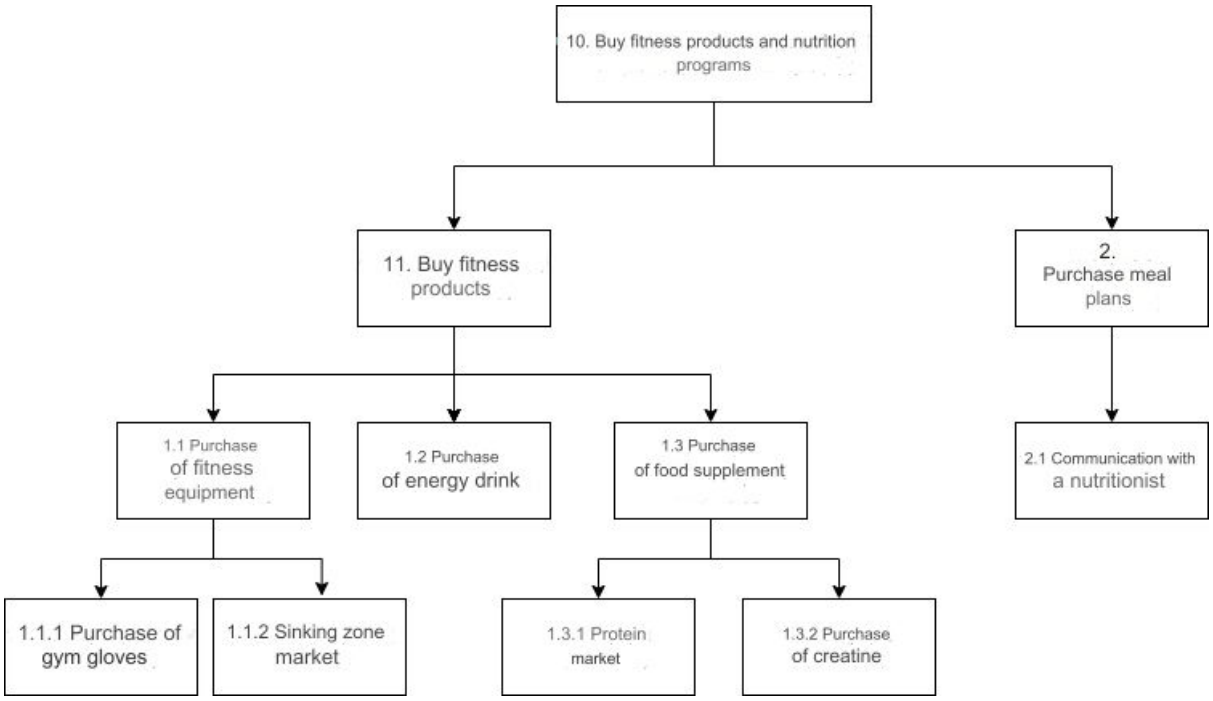
```
}

}
```

Online-Help

The user can see information about the available services of the virtual gym application by pressing " F 1" and viewing the User - Manual

Buy fitness products and nutrition programs



The diagram illustrates the process of purchasing fitness products and nutrition programs within the game. Starting from the main category, the player has two main options: buying fitness products and buying diet plans. In the first category, the purchase of fitness products includes the purchase of equipment, such as gloves and belts, the purchase of energy drinks, and the purchase of nutritional supplements, such as protein and creatine. In the second category, the purchase of nutrition plans includes communication with the nutritionist. The diagram provides a clear structure of the available options and their sub-categories, helping the player to easily understand and navigate the game's features.

HUMAN & COMPUTER INTERACTION

Thank you for your attention