COMP4701-INTRODUCTION TO GAME DEVELOPMENT

COIN COLLECTOR GAME PROJECT

Student Names and Numbers:

Sertaç Çakır – 20COMP1023

Mehmet Gökhan Kıyıcı – 20SOFT1057

● GAME DESCRIPTION

For our project, we made a game named **Coin Collector**. It is a 3D platform game and we developed it by using Unity Game Engine. The aim of the game is simple. Player controls a capsule shaped character and try to collect all gold coins in the map. The game has some basic mechanics to make it playable and challenging.

First, there is a **health system**. Player has 3 health points. When the player touch Hazard or Enemy objects which are red capsules, health decreases. If health becomes zero, game is over.

Also, there is a **time limit** in the game. A countdown timer is running while playing. Player must collect all coins before the time finish. If the time runs out, Game Over screen shows.

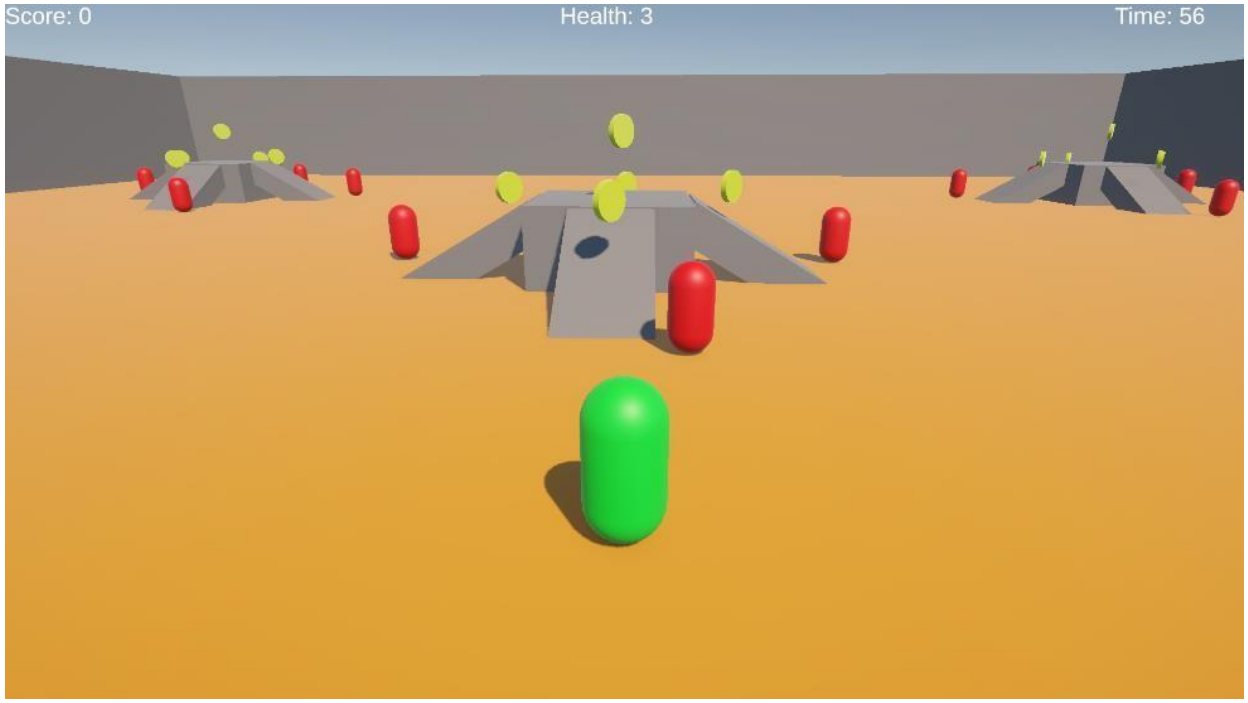
If the player collects all the coins before the time and if player health is not zero, **You Win** screen will appear.

We also added some **user interface elements**. There is a Main Menu and a Pause Menu. Player has some options like resume game, restart game, quit game, start game and by clicking on Settings he/she can change the volume level. During the game, UI shows score, health and time on the top of the screen.

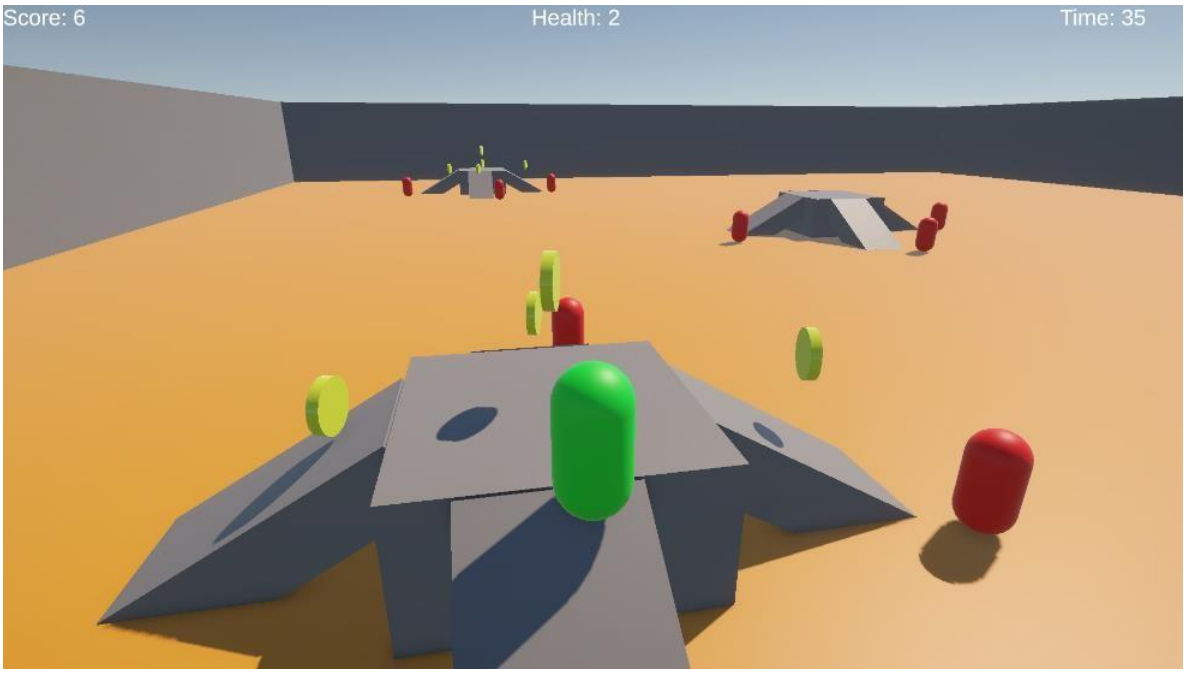
- **Screenshots of Gameplay**

a) Main Menu

b.1) Gameplay View



b.2) Gameplay View



6 Coins collected and 1 damage taken from enemy. 35 seconds left to end the game.

c) Pause Panel

Game paused. We see pause panel on screen here.

d.1) Lose Panel

Game lost because no health remaining.

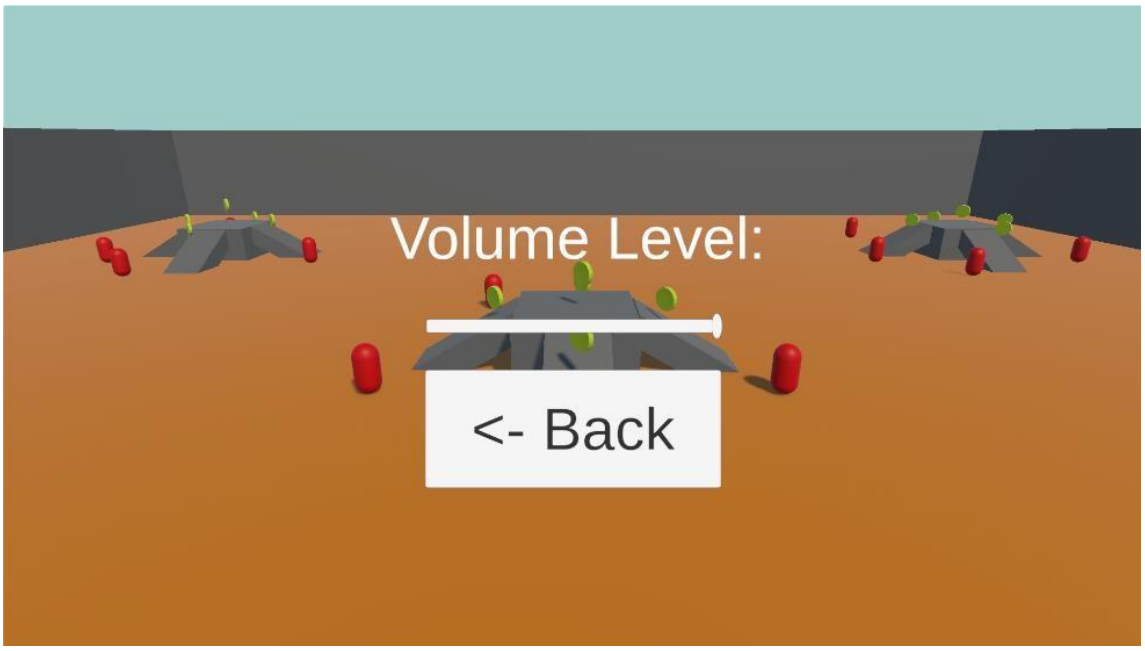
d.2) Lose Panel

Game lost because no time left.

e) Win Panel

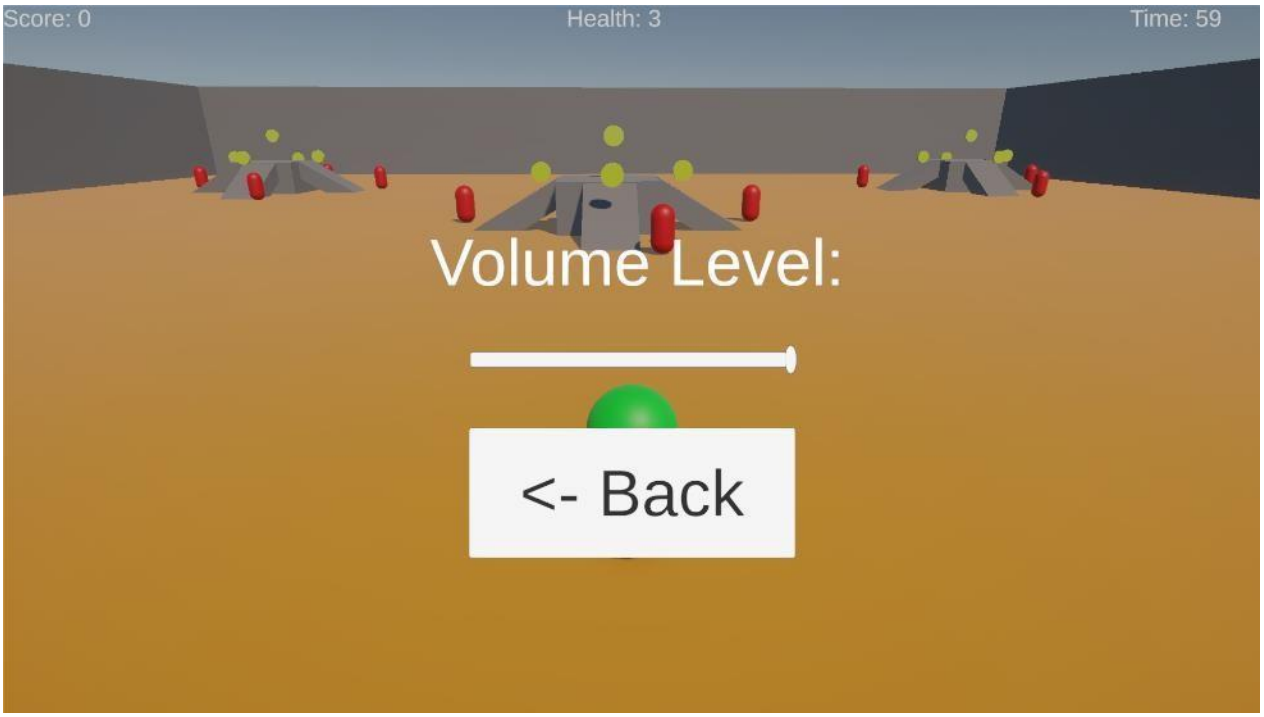
Game won because all the coins have been collected.

f.1) Settings Panel (In Main Menu)



Player can change volume level and turn back to main menu

f.2) Settings Panel (In Game)



Player can change volume level and turn back to pause panel

● Explanation of Core Scripts

We wrote several C# scripts to control the game logic.These scripts are used for player movement, game control and menu systems. Below is a basic explanation of how our code is structured and which Unity functions we used.

1) PlayerController.cs

This script is the main controller of the player character. It controls movement, jumping and also health logic. The movement is physics based and we used the Rigidbody component for this. Player input is taken by using Input.GetAxis. Horizontal axis is used for rotation and Vertical axis is used for forward movement.

Unity 6 Adaptation:
While using the course notes for Rigidbody movement, we noticed that rigibody.velocity is deprecated in Unity 6 and gives warning messages. Because of this, we searched the Unity documentation and replaced it with rb.linearVelocity. This way the code works correctly and without warnings.

Collision & Interactions:
We used the OnCollisonEnter function to handle collisions. With this function, we check if the player touches the Ground so jumping is allowed by using an isGrounded boolean. If the player hits a Hazard or Enemy, the TakeDamage( ) function is called.

Game State Management:
This script also communicates with the GameManager to update the UI when the player takes damage. When the player health becomes zero, the Die ( ) function is executed. In this function, the camera is un-parented to avoid rendering problems and then the Game Over screen is activated.

2)GameManager.cs:
This script serves as the central "brain" of our project, managing the overall game loop and global variables. Building upon the core logic structures taught in the course, we expanded this script to handle multiple systems simultaneously:

Game State & UI Management: It tracks critical variables like Score, Health, and the Countdown Timer. It acts as a bridge between the code and the Canvas, updating TextMeshPro elements in real-time so the player receives immediate feedback.

Time Control (Pause System): We implemented a pause mechanism using Time.timeScale. When the player presses ESC, time freezes (0f), and when they resume, it returns to normal (1f).

Script Communication (Unity 6 Update): To allow other scripts (like Collectibles) to find the Manager, we used object search functions. *Note:* While older course resources often use FindObjectOfType, we researched Unity 6 best practices and used Object.FindFirstObjectByType instead, as it is the modern, more optimized standard.

Camera Handling: In the WinGame and GameOver functions, we added specific logic to un-parent the Main Camera before disabling the player. This prevents the "No Cameras Rendering" error we encountered during testing.

3)Collectibles.cs

This script manages the behavior of the collectible items (gold coins) placed throughout the level.

Trigger Logic: Instead of using standard physical collisions (which would cause the player to bounce off the coin), we implemented the OnTriggerEnter function. This allows the player to pass through the object smoothly while triggering the collection event.

Script Communication: When a coin is collected, the script locates the GameManager to execute the AddScore() function. Consistent with our other scripts, we used the updated Object.FindFirstObjectByType command to ensure compatibility with Unity 6 standards.

Audio Handling: We addressed a common issue where sound effects cut off immediately when an object is destroyed. To prevent this, we used AudioSource.PlayClipAtPoint. This function creates a temporary audio source at the coin's position, allowing the collection sound to play completely even after Destroy(gameObject) removes the coin from the scene.

4) Rotator.cs

This script is purely for visual polish. We attached it to the "Coin" prefabs to make them spin continuously, making them more noticeable and attractive to the player.

Rotation Logic: Inside the Update loop, we used the transform.Rotate function to apply rotation around the object's axis.

Frame-Rate Independence: As emphasized in the course, we multiplied the rotation speed by Time.deltaTime. This ensures that the coins spin at the exact same speed on every computer, regardless of whether the game is running at 30 FPS or 144 FPS. Without this, the rotation speed would vary based on the computer's performance.

5) Hazard.cs

This script manages the movement of the obstacles.

Simple Patrol Logic: Although the course covers NavMeshAgent for complex AI behavior, we opted for a lightweight approach for simple hazards. We stored the object's starting position in Start(). In the Update() loop, we move the object continuously using transform.Translate.

Direction Control: We calculate the distance between the current position and the start position. If the object goes further than the specified range, we multiply its direction variable by -1 to make it turn back. This creates a loop without needing advanced Unity functions.

Collision: We used OnTriggerEnter to detect the player and apply damage, just like in the Enemy script.

6) MainMenuController.cs

This script controls the main menu of the game and the UI navigation before gameplay starts.

Scene Management:
Similar to restarting the level in GameManager, we used SceneManager.LoadScene(1) to move from the Main Menu scene to the gameplay scene. This allows the game to start when player press the Start button.

UI Panel Management:
Using the logic we learned in class for UI systems, we managed showing and hiding panels. We created Main Panel and Settings Panel and used SetActive true or false so only one panel is visible at the same time. This makes the menu simple and not confusing.

Audio Control:
We added a SetVolume function and connected it to a UI Slider. This function changes AudioListener volume, so player can control the sound level from the menu.

Application Control:
The QuitGame function uses Application Quit to close the game when the Quit button is clicked.

What We Learned From The Project

Developing Coin Collector was a useful learning experience for us. It helped us change the theory we learned in lectures into a working game.

During the project, we tried to follow the course topics but also faced many technical problems. We learned how scripts communicate with each other, how UI and scenes work together, and how small errors can cause big problems in the game. Also, using Unity 6 forced us to read documentation instead of copying code from internet.

Overall, this project improved our knowledge about Unity, C# scripting and basic game structure.

Applying Course Concepts to Scene Management: We solidified our understanding of the Scene Management system taught in class. We learned that creating separate scenes (Menu and Gameplay) is not enough; they must be correctly added to the Build Settings list. This practical experience reinforced the lecture notes regarding the game loop and build process.

Script Communication and Code Adaptation: We practiced making different scripts communicate, such as linking the Coin script to the GameManager. While following the logic from the course notes, we noticed that some commands like FindObjectOfType gave warnings in Unity 6. We learned to adapt by researching documentation and using the updated FindFirstObjectByType command, ensuring our code followed the course logic while meeting modern standards.

UI System Implementation: We applied the UI concepts (Canvas, Panels, Buttons) to create a user friendly interface. We learned to control UI visibility dynamically using code (SetActive), allowing us to create functional Win/Loss screens as discussed in the game mechanics lectures.

Debugging and Problem Solving: We improved our ability to troubleshoot logic errors using Debug.Log, a technique emphasized in the labs. We also faced a specific issue where the camera disappeared when the player died ("No Cameras Rendering"). We solved this by un-parenting the camera before destroying the player object, teaching us the importance of hierarchy management.

Physics and Component Logic: We gained hands-on experience with Rigidbody and Colliders. We learned the practical difference between OnCollisionEnter (for solid obstacles) and OnTriggerEnter (for collectibles), effectively applying the physics interactions covered in the course.