# Programming Languages Project

Fahrettin ÖZKAN 64210002

Tolga YAŞAR 64210006

Erdem SÖNMEZ 64210011

Betül BAŞAR 64220043

Efe UĞUR 64210039

Mustafa Alperen ERÇEVİK 64220034

# Exploring and Implementing a Modern Programming Language: C# with Unity

## Introduction to the Language and Its History

C# is a versatile and powerful object-oriented programming language developed by Microsoft in 2000 as part of the .NET framework. Initially created to compete with Java, C# has evolved to become a cornerstone of software and game development. Designed with simplicity, efficiency, and versatility in mind, C# provides a robust environment for creating scalable applications.  
  
In 2005, Unity Technologies adopted C# as the primary language for scripting in its Unity game engine. This decision transformed C# into one of the most popular languages for game development. Unity’s integration with C# allows developers to create cross-platform games and applications efficiently, making it a staple in modern software development for gaming.

## Core Features of C# in Unity

1. Object-Oriented Programming (OOP):  
 - C# promotes modular and reusable code design through encapsulation, inheritance, and polymorphism.  
  
2. Garbage Collection:  
 - Automated memory management prevents common issues like memory leaks.  
  
3. Rich API and Libraries:  
 - C# provides a wide range of built-in libraries for graphics, physics, and game mechanics in Unity.  
  
4. Cross-Platform Development:  
 - Write once, deploy anywhere: Unity games scripted in C# can run on Windows, macOS, Android, iOS, and more.  
  
5. Concurrency Support:  
 - Asynchronous programming and threading capabilities enable efficient task execution.

## Practical Implementation: 3D Game Development

### Overview of the Game

- Objective: Create a 3D survival arena game where the player must avoid falling off the platform while eliminating dynamically spawning enemies.  
- Key Gameplay Elements:  
 - Player movement and interaction.  
 - Enemy AI behavior.  
 - Camera control for dynamic player following.  
 - Level progression and game reset functionality.

### Core Scripts and Features

#### 1. Player Movement and Interaction

Script: PlayerController.cs  
- Implements smooth movement using WASD keys.  
- Applies physics-based repelling force during enemy collisions, enhancing gameplay realism.

void MovePlayer() {  
 float moveHorizontal = Input.GetAxis("HorizontalWASD");  
 float moveVertical = Input.GetAxis("VerticalWASD");  
  
 Vector3 forward = cameraTransform.forward;  
 Vector3 right = cameraTransform.right;  
  
 forward.y = 0;  
 right.y = 0;  
  
 forward.Normalize();  
 right.Normalize();  
  
 Vector3 movement = (forward \* moveVertical + right \* moveHorizontal).normalized;  
 rb.AddForce(movement \* moveSpeed);  
}

#### 2. Camera Control

Script: CameraController.cs  
- Smoothly follows the player.  
- Allows rotation using arrow keys for enhanced situational awareness.

void RotateCamera() {  
 if (Input.GetKey(KeyCode.LeftArrow)) {  
 currentRotationAngle -= rotationSpeed \* Time.deltaTime;  
 }  
 if (Input.GetKey(KeyCode.RightArrow)) {  
 currentRotationAngle += rotationSpeed \* Time.deltaTime;  
 }  
}

#### 3. Enemy AI Behavior

Script: EnemyController.cs  
- Enemies dynamically spawn and move towards the player.  
- Despawns enemies if they fall off the platform.

void MoveTowardsPlayer() {  
 if (player == null) return;  
  
 Vector3 direction = (player.position - transform.position).normalized;  
 rb.AddForce(direction \* moveSpeed, ForceMode.Force);  
}

#### 4. Game Management

Script: GameManager.cs  
- Handles enemy spawning, level progression, and game reset.  
- Checks win/lose conditions dynamically.

void StartLevel(int level) {  
 enemyCount = level;  
 for (int i = 0; i < enemyCount; i++) {  
 SpawnEnemy();  
 }  
}  
  
void CheckWinCondition() {  
 if (GameObject.FindGameObjectsWithTag("Enemy").Length == 0) {  
 LevelUp();  
 }  
}

## Comparison with Other Languages

### C# vs. Python

- Syntax: C# is statically typed, ensuring type safety and reducing runtime errors. Python, being dynamically typed, is more flexible but prone to runtime issues.  
- Performance: C# executes faster due to its compiled nature, whereas Python’s interpreted nature makes it slower for resource-intensive tasks.  
- Ecosystem: While Python excels in data science and AI, C# dominates game development with Unity.

### C# vs. JavaScript

- Application Scope: JavaScript is widely used for web development, while C# is preferred for robust game development and enterprise applications.  
- Syntax and Paradigms: Both support OOP, but C# offers more structured and comprehensive support for advanced features like multithreading.

## Strengths and Weaknesses of C#

### Strengths

1. Ideal for Game Development: Seamless integration with Unity makes it the top choice for game developers.  
2. Comprehensive Libraries: A vast library ecosystem supports diverse development needs.  
3. Efficient Memory Management: Automated garbage collection prevents memory leaks.

### Weaknesses

1. Steep Learning Curve: Requires familiarity with OOP principles and a strong programming foundation.  
2. Platform Dependency: While cross-platform, some features are heavily tied to the .NET framework.

## Conclusion

C# stands out as a powerful and versatile programming language, particularly in the domain of game development with Unity. This project demonstrated its capabilities in handling complex game logic, physics, and player interactions. By leveraging its features, a dynamic and interactive 3D game was developed, showcasing its strengths in modern software development.  
  
Through this implementation, C# proved its reliability and scalability, affirming its relevance in contemporary game and software development. Its structured syntax, robust performance, and vast ecosystem make it a language worth mastering for developers aiming to excel in the gaming industry.