A PRELIMENERY REPORT ON

”PC Game Development using Unity”

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE
IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE

Of

BACHELOR OF COMPUTER ENGINEERING

SUBMITTED BY

Ankur Patil (BCOB10)
Lalu Nair (BCOB05)
Viren Patil (BCOB14)
Sharan Thakur (BCOB09)

DEPARTMENT OF COMPUTER ENGINEERING

DR. D. Y. PATIL INSTITUTE OF TECHNOLOGY
PIMPRI, PUNE 411018

SAVITRIBAI PHULE PUNE UNIVERSITY
2022-2023

CERTIFICATE

This is to certify that the Project Entitled

PC Game Development using Unity

Submitted by

Ankur Patil (BCOB10)
Lalu Nair (BCOB05)
Viren Patil (BCOB14)
Sharan Thakur (BCOB09)

are bonafide students of Dr. D. Y. Patil Institute of Technology and the work has been carried out by them under the supervision of Mr. Sharad Adsure (Asst. Professor) and Mrs. Deepika Jaiswal (Asst. Professor), it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of Bachelor of Computer Engineering.

Mr. Sharad Adsure
(Internal Guide)

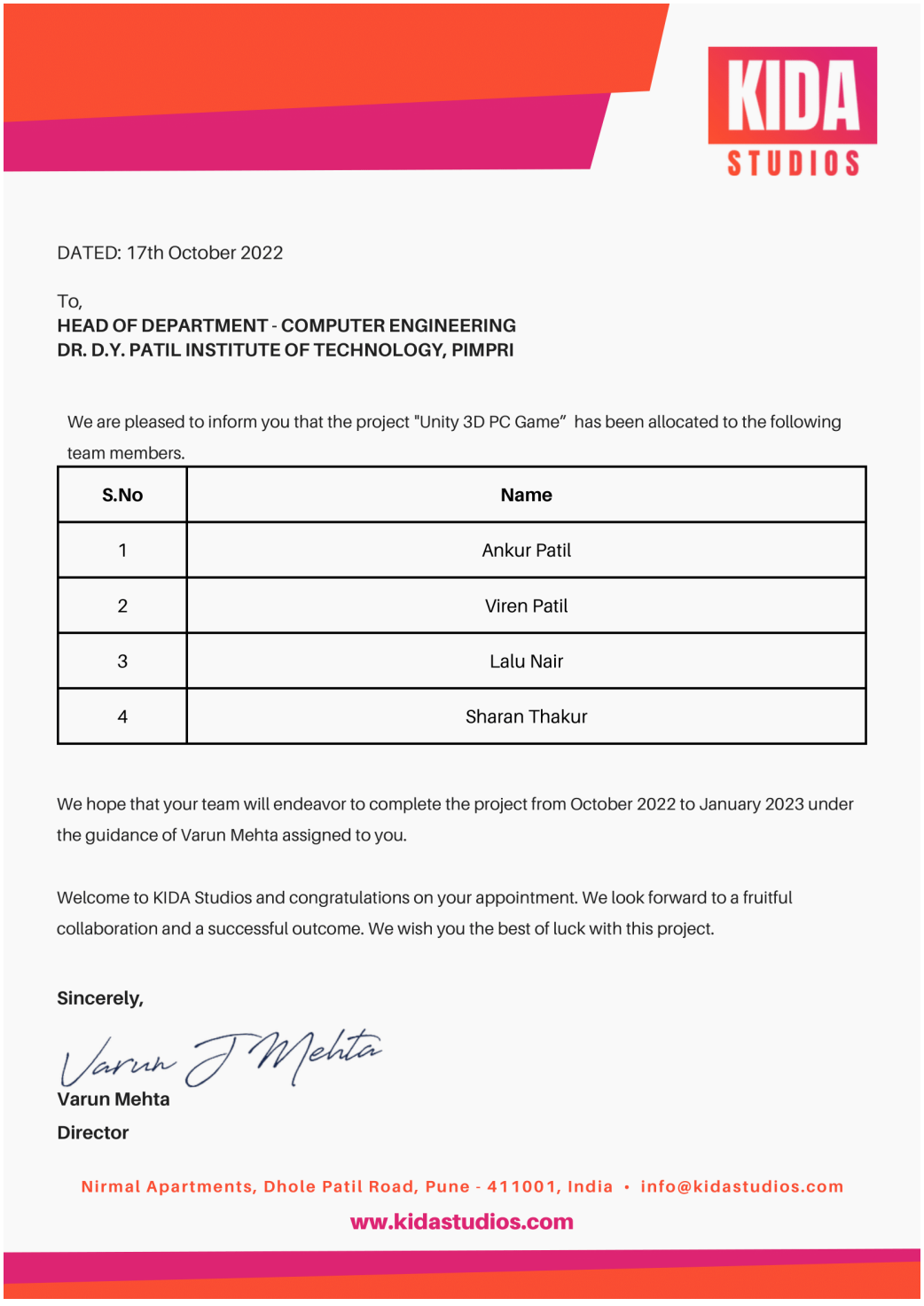
Dr.Vinod Kimbahune
(Head of the Department)

External Examiner
.

Mrs.Sunita Patil
(Project Coordinator)

Dr. Lalit Kumar Wadhwa
Principal,
Dr. D. Y. Patil Institute of Technology,
Pimpri, Pune – 411018

Date:



KIDA STUDIOS

DATED: 17th October 2022

To,
HEAD OF DEPARTMENT - COMPUTER ENGINEERING
DR. D.Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI

We are pleased to inform you that the project "Unity 3D PC Game" has been allocated to the following team members.

S.No	Name
1	Ankur Patil
2	Viren Patil
3	Lalu Nair
4	Sharan Thakur

We hope that your team will endeavor to complete the project from October 2022 to January 2023 under the guidance of Varun Mehta assigned to you.

Welcome to KIDA Studios and congratulations on your appointment. We look forward to a fruitful collaboration and a successful outcome. We wish you the best of luck with this project.

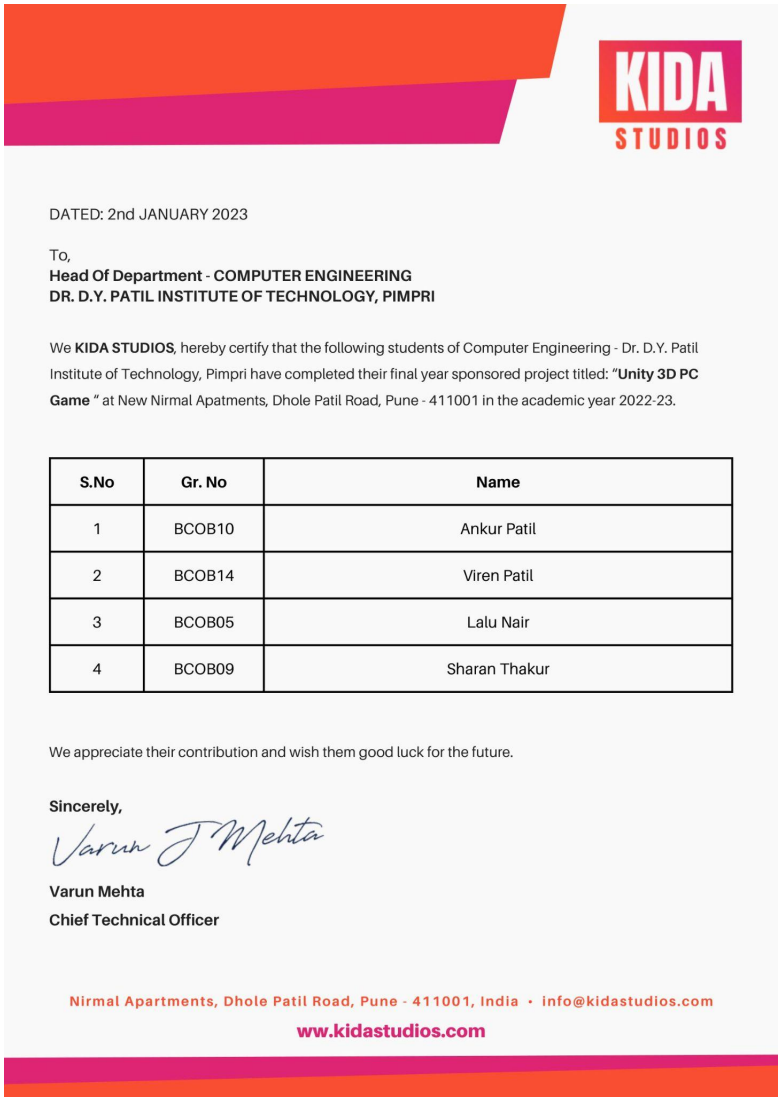
Sincerely,

Varun J Mehta

Varun Mehta
Director

Nirmal Apartments, Dhole Patil Road, Pune - 411001, India • info@kidastudios.com
ww.kidastudios.com

Figure 1: Sponsorship Allocation Letter



KIDA STUDIOS

DATED: 2nd JANUARY 2023

To,
Head Of Department - COMPUTER ENGINEERING
DR. D.Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI

We **KIDA STUDIOS**, hereby certify that the following students of Computer Engineering - Dr. D.Y. Patil Institute of Technology, Pimpri have completed their final year sponsored project titled: **"Unity 3D PC Game "** at New Nirmal Apatments, Dhole Patil Road, Pune - 411001 in the academic year 2022-23.

S.No	Gr. No	Name
1	BCOB10	Ankur Patil
2	BCOB14	Viren Patil
3	BCOB05	Lalu Nair
4	BCOB09	Sharan Thakur

We appreciate their contribution and wish them good luck for the future.

Sincerely,

Varun J Mehta

Varun Mehta
Chief Technical Officer

Nirmal Apartments, Dhole Patil Road, Pune - 411001, India • info@kidastudios.com
ww.kidastudios.com

Figure 2: Internship Completion Letter

ACKNOWLEDGEMENT

It gives us great pleasure in presenting the preliminary project report on “PC GAME DEVELOPMENT USING UNITY”

We would like to take this opportunity to thank my internal guides Asst. Prof. Sharad Adsure as well as Ms. Deepika Jaiswal for giving me all the help and guidance we needed. We are really grateful to them for their kind support. Their valuable suggestions were very helpful.

We are also grateful to Dr. Lalit Kumar Wadhwa, Principal as well as Dr. Vinod V. Kimbahune, Head of Computer Engineering Department, Dr. D.Y Patil Institute Of Technology, Pimpri for his indispensable support, suggestions.

Ankur Patil (BCOB10)

Lalu Nair (BCOB05)

Viren Patil (BCOB14)

Sharan Thakur (BCOB09)

ABSTRACT

The 2D (Dodgeball Video Game) is a multiplayer game written in Unity 3D using C#. The game allows players to play over a network, locally with friends on the same network, or over the internet via Photon Cloud. The game plays as you would expect from a dodgeball game. Players can run around, pick up balls, and throw them at each other. If the player is hit enough times, the player is eliminated. Players can choose to continue the game or stop at the end of the game. This game is made with assets and different packages that users can enjoy! This ”dodgeball video game” aims to appeal to a market that is still under-tapped. There are already several examples of dodgeball games on the market, but none are perfect and each has some compromises. Some with great interaction with the field, but leave room for variables such as ball/player skill, etc. Proper implementation will keep our users intrigued and provide a great and unique gaming experience Our main desire is to analyze the existing small market and build on previous dodgeball game attempts to develop a never-before-seen and unique dodgeball experience. Being able to reproduce the game and take advantage of the technology here today will help produce new life into the game dodgeball. Although it will not be 100 percent similar to the original game we all played as kids, the same basic principle of working together as a team to eliminate the enemy team is still present.

KEYWORDS : Unity 2D, Game Development, Mono Behaviour, C#, Real–World, Object Oriented Programming, WEBGL, iOS

Contents

1 INTRODUCTION 1

1.1 OVERVIEW 1

1.2 Motivation 2

1.3 Problem Definition 2

1.4 Project Scope And Limitations 3

1.5 Methodologies of Problem solving 4

2 LITERATURE SURVEY 7

3 SOFTWARE REQUIREMENT SPECIFICATION 10

3.1 Assumptions and Dependencies 10

3.2 FUNCTIONAL REQUIREMENT 11

3.3 EXTERNAL INTERFACE REQUIREMENT 11

3.4 NON-FUNCTIONAL REQUIREMENT 18

3.5 SYSTEM REQUIREMENTS 20

3.6 ANALYSIS MODELS: SDLC MODEL TO BE APPLIED 25

4 SYSTEM DESIGN 28

4.1 SYSTEM ARCHITECTURE 28

4.2 UML DIAGRAMS 28

4.3 Class Diagram 28

4.4 Activity Diagram 28

4.5 Use Case Diagram 29

5 DIAGRAMS 33

5.1 Data Flow Diagram 33

5.2 UML DIAGRAMS 35

5.3 Sequence Diagram 36

5.4 Entity Relationship Diagrams 41

6 PROJECT PLAN 42

6.1 Project Estimate 42

6.2 RISK MANAGEMENT 43

6.3 PROJECT SCHEDULE 45

6.4 Team Organization 46

7 RESULT 49

7.1 ScreenShots 49

8 TESTING 53

8.1 Introduction 53

8.2 Implementation 53

8.3 Objective 54

8.4 Testing Strategy 54

8.5 Types of Testing: 54

8.6 Unit Testing 55

8.7 Integrated system 56

8.8 Functional test 56

9 Test Cases 58

10 CONCLUSION 63

10.1 APPLICATIONS 64

List of Figures

1 Sponsorship Allocation Letter 3

2 Internship Completion Letter 4

3 SDLC MODEL 25

4 System Architecture 28

5 Class Diagram 29

6 Activity Diagram 30

7 Use Case Diagram 31

8 Data Flow diagram-0 33

9 Data Flow diagram-1 34

10 Data Flow diagram-2 34

11 Use case Diagram 37

12 Activity Diagram 38

13 Sequence Diagram 39

14 Class Diagram 40

1 INTRODUCTION

1.1 OVERVIEW

This engineering project report details the development of ”The Infinite Pleasure”, a multiplayer dodgeball video game created using Unity and the Photon Unity extension. The game offers an immersive and interactive experience for players, allowing them to choose their character, map, and compete with friends on the same local network.

The game follows traditional dodgeball rules, with players picking up balls, running around, and throwing them at each other. The game employs a server-client relationship to handle essential concepts such as rendering, player data, and network connectivity.

”The Infinite Pleasure” offers a unique twist to the classic dodgeball game, with innovative gameplay mechanics that would be impossible to replicate in real life. Players are placed into teams, and their objective is to catch, dodge, and launch the ball into the air to eliminate the opposing team.

The game’s matchmaking system ensures that players with similar or slightly higher experience levels are paired together. This report outlines the development process, including the game’s mechanics design, network architecture, and matchmaking system.

Overall, ”DodgeBall” combines the fun and chaotic gameplay of dodgeball .With its multiplayer features and Unity integration, it offers an enjoyable and competitive gaming experience for players to engage in virtual doge-themed dodgeball matches.

1.2 Motivation

- We are living at the cusp of modern technology with pocket computers with us (our smartphones), in our leisure time, all of us reach into our smartphones to play the next big game to pass time.
- Being the students we are and wanting to play the latest and greatest games; which drives us to build a unique mechanic for the multiplayer systems.
- A game is much more than just its software. It has to provide a much more enjoyable experience.
- Not enough good dodgeball games free-to-play.

1.3 Problem Definition

The lack of engaging and interactive multiplayer games for PC users has resulted in a gap in the market and limited options for entertainment. The need for a fun and challenging multiplayer game that can be played on PC has been identified, with the goal of providing an enjoyable and unique gaming experience for users.

In this problem definition, the identified issue is the lack of engaging multiplayer games for PC users, which presents an opportunity to develop a new game that can fill this gap in the market. This sets the stage for the project's objectives, such as developing a multiplayer dodgeball game using the Unity Game Engine for PC, to address this problem and provide a new and enjoyable gaming experience for users.

1.4 Project Scope And Limitations

Project Scope

The Project Scope for our multiplayer dodgeball game using the Unity Game Engine for PC includes the following features that will be implemented in the future to enhance the game's overall effectiveness, efficiency, and success:

i. Porting the game to phones (iOS A Android) and deploying it on App Store and Play Store to reach a wider audience.

ii. Implementing a health system as a mode for a match, which will add a new layer of complexity to the game and make it more challenging for players.

iii. Adding power-ups for players to choose from, which will give players temporary advantages over their opponents, adding a new strategic element to the game.

iv. Adding player-specific buffs that can be purchased, which will allow players to customize their characters and give them an advantage in the game.

v. Adding In-App Purchasing using Unity IAP features, which will allow players to purchase additional features or upgrades within the game, generating additional revenue for the project.

vi. Enhancing gamepad integrations, which will make the game more accessible to players who prefer using gamepads over keyboard and mouse controls.

These features are not included in the current project scope but are essential for the game's future success. The project team will carefully consider each feature's feasibility, resource requirements, and potential impact on the project's timeline, budget, and deliverables before implementing them. The project team will also consult with key stakeholders to ensure that the new features align with the project's overall goals and objectives.

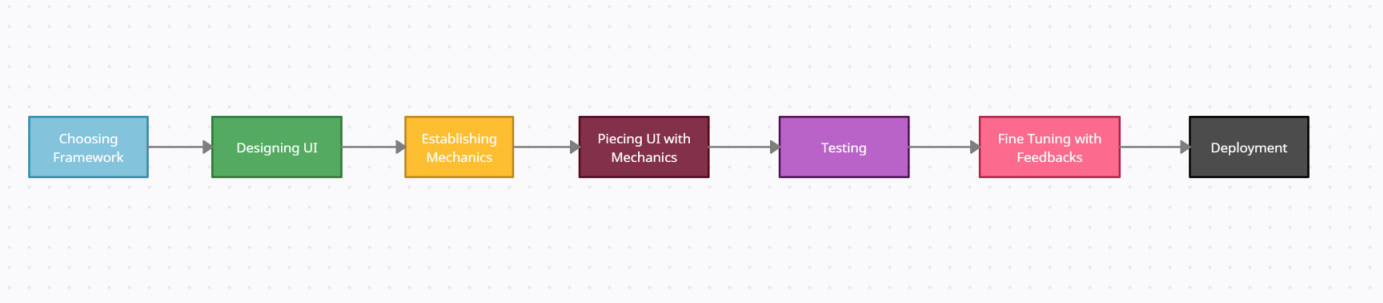
Limitations

1. Releasing To Users:- It can be tricky to figure out when and where to deliver your learning content to your employees without them feeling pressured into it or overwhelmed. You want your employees to grow personally and professionally but still perform their day-to-day tasks with accuracy. Be aware of the audience's schedule when you initially release the game. Don't try to build hype about it during a busy time, such as year-end or just before a conference.

2. Circumventing Resources:- There are a few options for making serious games cost-efficient. You can research low-cost or free platforms that offer a game you might adapt to your needs. The trouble with this is that off-the-shelf games might be hard to customize. You might not be able to integrate all the details of your learning strategy. You can also create your own game from scratch. This implies a different kind of process.

1.5 Methodologies of Problem solving

The lack of diversity in this specific area of gaming is a simple one to solve. We will create a two dimension, fast paced, dodgeball game that is central around dodgeball. We will use a high-level game engine and modern day techniques to create a fully immersive product for use on personal computers.



1. Choosing Framework: The process of selecting a framework for our project was a crucial step, and we carefully evaluated different options before settling on Unity. We considered the ease of adoption, the quality of the documentation, and the level of community support for each engine. 2. Designing UI: In terms of UI design, we were fortunate to have mentors with a keen sense of aesthetics who helped us create an interface that was both intuitive and visually appealing. We took great care to ensure that the UI complemented the gameplay mechanics and enhanced the user experience.

3. Establishing Mechanics: Developing game mechanics that were accessible to players of all ages and skill levels was one of our primary goals. We spent a significant amount of time refining and simplifying the mechanics to ensure that they were easy to understand and provided a fun and engaging experience.

4. Piecing UI with Mechanics: The mechanics were made in isolation and the UI designed differently since, we had to piece them together that really tells us the real story of development, fun and challenging at the same time. Integrating the UI and mechanics was a challenging but rewarding process. We designed them separately, which required us to carefully consider how they would work together and make the necessary adjustments to ensure that they were seamlessly integrated.

5. Testing: Testing in gaming is a very strenuous job since there are many scopes to test with the available limited resources we have done Playtests, Network connection testing as well making sure there is no problem in mechanics and UI.Testing and quality assurance were critical components of our development process. We conducted extensive playtests, network connection testing, and comprehensive checks of mechanics and UI to ensure that the game was polished and free of bugs and glitches.

6. Fine Tuning with Feedbacks: The edge and boundary cases of the game had a lot of bugs which led to backtracking our mistakes and making sure it was a tight ship. We encountered several bugs during the development process that required backtracking and debugging. We listened to feedback from players and continuously refined the game to ensure an optimal experience.

7. Deployment: Deployment for a Unity cross platform has to be done across their own respective store so, for PC it is Steam or something for Apple their App Store and Android it is Google Play Store, since we are 4 students we chose to upload our game on itch.io for deployment and uploaded binaries for each platform, we chose itch.io since it is free and a lot of indie games are there already on it. Deploying a Unity cross-platform game requires uploading it to respective stores such as Steam for PC, Apple App Store for iOS, and Google Play Store for Android. As a team of four, we chose itch.io, a free platform that hosts many indie games, to upload our game. We uploaded binaries for each platform to itch.io.

2 LITERATURE SURVEY

Sr.No.	Title of Paper	Year	Author	Key Points
1.	Research on the 3D Game Scene Optimization of Mobile Phone Based on the Unity 3D Engine | IEEE Conference Publication	2011	Jiang Jie; Kuang Yang; Shen Haihui	This paper describes a scene optimization problem for mobile phones based on Unity 3d. As computer technology develops, it promotes the growth of many related industries. Mobile phones are very popular in our society, and the trend of more and more people playing 3D games on their mobile phones is becoming more and more evident.
2.	Developing a game application to encourage face-to-face local gaming experience | IEEE Conference Publication | IEEE Xplore	2016	Yediya Juan; Teuku Aulia Geumpana; Jude J.L. Martinez	The problem that this research project seeks to solve lies in the increasing use of mobile devices in society. According to a study conducted by Scott Campbell, assistant professor at Communication Studies, and his Nojin Kwak, associate professor at the same university, people are increasingly mobile as mobile devices have become an important aspect of their daily communication. We tend to spend a lot of time focusing on our devices. communication studies. As a result, people tend to behave the same way in face-to-face meetings.
3.	Developing MOBA games using the Unity game engine | IEEE Conference Publication | IEEE Xplore	2017	D. Polančec; I. Mekterović	This paper describes the implementation of a typical MOBA game prototype for the Windows platform on the popular Unity 5 game engine. Emphasis on using built-in Unity components in MOBA environments, developing additional behaviors using Unity's Scripting API for C#, and integrating third-party components such as network engines, 3D

				models, particle systems, etc. is placed. Available from the Unity Asset Store.
4.	SOUL: Simulation of Objects in Unity for Learning | IEEE Conference Publication	2019	Lavina Nagpal; Meghna Jaglan; Anuraj Kathait; Aakil Mathur; Abhishek Vichare	This document explores the area of creating user-friendly and highly interactive environments for e-learning.
5.	Computing Games: Bridging the Gap Between Search and Entertainment | IEEE Journals & Magazine	2021	Anggina Primanita; Mohd Nor Akmal Khalid; Hiroyuki Iida	The purpose of this research is to find the optimal difficulty level for game solvers, define entertainment indicators, and focus on linking game tree search and entertainment in different game environments.
6.	Game or Watch: The Effect of Interactivity on Arousal and Engagement in Video Game Media | IEEE Journals & Magazine | IEEE Xplore	2021	Joshua Juvrud; Gabriel Ansgariusson; Patrik Selleby; Magnus Johansson	This study examined arousal and engagement while playing video games among interactive gamers and passive viewers watching game recordings.

3 SOFTWARE REQUIREMENT SPECIFICATION

3.1 Assumptions and Dependencies

Assumptions:

- Players are familiar with the rules of dodgeball and understand the objective of the game.
- Players have basic knowledge of gaming controls and mechanics.
- Players have access to the necessary equipment, such as a computer or a gaming console, to play the game.
- The game is designed for multiplayer mode, and players can connect to the game server without any issues.
- The game mechanics are well-defined and allow players to perform all the necessary actions to play the game effectively.

Dependencies:

- Players are familiar with the rules of dodgeball and understand the objective of the game.
- The game depends on the underlying software infrastructure, such as the operating system, web server, and database server, to function correctly.
- The game depends on network connectivity to connect players to the game server and enable multiplayer mode.
- The game depends on the hardware capabilities of the player's device, such as the CPU, GPU, and RAM, to run smoothly and without any lag.

- The game depends on the availability of game assets, such as graphics, sound effects, and music, to create a rich and immersive gaming experience.
- The game's success depends on the availability of players to participate in multiplayer mode and engage with the game.

3.2 FUNCTIONAL REQUIREMENT

- After running the game, the UX view of the game will appear on the screen.
- User Experience which is used to explain all aspects of a person's experience with a system.
- The gamer can directly select "Start" from the "Main Menu" which contains a number of Scenes like Tutorial, Matchmaking, Settings, and Quit.
- During Tutorial and Matchmaking players can control the character and have features like pause and play and can change settings in mid-game.

3.3 EXTERNAL INTERFACE REQUIREMENT

- Maximum high regulation with minimum hardware.
- We may provide each player with their rank.
- Easy to operate.
- Minimum hardware requirements which are relevant for this game.
- Design the whole system in an efficient manner.

User Interface

Hardware Interfaces:

Mobile

Operating system	**Android**	**iOS**
Version	7 (API 26)+	11+
CPU	Vulkan capable ARM64	A10 Fusion and above
Graphics API	OpenGL ES 3.0+, Vulkan	Metal
Additional requirements	- 1GB+ RAM - Supported hardware devices must meet or exceed Google's Android Compatibility Definition (Version 9.0) limited to the following Device Types: 1. Handheld (Section 2.2) 2. Television (Section 2.3) 3. Tablets (Section 2.6) - Hardware must be running Android OS natively. Android within a container or emulator isn't supported. - For development: Android SDK (10/API 29), Android NDK (r21d) and OpenJDK, which are installed by default with Unity Hub.	For development and debugging: Mac computer running minimum macOS 10.12.6 and Xcode 9.4 or higher. For App Store submission: see Apple's submission guidelines for the required Xcode version.

Desktop

Operating system	**Windows**	**macOS**	**Linux**
Operating system version	Windows 10 and Windows 11	High Sierra 10.13+	Distro(s) with Linux Kernel 5+(Mesa 20)
CPU	x64 architecture with SSE2 instruction set support.	Apple Silicon, Intel x64 architecture with SSE2.	x64 architecture with SSE2 instruction set support.
Graphics API	DX11, DX12,Vulkan capable.	Metal capable Intel and AMD GPUs /Apple Silicon	OpenGL 3+, Vulkan capable.
Additional requirements	Hardware vendor officially supported drivers. For development: IL2CPP scripting back-end requires Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.	Apple officially supported drivers. For development: IL2CPP scripting back-end requires Xcode. Targeting Apple Silicon with IL2CPP scripting back-end requires macOS Catalina 10.15.4 and Xcode 12.2 or newer.	Gnome desktop environment running on top of X11 windowing system Other configuration and user environment as provided stock with the supported distribution (such as Kernel or Compositor) Nvidia and AMD GPUs using Nvidia official proprietary graphics driver or AMD Mesa graphics driver.
	For all operating systems, the Unity Player is supported on workstations, laptop or tablet form factors, running without emulation, container or compatibility layer.		

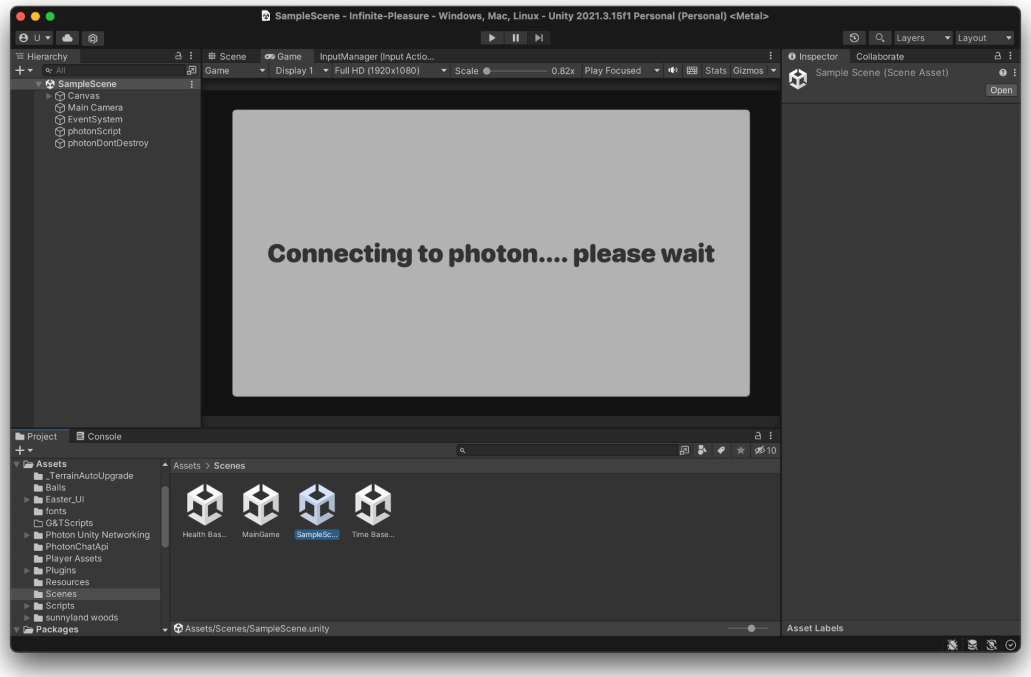
WebGL

Operating system running browsers	**Windows, macOS, and Linux**
Hardware	Workstation and laptop form factors.
Additional requirements	Versions of Chrome, Firefox or Safari that are: - WebGL 2.0 capable - HTML 5 standards compliant - 64-bit - WASM capable

Software Interfaces:

- Operating System: Windows/macOS/Linux
- Game Engine: Unity
- Programming Language: C#
- Build Systems : .NET, Mono, Gradle, Xcode

Unity: Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Worldwide Developers Conference as a Mac OS X game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console, and virtual reality platforms.

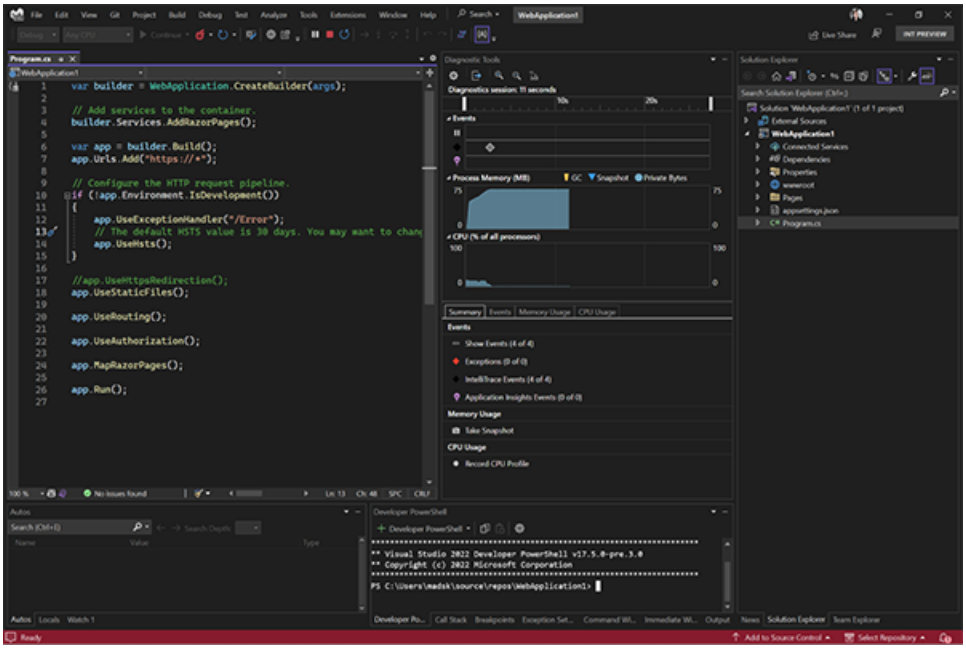


C-Sharp: C# is a general-purpose, high-level multi-paradigm programming language. C# encompasses static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented, and component-oriented programming disciplines. C-sharp (C#) is a popular programming language developed by Microsoft in 2002. It has also been a main language for Unity

game engine since 2005. Unity3D, being one of the two main obvious choices for AR/VR development, to get a handle of it if they want to develop applications with some complexity (think physics, animations, to design patterns, shaders or even sound effects).

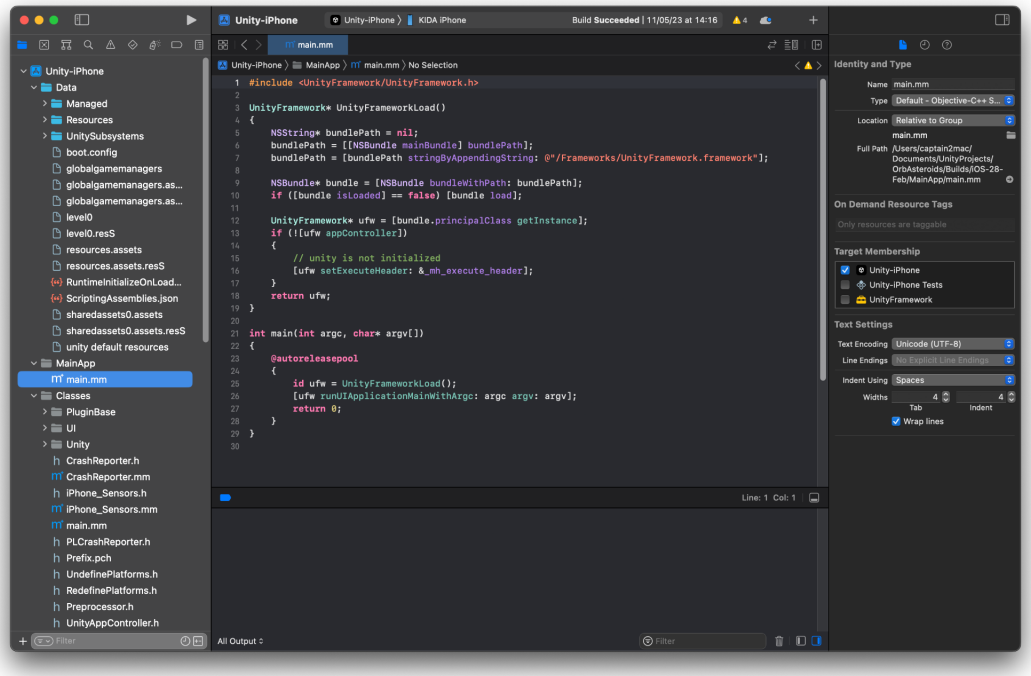
Rider: JetBrains Rider is a fast and powerful C# editor for Unity that runs on Windows, Mac, and Linux. With the unbeatable 2500+ smart code inspections and refactorings, Rider enhances your C# experience, letting you write error-proof code much faster.

Visual Studio: is an integrated development environment from Microsoft.The Unity engine integrates into one unparalleled platform to create 2D and 3D games and interactive content. Create once and publish to 21 platforms, including all mobile platforms, WebGL, Mac, PC and Linux desktop, web or consoles. Visual Studio brings powerful features to C# programmers. Write code quickly and with precision using IntelliSense. Navigate through your scripts easily and use powerful refactoring capabilities.



Xcode: Xcode is Apple's integrated development environment for macOS, used to develop software for macOS, iOS, iPadOS, watchOS, and tvOS. It was initially released in late 2003; the latest stable release is version 14.1, released on November 1, 2022, via the Mac App Store with macOS Monterey. The new multi platform target creates a single interface for use across iOS, iPadOS,

macOS, and tvOS. Your code is easier to maintain, and ready to be customized to take advantage of each platform's unique capabilities.



Gradle: Gradle is a build automation tool for multi-language software development. It controls the development process in the tasks of compilation and packaging to testing, deployment, and publishing. Supported languages include Java, C/C++.

Mono: The Mono scripting backend compiles code at runtime, with a technique called just-in-time compilation (JIT). Unity uses a fork of the open source Mono project. Some platforms don't support JIT compilation, so the Mono backend doesn't work on every platform. Other platforms support JIT and Mono but not ahead-of-time compilation (AOT), and so can't support the IL2CPP backend. When a platform can support both backends, Mono is the default.

3.4 NON-FUNCTIONAL REQUIREMENT

Performance Requirements

- The game will run on Android, MacOS, iOS, and Windows 10 (or newer) and requires the user to have the following components for Desktop: keyboard, mouse, monitor, and computer.
- The game will run with a CPU with SSE2 instruction set support and a Graphics card with DX10 (shader model 4.0) capabilities. Need a minimum of 4GB RAM, and 1GB of storage space.
- An Internet connection will also be required for multiplayer. The game will be running at a minimum of 1080p 60fps and a maximum of 1080p 120fps.

Safety Requirement

- Age appropriateness: Clearly define the target audience for your game and ensure that the content, themes, and challenges are suitable for that age group. Implement age verification mechanisms if necessary.
- User privacy: Respect user privacy by implementing appropriate data protection measures. Clearly communicate your data collection and usage policies, and obtain user consent when necessary. Follow applicable data protection regulations, such as the General Data Protection Regulation (GDPR) if your game is available to users in the European Union.
- Online interactions: If your game includes online multiplayer features or chat functionality, implement measures to prevent and handle inappropriate behavior, harassment, and cyber-bullying. Provide reporting and blocking mechanisms for users to address such issues.
- Parental controls: Consider implementing parental control features to allow parents or guardians to restrict access to certain content, set time limits, or manage in-game purchases for under-age players.

- Game content warnings: Provide appropriate content warnings for potentially sensitive or disturbing content, such as violence, horror, or mature themes. Allow players to customize content filters if necessary.
- Accessibility: Ensure your game is accessible to players with disabilities. Implement features such as colorblind modes, adjustable font sizes, captioning for audio content, and support for alternative input devices.
- Fair gameplay: Prevent cheating, hacking, or exploiting in the game to maintain a fair and enjoyable experience for all players. Implement appropriate security measures and address any vulnerabilities promptly.
- Compliance with regulations: Familiarize yourself with applicable laws, regulations, and industry standards related to game safety, including consumer protection laws, advertising standards, and intellectual property rights.
- Apple IDFA: The Identifier for Advertisers (IDFA) is a random device identifier assigned by Apple to a user's device. Advertisers use this to track data so they can deliver customized advertising. The IDFA is used for tracking and identifying a user (without revealing personal information).

Software Quality Attributes

- Usability: The game should be easy to navigate and play, with clear instructions and intuitive controls.
- Performance: The game should be responsive and run smoothly without any lag or glitches.
- Reliability: The game should function reliably and consistently without crashes or other technical issues.
- Maintainability: The game should be designed with maintainability in mind, with clear and well-organized code that is easy to update and modify.

- Security: The game should be secure, with measures in place to protect player data and prevent cheating.
- Scalability: The game should be able to handle a growing number of players and gameplay data without a significant decrease in performance.
- Compatibility: The game should be compatible with different devices, operating systems, and web browsers to reach a wider audience.
- Portability: The game should be easily portable between different platforms, such as mobile devices and desktop computers.
- Accessibility: The game should be designed to be accessible to players with different abilities, with features such as subtitles, audio descriptions, and color contrast options. By focusing on these software quality attributes, the 2D dodgeball game can provide a high-quality gameplay experience for players, increase engagement, and encourage repeat gameplay.

3.5 SYSTEM REQUIREMENTS

Software Requirements

1. Operating System: Windows/macOS/Linux

2. Game Engine: Unity

3. Programming Language: C#

Unity: Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Worldwide Developers Conference as a Mac OS X game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console, and virtual reality platforms. The technology that we refer to as IL2CPP has two distinct parts. An ahead-of-time (AOT) compiler A runtime library to support the virtual machine The AOT compiler translates Intermediate Language (IL), the low-level output from .NET compilers,

to C++ source code. The runtime library provides services and abstractions like a garbage collector, platform-independent access to threads and files, and implementations of internal calls (native code which modifies managed data structures directly). The IL2CPP AOT compiler is named il2cpp.exe. On Windows you can find it in the Editor

Data

il2cpp directory. On OSX it is in the Contents/Frameworks/il2cpp/build directory in the Unity installation. The il2cpp.exe utility is a managed executable, written entirely in C#. We compile it with both .NET and Mono compilers during our development of IL2CPP. The il2cpp.exe utility accepts managed assemblies compiled with the Mono compiler that ships with Unity and generates C++ code which we pass on to a platform-specific C++ compiler.

Hardware Requirements

The hardware requirements for a Unity dodgeball game will depend on the complexity of the game, the graphics and physics involved, and the target platform (e.g., PC, mobile, console). Here are some considerations when determining the hardware requirements:

Processor (CPU): Unity games typically rely on the CPU for gameplay and physics calculations. A multi-core processor with a higher clock speed will help handle complex calculations more efficiently.

Graphics Card (GPU): The GPU handles rendering and graphics-related tasks. A dedicated graphics card with decent performance will ensure smooth rendering and better visual quality. The specific requirements will depend on the level of graphics fidelity desired for the game. Here are the specific requirements for different platforms:

Mobile

Operating system	**Android**	**iOS**
Version	7 (API 26)+	11+
CPU	Vulkan capable ARM64	A10 Fusion and above
Graphics API	OpenGL ES 3.0+, Vulkan	Metal
Additional requirements	- 1GB+ RAM - Supported hardware devices must meet or exceed Google's Android Compatibility Definition (Version 9.0) limited to the following Device Types: 1. Handheld (Section 2.2) 2. Television (Section 2.3) 3. Tablets (Section 2.6) - Hardware must be running Android OS natively. Android within a container or emulator isn't supported. - For development: Android SDK (10/API 29), Android NDK (r21d) and OpenJDK, which are installed by default with Unity Hub.	For development and debugging: Mac computer running minimum macOS 10.12.6 and Xcode 9.4 or higher. For App Store submission: see Apple's submission guidelines for the required Xcode version.

Desktop

Operating system	**Windows**	**macOS**	**Linux**
Operating system version	Windows 10 and Windows 11	High Sierra 10.13+	Distro(s) with Linux Kernel 5+(Mesa 20)
CPU	x64 architecture with SSE2 instruction set support.	Apple Silicon, Intel x64 architecture with SSE2.	x64 architecture with SSE2 instruction set support.
Graphics API	DX11, DX12,Vulkan capable.	Metal capable Intel and AMD GPUs /Apple Silicon	OpenGL 3+, Vulkan capable.
Additional requirements	Hardware vendor officially supported drivers. For development: IL2CPP scripting back-end requires Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.	Apple officially supported drivers. For development: IL2CPP scripting back-end requires Xcode. Targeting Apple Silicon with IL2CPP scripting back-end requires macOS Catalina 10.15.4 and Xcode 12.2 or newer.	Gnome desktop environment running on top of X11 windowing system Other configuration and user environment as provided stock with the supported distribution (such as Kernel or Compositor) Nvidia and AMD GPUs using Nvidia official proprietary graphics driver or AMD Mesa graphics driver.
	For all operating systems, the Unity Player is supported on workstations, laptop or tablet form factors, running without emulation, container or compatibility layer.		

WebGL

Operating system running browsers	**Windows, macOS, and Linux**
Hardware	Workstation and laptop form factors.
Additional requirements	Versions of Chrome, Firefox or Safari that are: - WebGL 2.0 capable - HTML 5 standards compliant - 64-bit - WASM capable

3.6 ANALYSIS MODELS: SDLC MODEL TO BE APPLIED

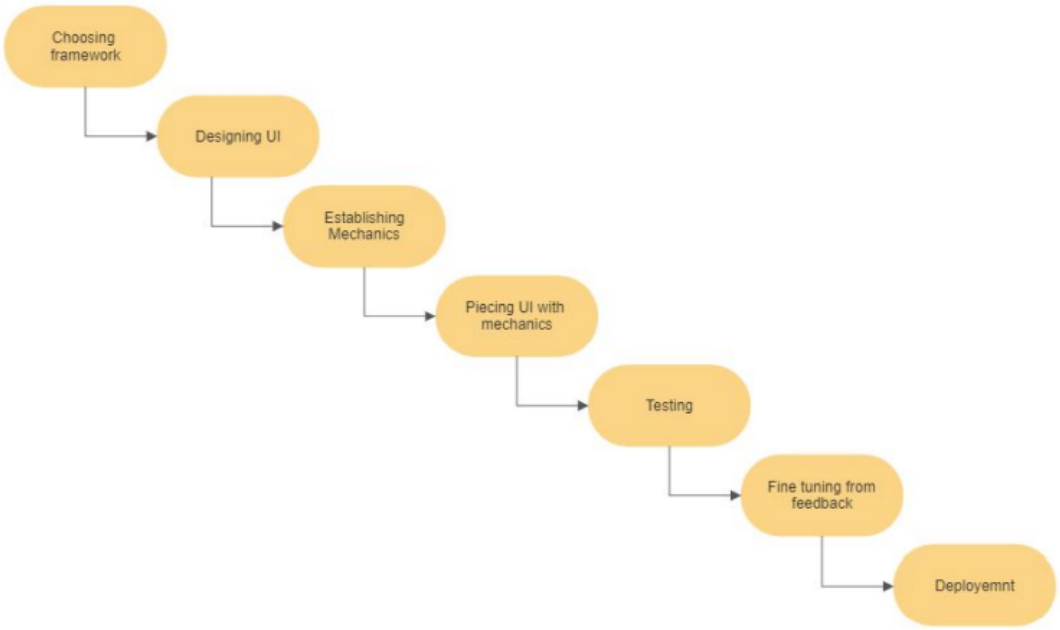


Figure 3: SDLC MODEL

SDLC Models stands for Software Development Life Cycle Models. There are several software development life cycle (SDLC) models that can be applied in game development. Let's take a look at a few examples:

Waterfall Model: The Waterfall Model is a linear sequential approach that involves a series of sequential phases. This model can be applied in game development by breaking down the game development process into different stages such as planning, design, implementation, testing, and maintenance. Each stage is completed before moving on to the next stage, and changes to previous stages are not allowed. This model is useful when the game requirements are well-defined and the project scope is clear.

Agile Model: The Agile Model is an iterative and incremental approach that involves continuous development and testing. This model can be applied in game development by breaking down the game development process into short development cycles called sprints. Each sprint involves designing, developing, testing, and integrating a small part of the game. The game is then tested and evaluated after each sprint, and changes are made accordingly. This model is useful when the game requirements are not well-defined, or when the project scope is subject to change.

Spiral Model: The Spiral Model is a risk-driven model that involves identifying and mitigating risks at each stage of the development process. This model can be applied in game development by breaking down the game development process into different stages, each involving risk analysis and mitigation. The game is then developed and tested, and the risks are reevaluated and mitigated in subsequent stages. This model is useful when the game requirements are not well-defined, or when the project scope is subject to change.

V-Model: The V-Model is a variation of the Waterfall Model that involves testing at each stage of the development process. This model can be applied in game development by breaking down the game development process into different stages, each involving testing and verification. The game is then developed and tested, and the testing results are used to verify and validate the game requirements. This model is useful when the game requirements are well-defined and the project scope is clear, but the testing and verification process is crucial.

Ours is: Planning: During the planning phase, the team identifies the objectives and requirements for the game. They define the game mechanics, art style, target audience, and development timeline.

Analysis: In the analysis phase, the team evaluates the feasibility of the project and identifies potential risks. They analyze the game mechanics, UI design, and overall user experience to ensure that they meet the requirements set during the planning phase.

Design: During the design phase, the team creates detailed design documents that outline the game mechanics, level designs, and UI layout. They create wireframes, storyboards, and mockups to visualize the game design.

Development: In the development phase, the team begins coding the game using Unity. They implement the game mechanics, UI, and art assets created during the design phase. The team also conducts regular testing to identify and fix any bugs that may arise.

Testing: In the testing phase, the team conducts comprehensive testing to ensure that the game is stable, functional, and meets the user requirements. They conduct unit testing, integration testing, and acceptance testing to ensure the game is ready for release.

Deployment: During the deployment phase, the team releases the game to the appropriate platforms such as itch.io or other app stores. They ensure that the game meets the platform requirements and fix any issues that may arise during deployment.

Maintenance: After the game is released, the team continues to monitor and maintain the game to ensure it remains stable, secure, and enjoyable for players. They may release patches and updates to address bugs, improve gameplay, and add new features based on user feedback.

4 SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

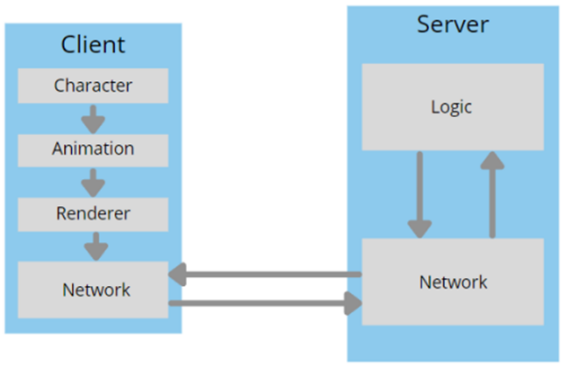


Figure 4: System Architecture

4.2 UML DIAGRAMS

4.3 Class Diagram

Class diagram describes the structure of a system by showing the system's classes, Their attributes, and the relationships among the classes. Proposed system contains five different types of classes and each posses their own attributes and methods. Main Classes of the proposed system are ND-SRRC, FP Tree, Apriory, Sanitised DB each have different functionalities.

4.4 Activity Diagram

An activity is particular operation of the system. An activity diagram is intended to represent stepwise work-flow of activities or actions that can take place in the system. It shows overall flow

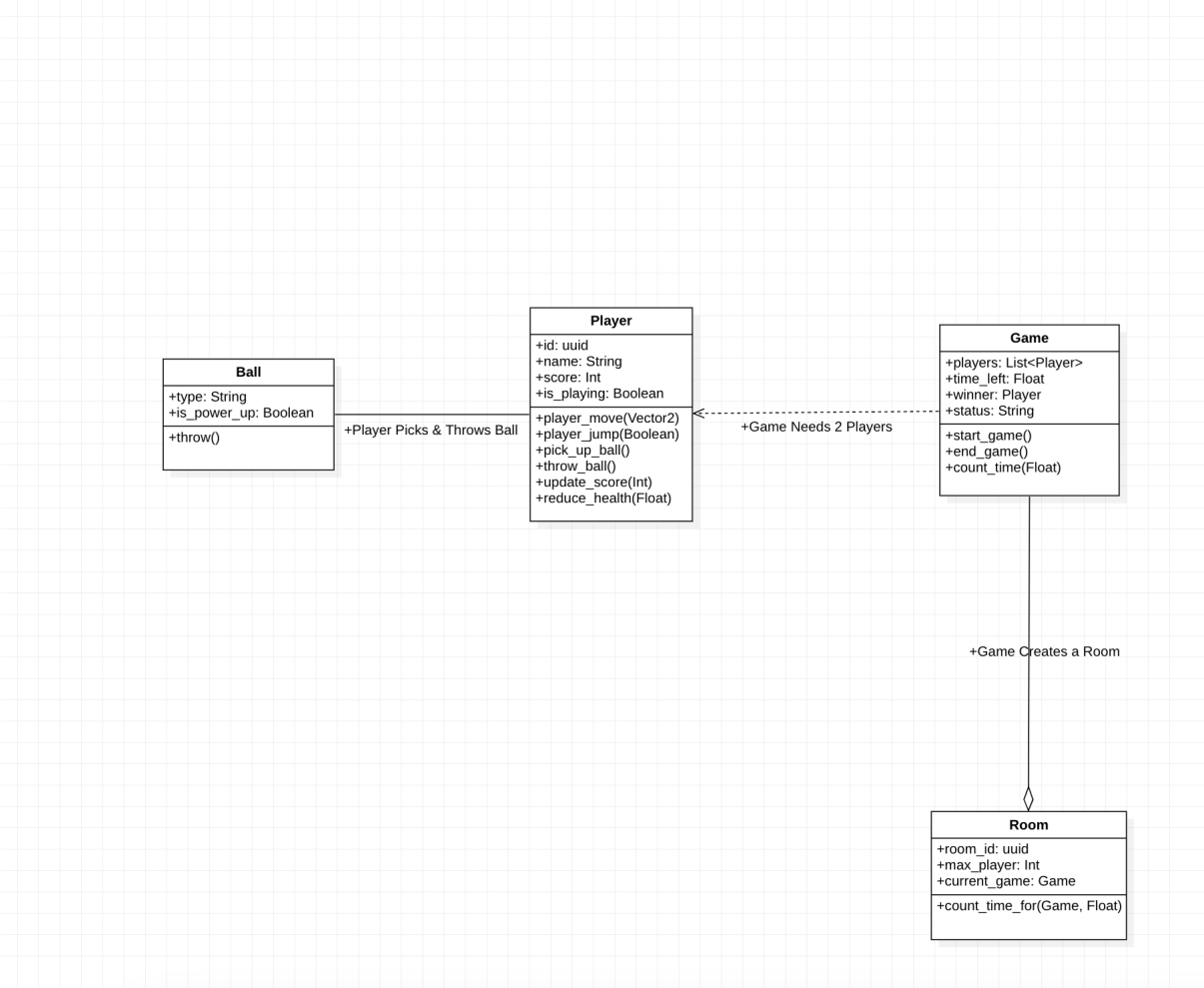


Figure 5: Class Diagram

of control and models computational and organizational processes. Activity diagrams are used to model dynamic aspects of the system.

4.5 Use Case Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases),

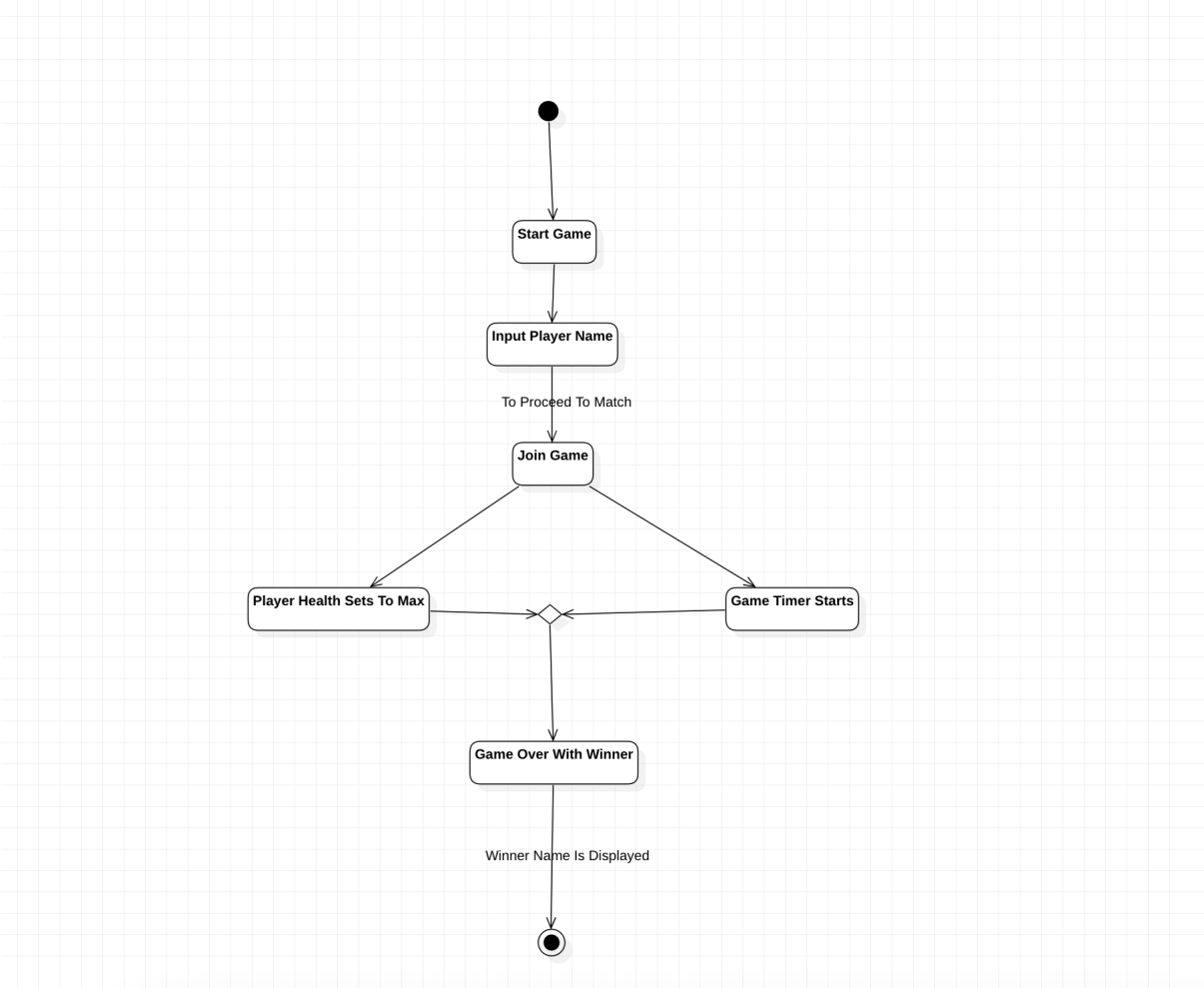


Figure 6: Activity Diagram

and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

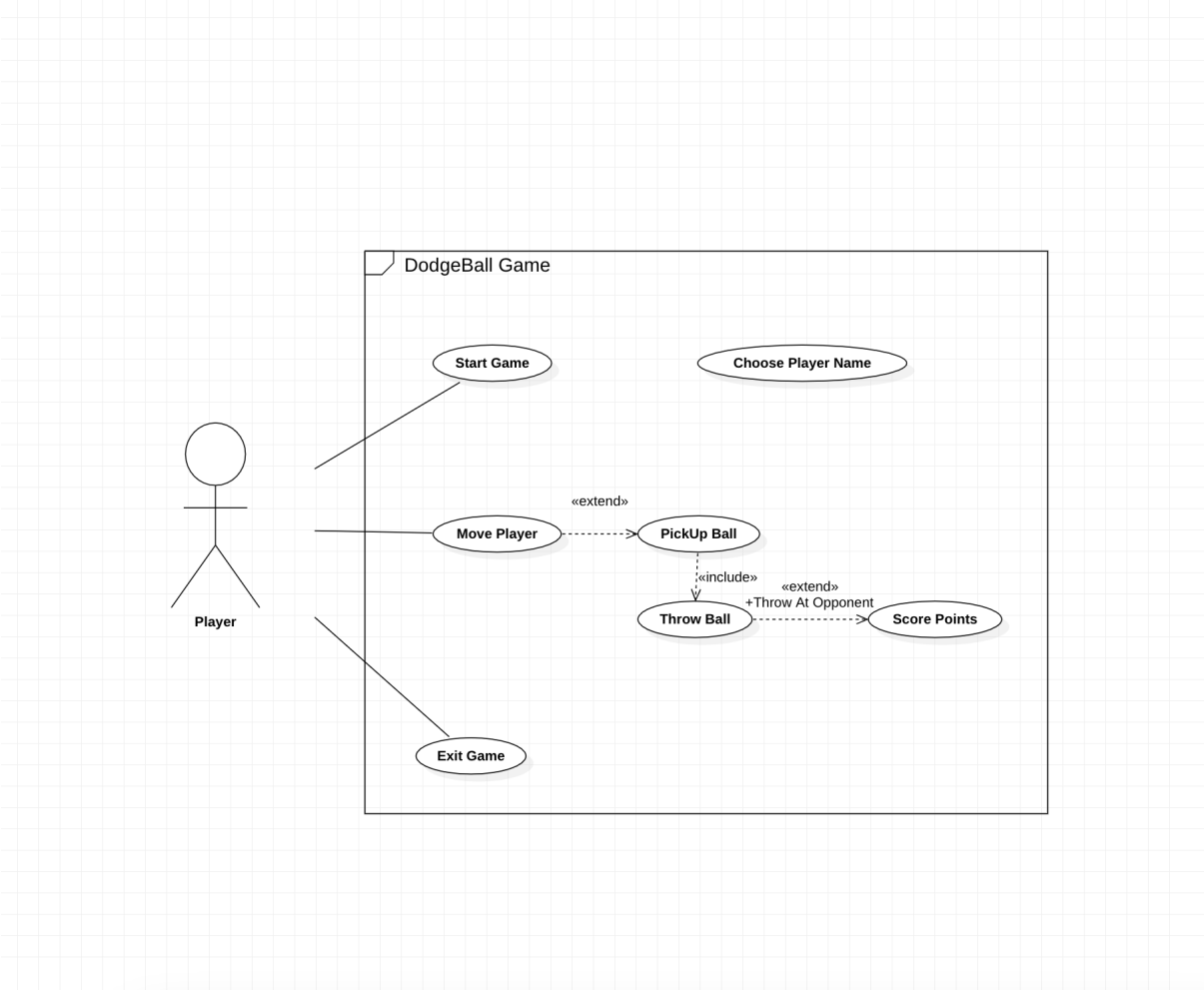


Figure 7: Use Case Diagram

MATHEMATICAL MODEL

1. Player positions: Let P1(x1, y1) and P2(x2, y2) represent the positions of Player 1 and Player 2 on a 2D plane.

2. Position and Velocity: Let B(x, y) represent the position of the ball on the 2D plane. Let Vx and Vy represent the ball's velocity components in the x and y directions.

3. Player movement: Players can move in the x and y directions. Let ΔRx1, ΔRy1, ΔRx2, and ΔRy2 represent the changes in position for Player 1 and Player 2, respectively. P1_new(x1 + ΔRx1, y1 + ΔRy1) and P2_new(x2 + ΔRx2, y2 + ΔRy2) represent the new positions of Player 1 and Player 2 after movement.

4. Ball movement: The ball's position changes based on its velocity: B_new(x + Vx * t, y + Vy * t), where t is the time elapsed since the last position update.

5. Throwing the ball: When a player throws the ball, the ball's velocity components (Vx, Vy) are updated based on the throw's direction and strength.

6. Collision detection: To determine if the ball hits a player, calculate the distance between the ball and each player using the Euclidean distance formula: d = sqrt((x2 - x1)$\hat{2}$ + (y2 - y1)$\hat{2}$). If the distance is less than or equal to a predefined threshold (e.g., the sum of the player's and ball's radii), the player is considered hit.

7. Scoring and game state: When a player is hit, the opposing player scores a point. The game continues until a predefined number of points are scored by one of the players, or a time limit is reached.

5 DIAGRAMS

5.1 Data Flow Diagram

In Data Flow Diagram, we Show that flow of data in our system in DFD0 we show that base DFD in which rectangle present input as well as output and circle show our system. In DFD1 we show actual input and actual output of system input of our system is text or image and output is rumor detected likewise in DFD 2 we presentoperation of user as well as admin.

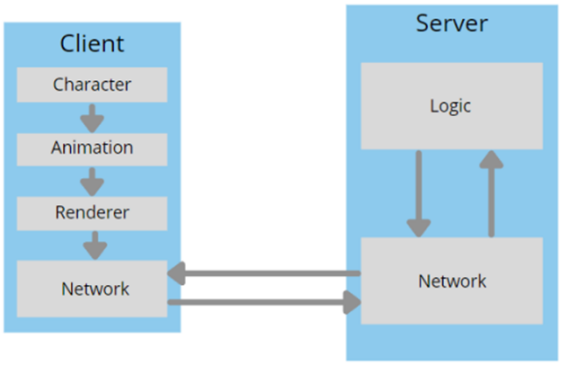


Figure 8: Data Flow diagram-0

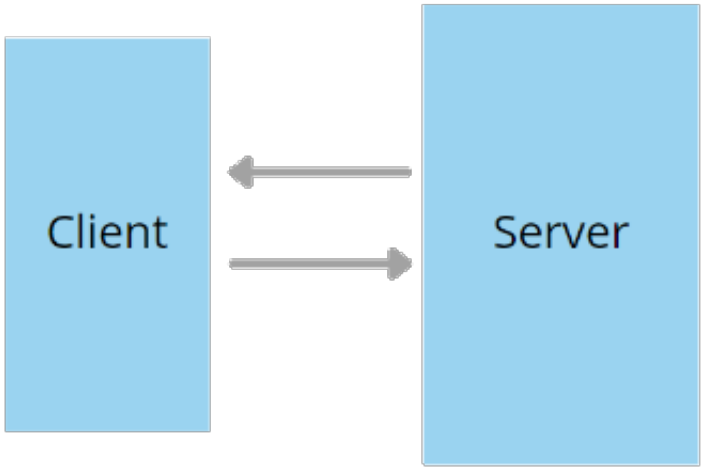


Figure 9: Data Flow diagram-1

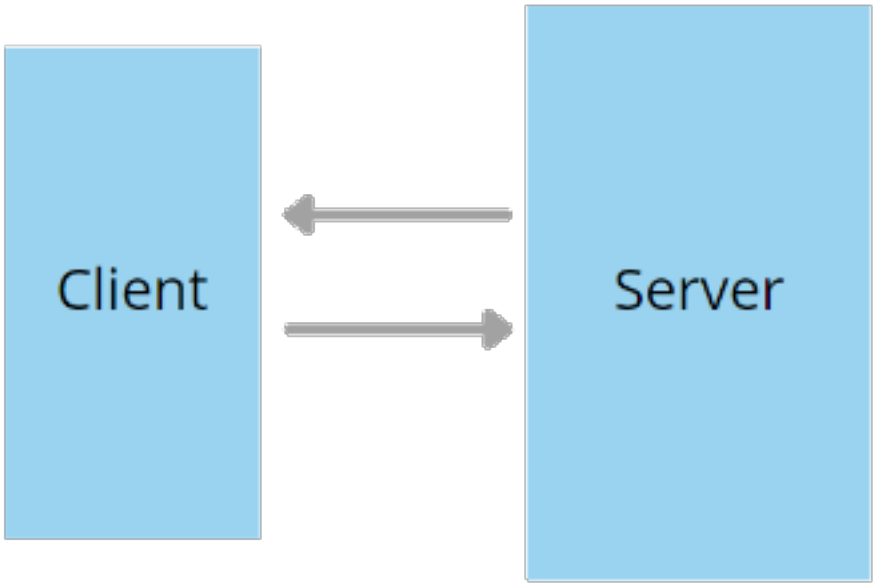


Figure 10: Data Flow diagram-2

5.2 UML DIAGRAMS

Unified Modeling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct and document the artifacts of a soft- ware intensive system. UML is process independent, although optimally it should be used in process that is use case driven, architecture centric, iterative and incremental. The Number of UML Diagram is available.

- Use case Diagram.
- Component Diagram.
- Activity Diagram
- Sequence Diagram.

5.3 Sequence Diagram

Sequence diagram shows how objects communicate with each other in terms of a sequence of messages. It also indicates the lifespans of objects relative to those messages.

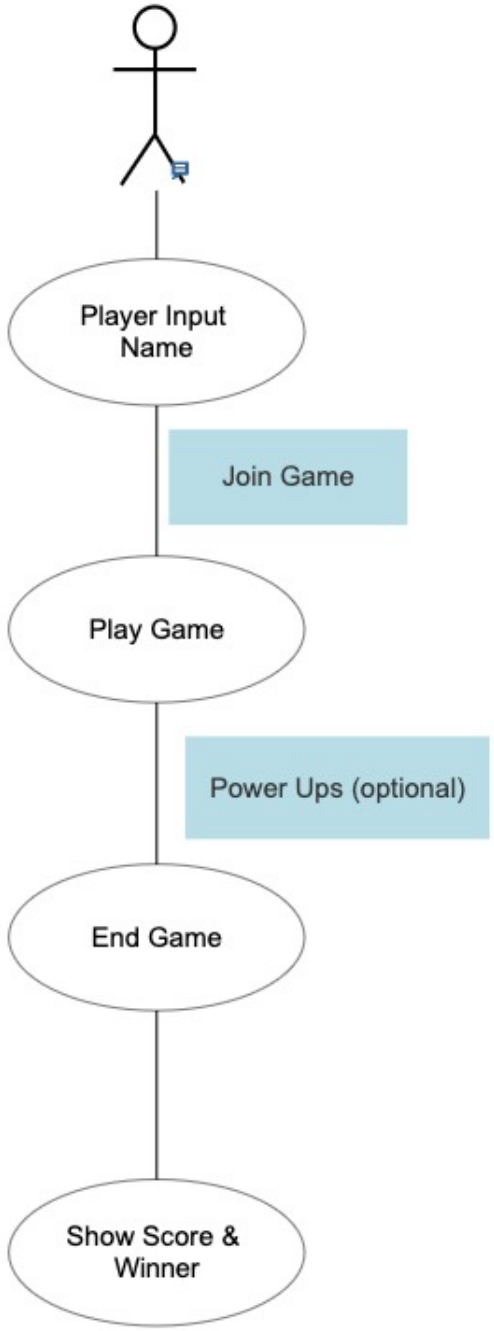
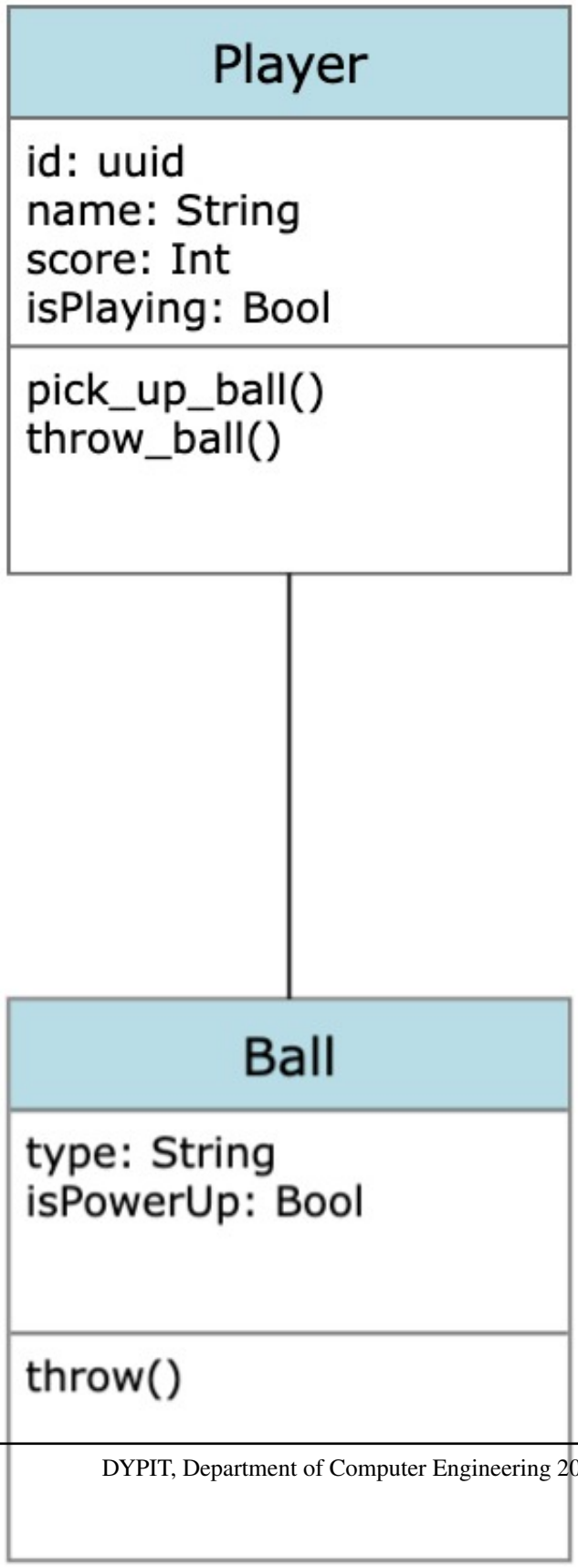
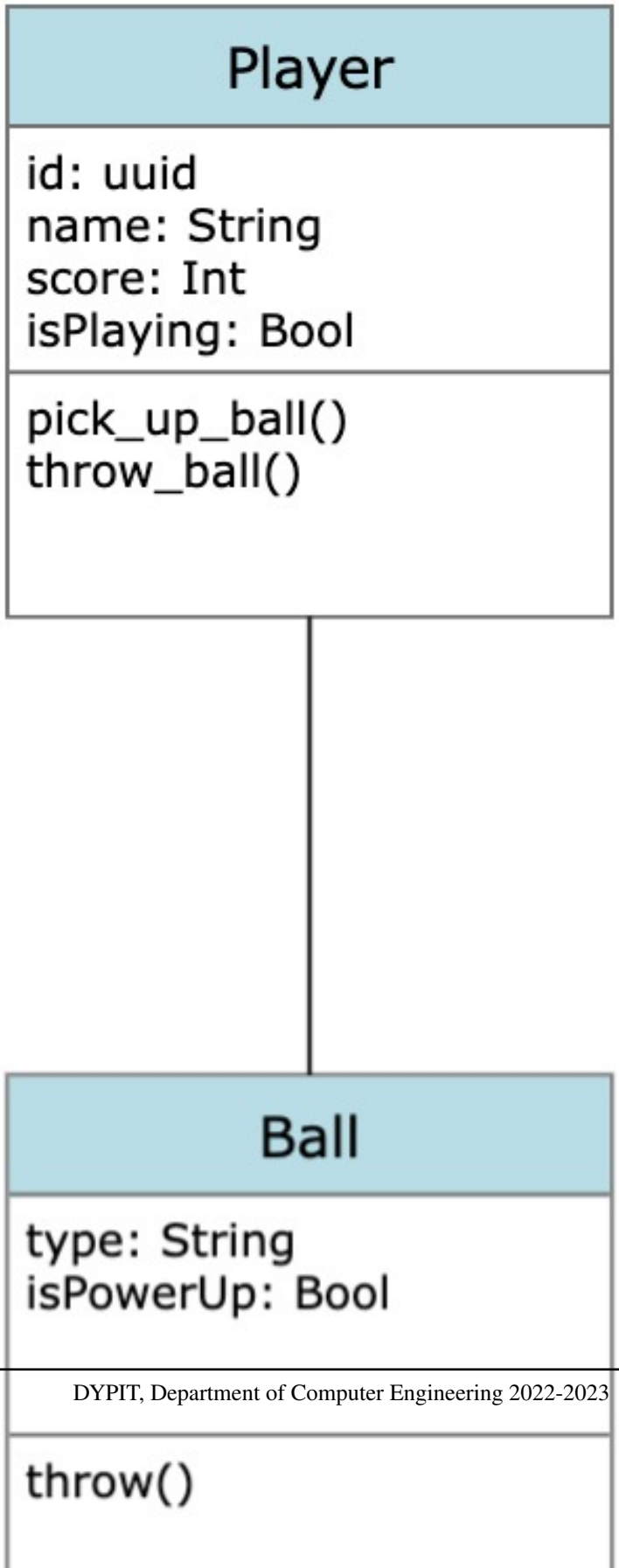


Figure 11: Use case Diagram


Player
id: uuid
name: String
score: Int
isPlaying: Bool
pick_up_ball()
throw_ball()
Ball
type: String
isPowerUp: Bool
throw()


Player
id: uuid
name: String
score: Int
isPlaying: Bool
pick_up_ball()
throw_ball()
Ball
type: String
isPowerUp: Bool
throw()

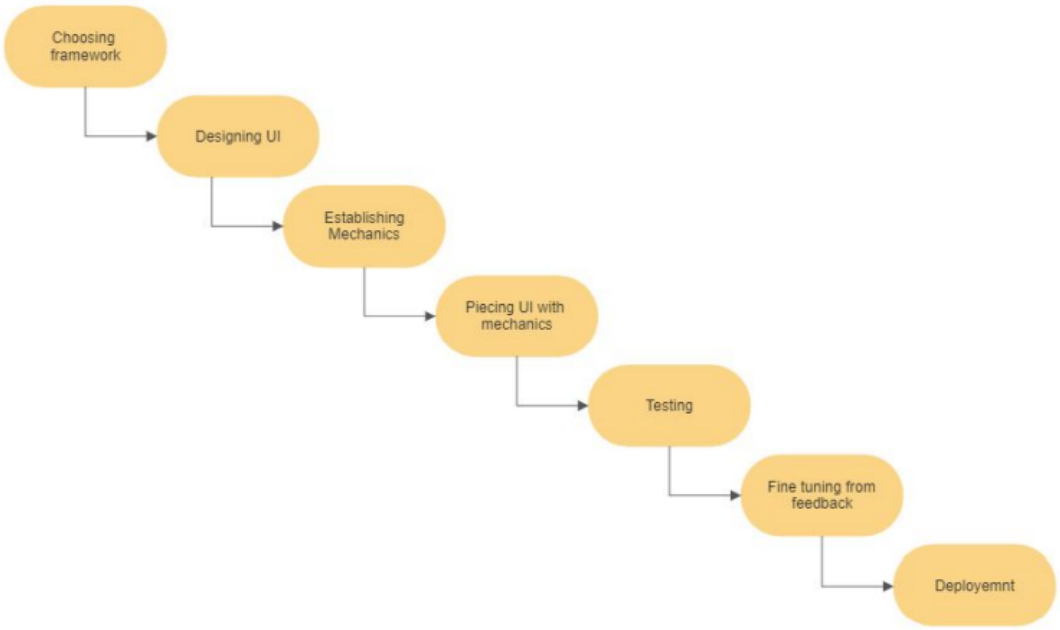


Figure 14: Class Diagram

5.4 Entity Relationship Diagrams

Class diagram describes the structure of a system by showing the system's classes, Their attributes, and the relationships among the classes. Proposed system contains five different types of classes and each posses their own attributes and methods. Main Classes of the proposed system are ND-SRRC, FP Tree, Apriory, Sanitised DB each have different functionalities.

6 PROJECT PLAN

6.1 Project Estimate

Hardware Cost

- Laptop Cost : Rs 2,35,001
- Controller: Rs 2,199

Laptop Used

- Asus ROG :
  - Processor: AMD Ryzen 9 5000 series
  - GPU: Nvidia RTX GeForce 3050
- MacBook Pro (M1)

Software Cost

- Unity License: Rs 1,80,000
- Apple Developer License: 99 Dollars
- Google Developer License:9 Dollars
- JetBrains Rider License: 349 Dollars

Other Cost

- Unity Asset
  - Animated Characters: 34.99 Dollars

- Photon Networking: 99 Dollars
- Royalty free: 1.99 Dollars

6.2 RISK MANAGEMENT

Risk Identification And Risk Analysis

Developing a dodgeball game using Unity can pose several risks, which the development team needs to manage to ensure project success. Here are some potential risks and their corresponding risk management strategies:

Technical Risks: There are risks associated with the technical aspects of developing a game, such as software and hardware failures, bugs, and compatibility issues. To manage these risks, the development team should implement a robust quality assurance and testing process, conduct regular code reviews, and maintain a backup plan in case of unforeseen technical issues. The team should also keep up-to-date with the latest Unity updates and patches to minimize the risk of compatibility issues.

Scope Creep: As the development process progresses, there may be a tendency to add additional features and functionality to the game, leading to scope creep. To manage this risk, the development team should establish a clear scope of work and requirements document at the outset of the project and strictly adhere to it. Any changes to the scope should be carefully evaluated and approved before implementation. The team should also maintain effective communication with stakeholders to ensure that their expectations are aligned with the project scope.

Resource Constraints: Developing a game requires significant resources, including time, money, and talent. To manage the risk of resource constraints, the development team should carefully evaluate available resources and allocate them efficiently. This may involve prioritizing tasks, outsourcing certain functions, or seeking additional funding or talent. The team should also maintain a contingency plan in case of unforeseen resource constraints.

User Acceptance: Ultimately, the success of a game depends on its acceptance by the end-users. To manage the risk of user acceptance, the development team should conduct market research to understand user preferences and expectations, engage in regular user testing and feedback, and implement changes based on user feedback. The team should also consider incorporating analytics tools to gather data on user engagement and behavior.

Legal Risks: Developing a game can also expose the development team to legal risks, such as copyright infringement, trademark violations, and data privacy breaches. To manage these risks, the development team should conduct regular legal reviews, adhere to industry standards and regulations, and maintain appropriate data protection measures. The team should also ensure that all game assets, including art, music, and sound effects, are legally acquired or licensed.

Overview of Risk Mitigation, Monitoring, Management

Risk Mitigation:

Technical Risks: To mitigate technical risks, the development team should establish a robust quality assurance and testing process, conduct regular code reviews, and maintain a backup plan in case of unforeseen technical issues. Scope Creep: To mitigate scope creep, the development team should establish a clear scope of work and requirements document at the outset of the project and strictly adhere to it. Any changes to the scope should be carefully evaluated and approved before implementation. Resource Constraints: To mitigate the risk of resource constraints, the development team should carefully evaluate available resources and allocate them efficiently. This may involve prioritizing tasks, outsourcing certain functions, or seeking additional funding or talent. User Acceptance: To mitigate the risk of user acceptance, the development team should conduct market research to understand user preferences and expectations, engage in regular user testing and feedback, and implement changes based on user feedback. Legal Risks: To mitigate legal risks, the development team should conduct regular legal reviews, adhere to industry standards and regulations, and maintain appropriate data protection measures.

Risk Monitoring:

The development team should regularly monitor risks and their potential impact on the project.

This may involve conducting regular risk assessments and updating the risk management plan accordingly.

Risk Management:

The development team should have a clear risk management plan in place that outlines the steps to be taken in the event of a risk materializing. This may involve contingency planning, risk transfer, or risk acceptance. Overall, risk mitigation, monitoring, and management are essential for the successful development and deployment of your game. By identifying potential risks and implementing strategies to mitigate them, your team can ensure a smooth and efficient development process, leading to a successful and well-received game.

6.3 PROJECT SCHEDULE

Creating a project schedule for a game development using Unity for a dodgeball game can be a complex task as it depends on various factors such as the game's scope, the number of features, the size of the development team, and the available resources.

1. Planning phase (2 weeks)
    - Define project scope, requirements, and goals
    - Create a detailed project plan and schedule
    - Identify necessary resources and allocate them accordingly
    - Set up communication channels and reporting mechanisms
2. Design phase (4 weeks)
    - Create game design documents including the game mechanics, UI/UX, levels, and characters
    - Develop the game's art style, visual design, and audio design
    - Conduct internal reviews and get feedback from stakeholders

3. Development phase (7 weeks)

    - Develop the game mechanics using Unity, including player movement, ball physics, and collision detection
    - Implement the UI/UX design, including menus, HUD, and feedback elements
    - Develop levels and environments, and design AI opponents
    - Conduct regular testing and debugging, and refine the game mechanics and design based on feedback

4. Quality assurance and testing phase (2 weeks)

    - Conduct rigorous testing on multiple platforms and devices
    - Identify and resolve bugs and issues
    - Conduct user testing and incorporate feedback

5. Deployment and launch phase (2 weeks)

    - Prepare the game for deployment on various platforms, including Steam, App Store, and Google Play Store
    - Develop marketing and promotional materials
    - Launch the game and monitor user feedback

It's important to note that this is just an example project schedule and can vary depending on the project's specific requirements of the available resources.

6.4 Team Organization

Team Structure

1. Team Members

- Ankur Patil
- Lalu Nair
- Sharan Thakur
- Viren Patil

2. Project Guide

- Guide: Asst. Prof. Sharad Adsure
- Co Guide : Asst. Prof. Deepika Jaiswal

3. Company Guide

- Mr. Varun Mehta (CTO)
- Mr. Avnish Oswal (Sr. Software Engineer)

Team Management Reporting And Communication

In our company, we were fortunate to have a mentor named Varun who played an essential role in our project development process. Varun would assign tasks to the team, taking into account each member's skill set and experience level. He provided clear and concise instructions for each task and ensured that we understood the requirements before starting work.

During the task execution, the team would frequently face questions and doubts about the task requirements. In such instances, we would seek help from another mentor, Avnish, who was readily available to clarify any uncertainties and provide guidance on best practices. Once we had a better understanding of the task requirements, we would start working on it.

After the task completion, we would submit our work to Avnish for review. Avnish would evaluate our output and provide us with feedback on areas of improvement or potential issues. We would then use this feedback to refine and improve our work.

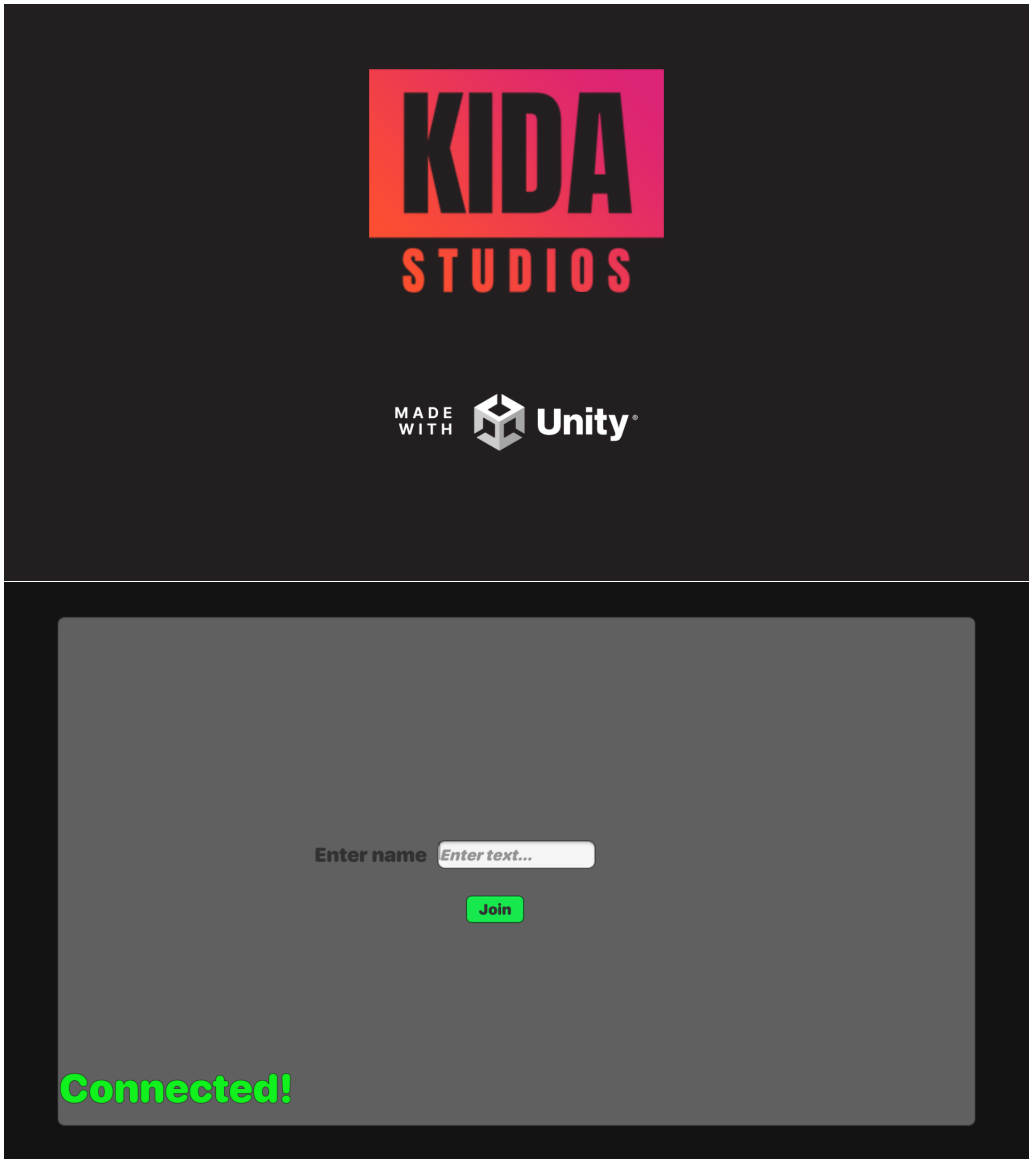
Once we had addressed the feedback, Varun would review our completed task to ensure that it met the project's quality standards. If there were any shortcomings, he would inform us, and we would work to rectify them. Once the task met the project's standards, we would proceed to the next task assigned by Varun.

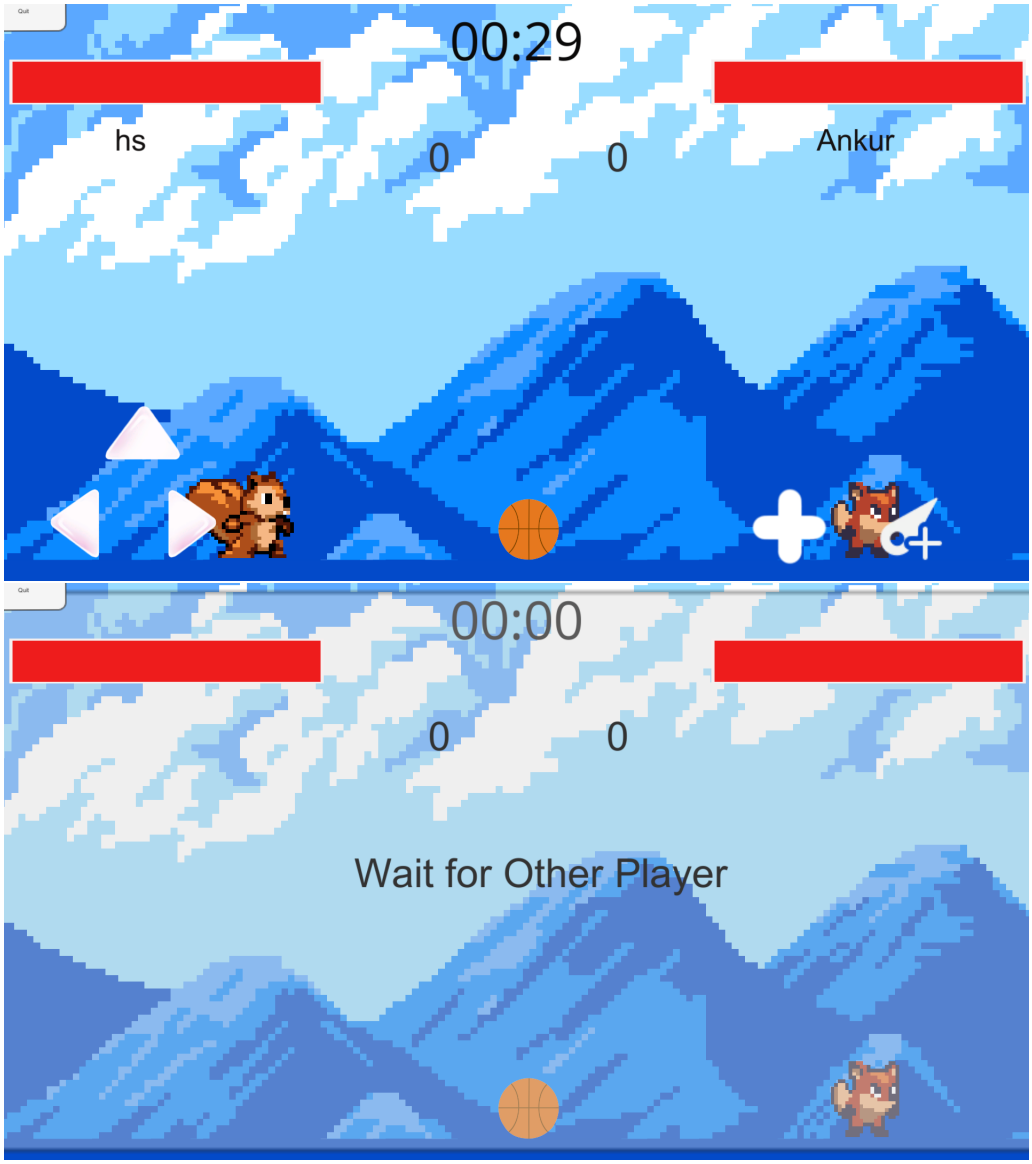
In summary, our company had a well-defined process for task assignment, execution, feedback incorporation, and review. This process helped us ensure that our work met the project's quality standards and contributed to a successful outcome.

7 RESULT

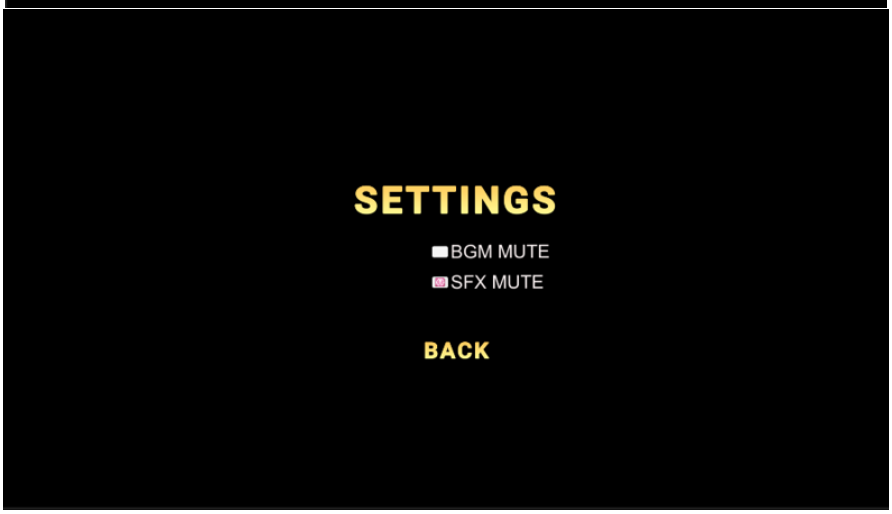
7.1 ScreenShots

Game UI




00:29
hs
0
0
Ankur
00:00
0
0
Wait for Other Player

Future Updates




5000
KIDA
Credits
BASKETBALL
COURT
REWARD : 1000
ENTRY : 500
PLAY
COURTROOM
REWARD : 2000
ENTRY : 1000
PLAY
FOOTBALL
REWARD : 10000
ENTRY : 5000
PLAY

8 TESTING

8.1 Introduction

Testing is an important part of software development life cycle. It is performed to ensure quality of the developed system. Testing includes a set of investigative activities that can be planned in advance and conducted systematically, to assure the stakeholder that system fulfils all the requirements gathered during requirement gathering phase. Software testing is one of the key elements in software projects that is often referred to as verification and validation. Verification refers to the set of activities that ensure that software correctly implements specified functionality. Validation refers to a set of activities built around traceability matrix which ensure that the functionality implemented by the system is traceable to customer requirements

Tests are the individual tests specified in a test plan document. Each test is typically described by

- An initial system state.
- A set of actions to be performed.
- The expected results of the test.

8.2 Implementation

Test cases are planned in accordance to the test process and documented with detailed test descriptions. These test cases use cases based on projected operational mission scenarios. The testing process also includes stress or load testing for stability purpose (i.e., at 95use, system stability is still guaranteed. The test process thoroughly tests the interfaces and modules. Software testing includes a traceable white box testing, black box testing and other test processes verifying implemented software against design documentation and requirements specified.

8.3 Objective

The software test plan (STP) is designed to test each module to measure its performance, to uncover bugs in the system, to set aright any flaws in logic that may be present, and to check logical flow from one module to another within system.

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

8.4 Testing Strategy

A strategy outlines what to plan, and how to plan it. A successful strategy is your guide through change, and provides a firm foundation for ongoing improvement. Unlike a plan, which is obsolete from the point of creation, a strategy reflects the values of an organization - and remains current and useful. When an organization tests its products or its tools, it tries to compare them against its expectations and values. By its nature, testing introduces change as problems are identified and resolved. A test strategy is necessary to allow these two impulses to work together. Furthermore, testing can never be said to be 'complete', and a core skill in testing is the justified management of conflicting demands; without a strategy, these judgements will be inconsistent to the point of failure.

Software development is a creative process. A test strategy is a vital enabler to this process keeping focus on core values and consistent decision-making to help achieve desired goals with best use of resource.

8.5 Types of Testing:

1. Unit Testing: Test individual components or functions of the game, such as player movement, ball physics, and scoring logic, to ensure they work correctly in isolation.

2. Integration Testing: Test the interaction between different components of the game, such as how player input affects movement, how the ball interacts with the environment, and how the game state updates based on player actions.

3. System Testing: Test the entire game as a whole to ensure that all components work together seamlessly and that the game meets its requirements and specifications.

4. Compatibility Testing: Test the game on different platforms, devices, and operating systems to ensure it works correctly and consistently across various environments.

5. Performance Testing: Test the game under various loads and conditions to ensure it maintains optimal performance, such as handling multiple players, managing network latency, and maintaining frame rates.

6. Load Testing: Test the game's ability to handle a large number of simultaneous players to ensure the game server and infrastructure can support the expected player base.

7. Playtesting: Have real players play the game to gather feedback on game mechanics, balance, and overall enjoyment. This can help identify issues that may not be apparent during other testing phases.

8. Regression Testing: Test the game after updates, bug fixes, or new features have been implemented to ensure that existing functionality has not been negatively affected.

8.6 Unit Testing

Unit testing is used to check the execution path of the module, function, and procedure of the system. Test is conducted with the help of normal data and abnormal data. This testing includes the different factors like statement coverage, branch coverage, loop processing, abnormality, and circulation etc. With the help of this Unit testing we check that all the statement in the code is executed or not so it avoids the dead code statement. It checks all the branches and execution path of the code. It ensures that all the internal method of program are executed and properly integrated with program.

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

8.7 Integrated system

In integrated testing, all the modules are checked together to ensure that all the modules are executing together according to the program specification. Once all the mod ules have been tested individually, the most legitimate question can be asked is that when all the modules are working properly, why there is need of integrated testing.

The answer is, though all modules are working properly problem may occur while interfacing individual module. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

8.8 Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items: Valid Input: identified classes of valid input must be accepted. Invalid Input: identified classes of invalid input must be rejected. Functions: identified

functions must be exercised. Output: identified classes of application outputs must be exercised. Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

9 Test Cases

Test Case ID	Test Cases	Actual Result
T1	Verify that the game can be launched successfully on different platforms (Windows, Mac, etc).	The game was successfully launched on Windows, Mac platforms.
T2	Verify that the game can be played in multiplayer mode with at least two players.	The game was played in multiplayer mode with two players, and it worked correctly.
T3	Verify that the game's graphics and sound effects are displayed correctly.	The game's graphics and sound effects were displayed correctly.
T4	Verify that the game's controls are responsive and intuitive.	The game's controls were responsive and intuitive.
T5	Verify that the game's scoring system is working correctly.	The game's scoring system was working correctly.
T6	Verify that the game's collision detection system is working correctly.	The game's collision detection system was working correctly.
T7	Verify that the game's power-ups are working correctly.	The game's power-ups were working correctly.
T8	Verify that the game's network connectivity is stable and reliable.	The game's network connectivity was stable and reliable.

Testing with NUnit in Unity:

NUnit is an open-source unit testing framework for the .NET Framework and Mono. It serves the same purpose as JUnit does in the Java world, and is one of many programs in the xUnit family.

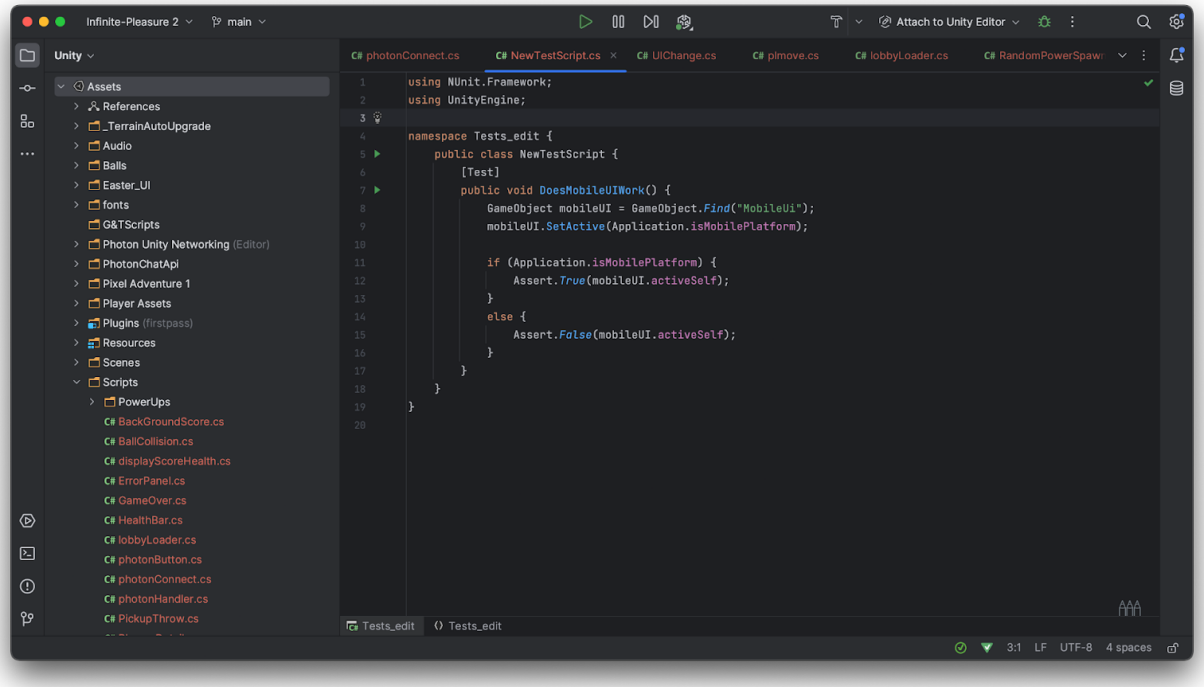
NUnit provides a console runner (nunit3-console.exe), which is used for batch execution of tests. The console runner works through the NUnit Test Engine, which provides it with the ability to load, explore and execute tests. When tests are to be run in a separate process, the engine makes use of the nunit-agent program to run them.

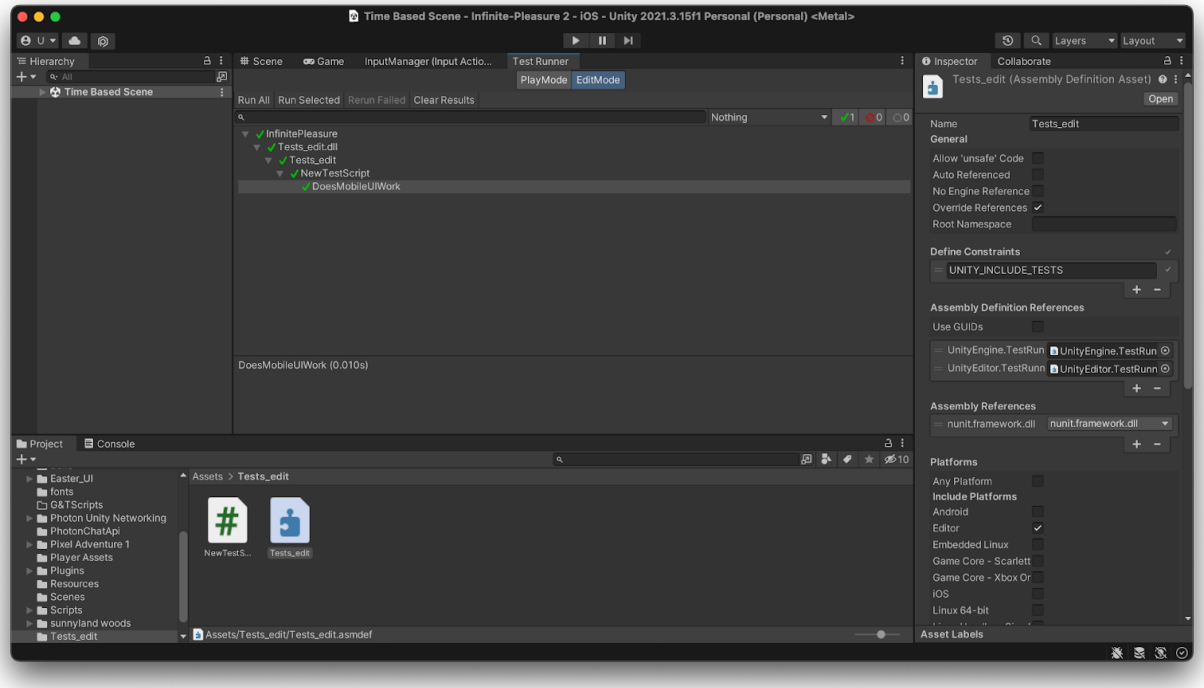
Unit provides a rich set of assertions as static methods of the Assert class. If an assertion fails, the method call does not return and an error is reported. If a test contains multiple assertions, any that follow the one that failed will not be executed. For this reason, it's usually best to try for one assertion per test.[citation needed]

Nunit 3.x is supporting multiple assertions.

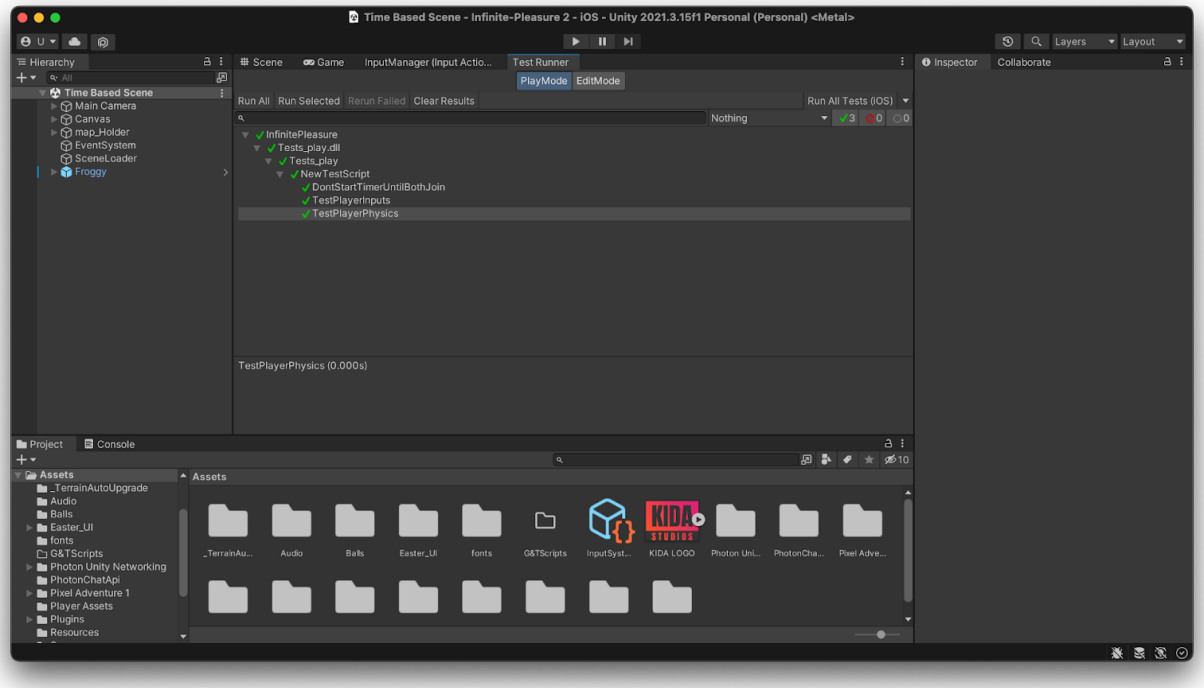
```
[Test]
public void ComplexNumberTest()
{
   ComplexNumber result = SomeCalculation();

   Assert.Multiple(() =>
   {
      Assert.AreEqual(5.2, result.RealPart, "Real part");
      Assert.AreEqual(3.9, result.ImaginaryPart, "Imaginary part");
   });
}
```

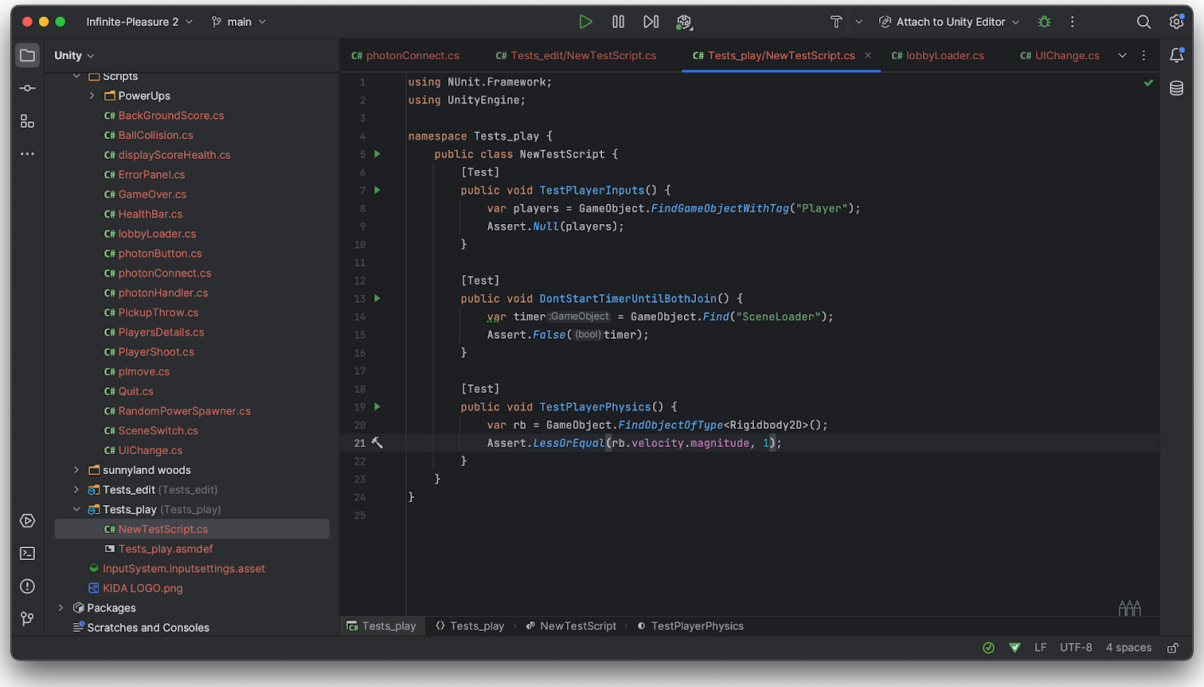
Edit mode tests





Play mode tests





10 CONCLUSION

Defining appropriate matchmaking is the first step in developing a matchmaking system. If you don't offer the machine the rules for what that actually implies, no algorithm can come up with good matches. I've talked about our regulations in this post, which are primarily centered on skill, ping, and pre-made. You can change how the rules are defined or add new rules depending on what is important for your game.

10.1 APPLICATIONS

- **Interactive Experience** These touchscreen devices are popping up everywhere. Traditional maps in malls have been replaced with interactive touchscreen maps, McDonald's now offers a touchscreen experience for ordering groceries, and major movie theater chains now have kiosks to purchase tickets. All of these solutions require a rich multimedia platform to run them, and the popularity of such solutions is undiminished. Unity's extensibility allows integration with the existing backends these companies already have, and Unity's powerful graphics capabilities make it easy to visually blow people's minds. Again, Unity's various export options allow building on any platform that supports a touchscreen interface.

- **Architectural Visualization** Video game engines like Unity can already handle large amounts of complex geometry and render realistic lighting and surfaces. Exploring virtual buildings for engineering and architectural purposes is easy with a game engine like Unity. It's fairly easy to import data from Sketchup or Revit directly into Unity and add Unity's high-end graphics capabilities. Architects can not only look at client concept renders and blueprints, but also use game engines to create 3D experiences that can be delivered to client mobile, desktop, or web browsers. We actually built the Cinema Suite ourselves.

- **Previsualization of Film** This one is quite similar to the animation, but not exactly the same. Live action filmmaking requires a lot of advance planning: in many cases, the more planning the better. Drawing traditional scenarios can be a good starting point, but sometimes it can be useful to have a more detailed and realistic approximation of what these scenes will look like. With Unity, Asset Store and The Cinema Suite it is very much possible. As we discussed with Animation, Unity and Cinema Director (part of Cinema Suite) can work together to create great animated sequences in Unity.

  With our Cinema Pro Cams, you can use the same real-world lenses you'd find on set right in the Unity game engine, so you can tell more precisely what your final shot will look like. Cinema Mocap 2 and our soon to be released facial motion capture tool will be great for making your characters move with very little effort. You can find many props, characters,

environments and building tools in Unity's Asset Store. It's a great way to populate your virtual scene. Finally, you can use Cinema Director to time and approximate how your movie will look when it's shot and edited.

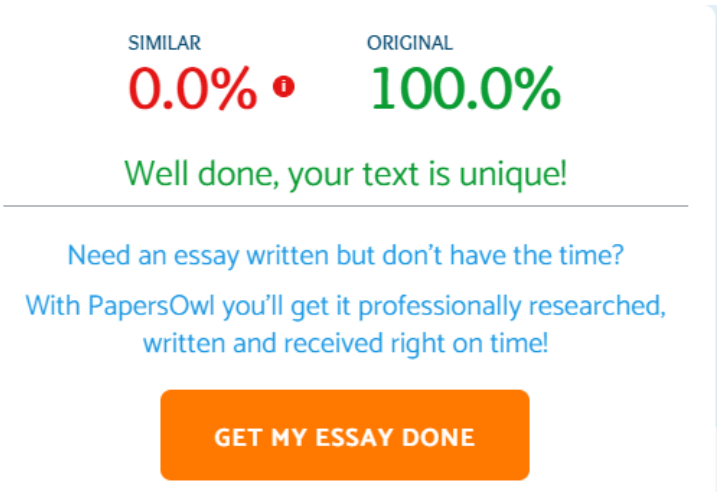
ANNEXURE B

APPENDIX B

Details of game deployment: ”itch.io” is an open marketplace for independent digital creators with a focus on independent video games. Our Games link on Itch.io our deployment platform of choice.

APPENDIX C

Plagarism Report:



REFERENCES

1. Unity Multiplayer Game development with Photon PUN2 [2020],
2. Make a 2D Platformer Character with State Machines in Unity,
3. Unity 2020 URP Make a juicy 2d Shooter prototype
4. Research on the 3D Game Scene Optimization of Mobile Phone Based on the Unity 3D Engine — IEEE Conference Publication,
5. Developing a game application to encourage face-to-face local gaming experience — IEEE Conference Publication — IEEE Xplore,
6. Developing MOBA games using the Unity game engine — IEEE Conference Publication — IEEE Xplore,
7. SOUL: Simulation of Objects in Unity for Learning — IEEE Conference Publication,
8. Computing Games: Bridging the Gap Between Search and Entertainment — IEEE Journals & Magazine,
9. Game or Watch: The Effect of Interactivity on Arousal and Engagement in Video Game Media — IEEE Journals & Magazine — IEEE Xplore,