COMP4801 Final Year Project

Interim Report

The Road to Castle 3D

Supervisor:

Dr. T. W. Chim

Group Members:

Leung Chiu Yuen, Rain (3035857294)

Ko King Nam, Vincent (3035864297)

Date of Submission:

17 Jan 2024

Abstract

The popularity of RPGs has spanned over decades, along with the development in gaming technology and growth of the game market. This project aims to develop and deliver a third-person 3D fantasy action RPG through a game rework. To achieve this goal, several gameplay systems and features will be implemented as objectives, by integrating Unity and other applications. Currently, five major game features, namely Inventory, Combat, Quest, Leveling, and Save and Load, have been tested and deployed successfully. These systems control the game mechanics in different scenarios and are the basic framework for future game development. In addition, a village demo game map with a few interactable NPCs and 3D house models has been developed to demonstrate the functionalities of these features. During the development, difficulties on 3D modelling, collaboration, and time have been encountered and mitigated respectively. In the coming months, the remaining work, including game maps, dungeon level editor, characters models, and animations will be completed by the scheduled deadline.

Acknowledgment

We would like to express sincere gratitude to our supervisor, Dr. T. W. Chim, for his guidance and support throughout the development of this final year project. Dr. Chim has given us many valuable comments and suggestions for our game.

Table of Contents

List of Figures

List of Tables

Abbreviations

3D	Three-dimensional
AI	Artificial Intelligence
CG	Computer Graphics
ECS	Entity Component System
IDE	Integrated Development Environment
NPC	Non-player Character
RPG	Role-playing Game
UI	User Interface
XP	Experience Points

1 Introduction

This chapter introduces the context behind the project in 4 sections. In Section 1.1, a brief background of this project is introduced. In Section 1.2, project rationales are discussed. In Section 1.3, the purpose and objectives of this project are listed. In Section 1.4, an outline of subsequent chapters is provided.

1.1 Background

Role-playing is generally considered one of the most popular game genres, which originated from the publication of Dungeon and Dragons (D&D) in 1974 [1]. The term role-play often correlates with make-believe, an action of imaging and pretending something unreal. In RPGs, players can cast themselves and act as characters in fictional settings, based on "the range of imagination" [2] of game developers.

Over the past few years, there has been a rapid growth in the global gaming market, possibly due to pandemic lockdowns [3]. The game market has witnessed great success in classic RPG franchises, such as *Baldur's Gate*, *Monster Hunter*, and *Dark Souls*. For a long time, the video game market has been dominated by large publishers and studios, with their huge investments in game development and research. But thanks to advances in gaming technology, indie game developers are now capable of creating games with good quality.

Various game development tools and software are now ubiquitous and often accessible to the public. There are free and yet powerful game engines, such as Unity and Unreal Engine 5, available on the market. Moreover, AI technologies are now widely employed in game development to lower production costs. For instance, content creators can utilize AI to write storylines, generate images, and create music for games [4].

1.2 Motivation

As game enthusiasts, the inspiration from RPG masterpieces and the great interest in their game design principles are the sources of motivation for this project. This project serves as a valuable opportunity for the team to gain hands-on experience in game development, and to develop communication and collaboration skills from teamwork.

1.3 Objectives

This project will be a rework and migration of *The Road to Castle* [5], which is a text-based fantasy RPG played on Linux terminal. The final deliverable will be a third-person 3D fantasy action RPG, with implemented gameplay systems and contents.

The main purpose of this game project is to present a fantasy game world with a captivating gaming experience to players. To achieve this, multiple game features and systems will be accomplished as the objectives:

- Develop a flexible, maintainable, and extendable framework from scratch
- Create a compelling storyline, integrated with tutorials
- Provide a user-friendly user interface (UI)
- Design maps with diverse terrains, eco-systems, and weathers
- Provide an inventory system
- Design and implement a balanced combat mechanism
- Allow players to save and load their progress locally and remotely
- Provide a level system for character development
- Allow players to design and share their own dungeons via an in-game editor

1.4 Report Outline

In Chapter 2, the methodologies of this project will be discussed, including development tools, concepts, and techniques to be used and their justifications. Following that, Chapter 3 introduces the results achieved in this project, as well as the problems encountered with mitigations. Finally, there will be a summary of current progress and future work plan in Chapter 4.

2 Methodology

In this chapter, software, concepts, and other methods used in this project and their justifications are discussed in 8 sections. In Section 2.1, the choices of game engine, IDE, and CG software are discussed. In Section 2.2, the project structure and codebase design are explained. In Section 2.3, the event-driven design of the game is introduced. In Section 2.4, the singleton pattern is discussed. In Section 2.5, the physics and projectile motion is elaborated. In Section 2.6, serializations on game data and configuration are mentioned. In Section 2.7, player movements such as walking and climbing are discussed. In Section 2.8, different useful techniques in Unity are presented.

2.1 Development Tools

2.1.1 Game Engine – Unity

The game will be made with Unity, a free and user-friendly game engine, which supports cross-platform 3D game development with C# scripting. It acts as the platform that loads and integrates all gameplay systems and local game contents into a game.

Unity also serves as the "backbone" for connecting all contents from different software. Figure 2.1 below illustrates the concept of such integration between these applications. In the figure, empty entities are created in Unity as containers to store scripts, 3D models, and animations from IDE and 3D modeler. These entities are later composed into unique game objects for different game scenes.

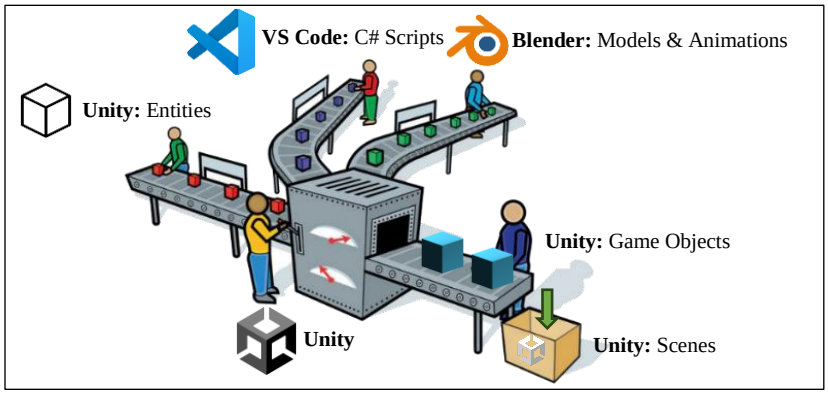


Figure 2.1 Conceptual "assembly line" of the project

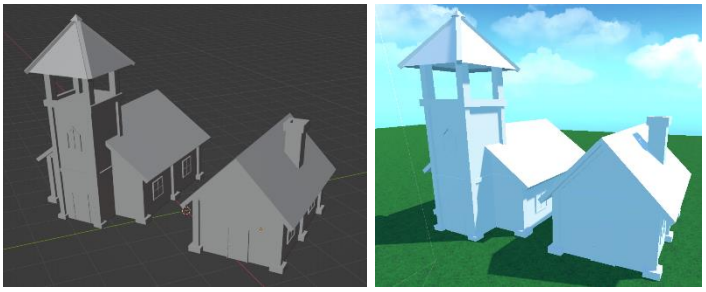
Compared with other alternatives like Unreal Engine 5, Unity is more user-friendly in terms of its interface and functions, and it is easier to run on small-scale projects. Also, it provides some useful built-in systems that could facilitate the development process.

2.1.2 IDE – Visual Studio Code

An integrated development environment (IDE) is an application that provides developer tools for software development. Visual Studio Code is selected as the IDE, as it provides high extensibility with custom extensions to fulfill different requirements. For this project, extensions can be installed to support Unity development and debugging on C# scripts.

2.1.3 3D Modelling and Animation – Blender

Blender is a free 3D creation software that supports most CG functionalities, such as modelling, animation, and rendering. In the project, architectural models, character models, and animation can be created in Blender (see Figure 2.2a), and then imported directly to Unity (see Figure 2.2b).



(a) (b)

Figure 2.2 3D models created in Blender (a) and imported to Unity (b)

2.2 Codebase Design

To develop a flexible and extendable gaming framework, a well-organized file structure is required. In this project, game contents are modularized and grouped under the standard convention [6]. For example, a “Scripts” directory is used to contain all scripts of the game, with sub-directories separated by content types. This improves division of work, as team members can develop different features simultaneously without interruption.

In addition, Unity attributes and data container classes have been applied to facilitate the development process on game assets. For instance, with “[CreateAssetMenu]” attribute and

objects inherited from ScriptableObject class in Unity, self-defined assets can then be created by simply clicking on the asset menu, instead of repetitive programming, as shown in Appendix A.

For code reuse, inheritance is extensively used in this project. For example, a GameItem class has been developed as the superclass for different game items, such as game equipment and consumables, to inherit common behaviors.

2.3 Event-Driven Development

For games, event-driven design is an architecture that implements the game flow based on events, instead of a predetermined series of actions. Events in games are generally the inputs from users, such as mouse click and keypress. These events are captured as the "triggers" for follow-up actions or other events designed by the developers.

Such design allows the game to provide prompt and dynamic responses to players depending on their behaviors, as well as for system-to-system communications. For example, the combat system can notify the quest system whenever the player has made progress in quests. Also, different events are independent and are developed separately for specific tasks, which facilitate the development and testing process.

2.4 Singleton Pattern

A singleton is a globally accessible class that has only one instance at a time. A singleton object will not be re-assigned after instantiation. This helps protect variables and values from unexpected behaviors. One example is to create a singleton for user settings to prevent them from changing when switching to a new game scene.

As Unity allows configuration on objects in the editor before the game runs, constant values can be assigned to singletons. This prevents repetitive coding and simplifies the development process of complicated systems, such as the UI systems.

2.5 Physics

To make the game more immersive, simulations on physics such as gravity and collision are added to the game world. For example, projectiles with weights like arrows are shot

with parabola-like locus instead of a straight line. Such curve on a 2D plane follows the equations derived in Appendix B. In a 3D game world, this is done by first forming a 2D plane between the faces of two entities, such as a player and an enemy (see Figure 2.3). An ideal angle can then be calculated and assigned to the projectile.

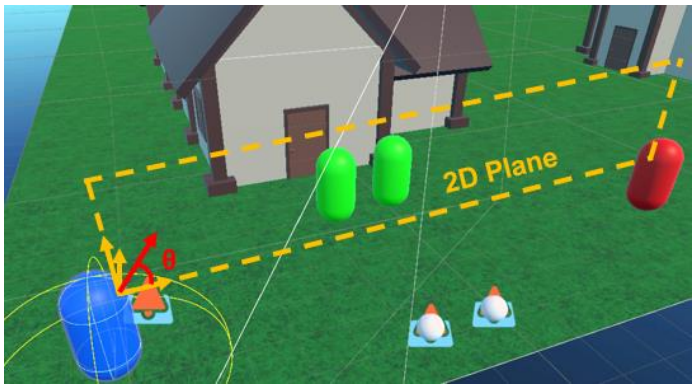


Figure 2.3 Player faces enemy such that a 2D plane is formed.

2.6 Serialization

Serialization is a process of converting data structures into certain file format for storage and transmission. Objects are saved into specific file formats via serialization and are later loaded back to the game.

To allow players to save their game progress and resume afterwards, a “savable” object class and a Save and Load system are desired. One possible solution is to loop through all game objects once to check whether they are savable or not. A GameData class is designed as the container for the game data in objects. The GameData objects will be converted and saved as JSON files using Unity built-in utilities. One example is to save the position of a player. An ISaveLoad interface can be implemented with functions to save the Vector3 position of player into a GameData object, and to convert the object into JSON.

Besides game data, other types of objects such as the game configurations are also saved and loaded with the Save and Load system using serialization. To improve code reusability, a generic DataPersistence class is designed to handle these cases. A snippet can be found in Appendix C.

2.7 Player Movement

To support player movements on the map, two movement states for moving and standing are required. These two states are combined to provide a smooth movement behavior of the player, such as walking, jumping, and climbing. Also, they allow later integrations of character animations. Figure 2.4 below shows the two states and their respective sub-states.

```
public enum MovementStates{
    RUN, WALK, IDLE, JUMP, CLIMB, AIR
}
public enum GroundStates{
    NORMAL, SLOPED
}
```

Figure 2.4 Player movement states and sub-states.

2.8 Unity Techniques

In this project, Unity APIs are utilized to reduce development time. For example, Physics API provides some useful functions for ray casting and creating colliding spheres on the fly. Rays are cast onto the floor to detect whether the player is on the ground, while collider spheres are created to detect objects near the player. These can be applied in player-object interactions, where spheres are used to detect and check whether colliding objects are interactable.

It is also remarkable that a wise use of Unity can sometimes lead to no-code solutions. For example, by using built-in packages like Cinemachine, the camera controls can be simplified.

3 Results and Discussion

This chapter presents the current project progress and technical issues encountered during the development in 2 sections. In Section 3.1, status and details of the progress are provided. In Section 3.2, the problems faced in the project and their mitigations are discussed.

3.1 Current Progress

Five major game features, namely Inventory, Combat, Quest, Leveling, and Save and Load, have been implemented and deployed in the game. Also, a village game map with a few NPCs and buildings has been created to demonstrate these features.

3.1.1 Inventory

Inventory management is considered an important feature of an RPG. In the game, an inventory system is developed for players to pick up and store items in the inventory, with a UI to view, use, and equip game items. For instance, players can pick up items on game maps with a UI panel and the "F" hotkey, and these items will be stored as metadata in the inventory (see Figure 3.1). Also, players can discard an item by clicking the cross button on the right corner.

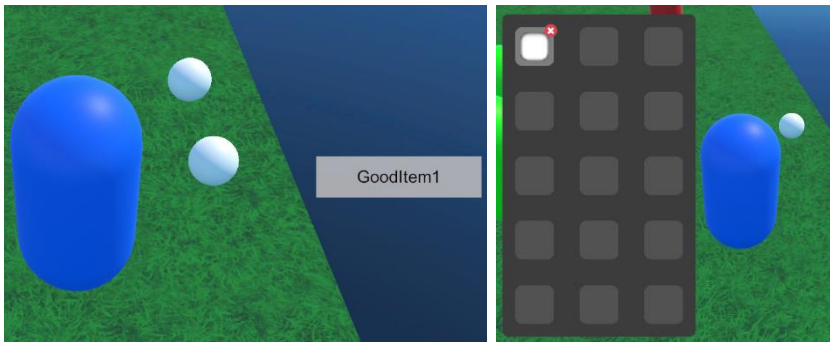


Figure 3.1 Nearby items appear on UI panel (left) and are picked up (right).

3.1.2 Combat

A basic combat system is developed to handle all the logic in battles, such as calculation of attack damages and statistics of game items. For example, equipping with a helmet will increase specific player stats like armor, which lowers the damage received. Also, players can choose to use consumable game items for combat, such as using HP potions for healing

or strength potions for attack damage. During the battle, damage indicators are created to show the damage received from or delt to an enemy (see Figure 3.2).

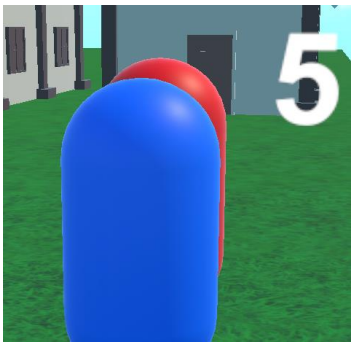


Figure 3.2 Damage indicator pops out when player attacks an enemy.

Once a player defeats an enemy, loots are dropped according to the current game configuration. Such configuration associates drop items with specific enemies, and defines their corresponding drop rates of the game world. An example is shown in Appendix D.

Additionally, an enemy AI agent is also developed to make the game more immersive. When a player moves close enough to an enemy, it will start chasing and try to attack the player. When the HP of an enemy drops to a low level, it will escape from the player.

3.1.3 Quest

A quest system is developed to allow players to progress through the storyline with side quests, known as the storyline tasks. Players are rewarded with experience points, game items, and money once they complete a task.

A location indicator is created to show the direction of the quests, usually the position of the NPCs. These indicators will not leave the screen and can lead players to where a quest takes place (see Figure 3.3).

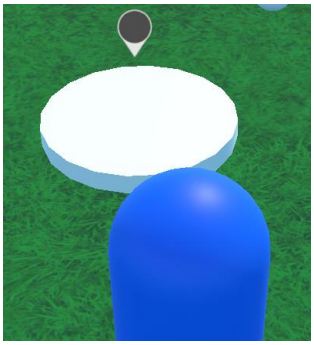


Figure 3.3 A quest indicator for an investigation quest.

The system is designed to be a maintainable and extendible product for future development. For example, it allows developers to create different types of custom tasks and mini games, such as investigation quest that requires players to move to a certain location, or kill quest that asks players to defeat certain number of target enemies, as shown in Appendix E.

3.1.4 Leveling

A leveling system is built to manage the experience points (XP), levels, and their effects on player attributes and abilities. The threshold for each level, or the XP required to level up, is designed to grow quadratically to balance the difficulty of the game. When a player accumulates enough XP and reaches a threshold, the system will trigger a level-up event, which increases the level and corresponding player stats according to the level (see Figure 3.4).

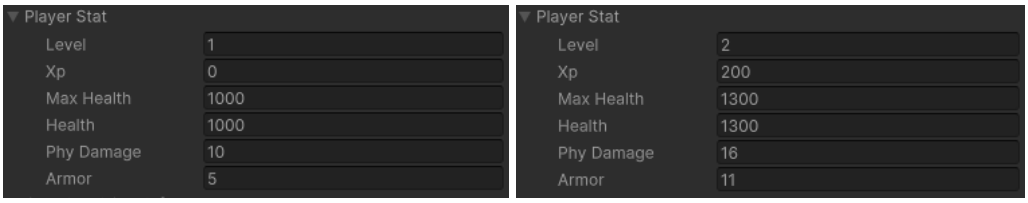


Figure 3.4 Player stats before (left) and after a level-up event (right).

3.1.5 Save and Load

A Save and Load system is developed based on serialization method as mentioned. It allows players to save and load their progress during gameplay. Also, game configurations in different game scenes can also be saved and loaded. One example is the above-mentioned configuration for looting on item drops and drop rates.

3.1.6 Village Game Map

A village game map is built to show and test the implemented and new features. A grass terrain and some 3D architectural models such as houses are created for the map. Textures and materials are added to these assets in Unity and Blender. Also, a few interactable NPCs have been added to the map as a demo for dialogue and quest system (see Figure 3.5).

Figure 3.5 UI panel (left) and dialogue box (right) for player-NPC interaction.

3.2 Difficulties and Mitigations

3D modelling is generally considered one of the most challenging tasks in game development. To save time and effort, developers usually choose to buy models directly from professional game artists through platforms like Unity Asset Store. However, these assets could be expensive and may not perfectly fit the game. Given limited budgets, the team decided to create part of the 3D models with Blender.

Besides 3D modelling, there were problems on collaboration in Unity. For this project, GitHub is used as the collaboration platform. However, conflicts often occurred when the project team was working on the same files simultaneously. To mitigate this issue, a separate branch is created for development purposes, and new committed features are merged via pull requests, which is an effective way to avoid conflicts.

Also, time constraint is one of the major problems. To accelerate the development process, software engineering techniques like code-reuse, and pair programming have been applied. For example, regular meetings have been hosted for the team to share their progress. In the meeting, one member, known as the “driver”, focuses on developing new game features, while another one, known as the “navigator”, offers suggestions and focuses on debugging.

4 Conclusion

To conclude, this project aims to deliver a fantasy action RPG world through a rework on a previous game project. All proposed gameplay systems, such as Inventory and Combat, have been tested and successfully implemented. In addition, a village game map with a few basic 3D architectural models has also been created. Extra features like Quest have also been added to the game.

4.1 Future Work

As planned, most of the objectives in the minimum viable project development, including inventory, combat, and leveling systems, have been completed (see Table 4.1). In the coming months, the focus will be on the remaining work, including new game maps, world editor, 3D models and animations for characters.

Table 4.1 Project schedule with current phase highlighted

Milestones	**Objectives**	**Deadline**
Phase 1 (Inception)	Detailed project plan Project webpage	1 Oct 2023
Minimum Viable Project Development	Inventory system Combat system Leveling system Basic 3D models Basic map design	31 Dec 2023
First Presentation		8-12 Jan 2024
Phase 2 (Elaboration)	Preliminary implementation Detailed interim report	21 Jan 2024
Final Project Development	Dungeon Level Editor Character models Animations	15 Mar 2024
Game Testing and Adjustments		25 Mar 2024
Final Presentation		15-19 Apr 2024
Phase 3 (Construction)	Finalized implementation	23 Apr 2024

	Final report	
Project Exhibition		26 April 2024

4.2 Limitations

The main project constraints are time and abilities. Given a period of 6 months and small team size, it is challenging to develop a good RPG with various gameplay systems and features on schedule, especially when the project team is inexperienced.

Game development involves a wide variety of soft and hard skills, which are generally specified by roles in real production. For instance, game developers specialize in coding and programming, while game designers create game content. In this project, all the teammates have taken on several roles that are unfamiliar to them. As beginners in game development, the project team is facing steep learning curves, and the development process is slower than expected, especially for 3D modelling.

References

[1] P. Mason, “In Search of the Self”, *Beyond Role and Play: tools, toys and theory for harnessing the imagination*. Helsinki: Ropecon ry, 2004, pp. 1-14.

[2] M. Hitchens and A. Drachen, “The Many Faces of Role-Playing Games,” International Journal of Role-Playing, no. 1, pp. 3–21, Dec. 2008, doi: https://doi.org/10.33063/ijrp.vi1.185.

[3] S. Read, “Gaming Is Booming and Is Expected to Keep growing. This Chart Tells You All You Need to Know,” *World Economic Forum*, Jul. 28, 2022. https://www.weforum.org/agenda/2022/07/gaming-pandemic-lockdowns-pwc-growth/

[4] N. Anantrasirichai and D. Bull, “Artificial Intelligence in the Creative industries: A Review,” *Artificial Intelligence Review*, vol. 55, no. 1, Jul. 2021, doi: https://doi.org/10.1007/s10462-021-10039-7.

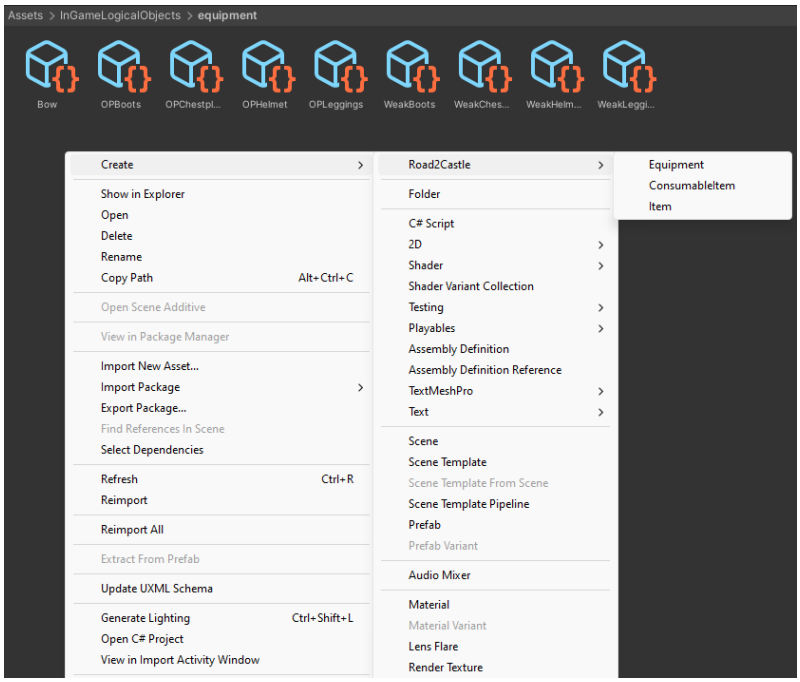
[5] “The Road to Castle”, *GitHub*. https://github.com/pystander/ENGG1340_Gp106_RTC

[6] “Best practices for organizing your Unity project | Unity,” *Unity*. https://unity.com/how-to/organizing-your-project

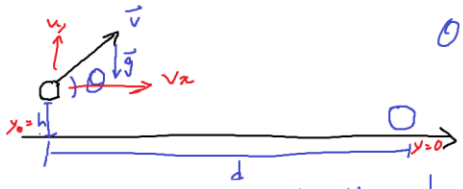
Appendices

Appendix A: Creating GameItem in Unity Asset Menu with Attributes

```
[CreateAssetMenu(fileName = "New Equipment", menuName = "Road2Castle/Equipment")]
public class GameEquipment : GameItem{
    [Header("Equipment info")]
    public EquipmentSlotName slot;
    public WeaponType weaponType = WeaponType.NONE;
    public EquipmentEffect effect;

    public override bool use(GameEntity actor){
        return base.use(actor);
    }
}

public enum EquipmentSlotName{
    HEAD, CHEST, LEGS, FEET,

    WEAPON,
    SHIELD
}

public enum WeaponType{
    NONE, SWORD, BOW, STAFF
}

[System.Serializable]
public class EquipmentEffect{
    public float armorModifier = 1;
    public float damageModifier = 1;
}
```



Appendix B: Equation Derivations of 2D Projectile Motion



$v_y = v \sin\theta$

$v_x = v \cos\theta$

Type of Formula	Linear	Angular
Velocity	$v = \frac{\Delta s}{\Delta t}$	$\omega = \frac{\Delta\theta}{\Delta t}$
Acceleration	$a = \frac{\Delta v}{\Delta t}$	$\alpha = \frac{\Delta\omega}{\Delta t}$
Displacement	$s = v_i t + \frac{1}{2}at^2$	$\theta = \omega_i t + \frac{1}{2}\alpha t^2$
Motion with time canceled out	$v_f^2 - v_i^2 = 2as$	$\omega_f^2 - \omega_i^2 = 2\alpha\theta$

$$s_y = y_0 + v_{0y} t + \frac{1}{2} a_y t^2$$

$$0 = h + v_{0y} t - \frac{1}{2} a_y t^2$$

$$= h + \cancel{v}\sin\theta \left(\frac{d}{\cancel{v}\cos\theta}\right) - \frac{1}{2} a_y \left(\frac{d}{v\cos\theta}\right)^2$$

$$= h + \frac{d\sin\theta}{\cos\theta} - \frac{1}{2} a_y \left(\frac{d}{v\cos\theta}\right)^2$$

$$s_x = v_{0x} t + \cancel{\frac{1}{2} a_x t^2}$$

$$t = \frac{d}{v_{0x}}$$

$$t = \frac{d}{v\cos\theta}$$

$$= h\cos^2\theta + d\sin\theta\cos\theta - \boxed{\frac{1}{2} a_y \frac{d^2}{v^2}}$$

$$= h(1 - \sin^2\theta) + d\sin\theta\cos\theta - a$$

$$= h - h\sin^2\theta + d\sin\theta\cos\theta - a$$

$$= -h\sin^2\theta + d\sin\theta\cos\theta + (h - a)$$

$$= -h\sin^2\theta + d\,\frac{1}{2}\sin 2\theta + (h - a)$$

$$= -\frac{h}{2}(1 - \cos 2\theta) + \frac{d}{2}\sin 2\theta + (h - a)$$

$$= \frac{h}{2}\cos 2\theta + \frac{d}{2}\sin 2\theta + \left(\frac{h}{2} - a\right)$$

$$= h\cos 2\theta + d\sin 2\theta + h - 2a$$

$$= \sqrt{h^2 + d^2}\cos\left(2\theta - \tan^{-1}\left(\frac{d}{h}\right)\right) + (h - 2a)$$

$$A\sin(\alpha) + B\cos(\alpha) = R\cos(\alpha - \phi)$$

$$\phi = \tan^{-1}\left(\frac{A}{B}\right)$$

$$R = \sqrt{A^2 + B^2}$$

$$\frac{2a - h}{\sqrt{h^2 + d^2}} = \cos\left(2\theta - \tan^{-1}\left(\frac{d}{h}\right)\right)$$

$$\cos^{-1}\left(\frac{2a - h}{\sqrt{h^2 + d^2}}\right) = 2\theta - \tan^{-1}\left(\frac{d}{h}\right)$$

$$\boxed{a = \frac{1}{2} a_y \frac{d^2}{v^2}}$$

$$\cos^{-1}\left(\frac{a_y \frac{d^2}{v^2} - h}{\sqrt{h^2 + d^2}}\right) - \tan^{-1}\left(\frac{d}{h}\right) = 2\theta$$

Appendix C: Saving GameData with DataPersistence<GameData>

```
public class DataPersistence<T> where T: new(){
    private T data;

    public string dataFolder = "Saves";
    public string dataFile;

    protected string fullPath;

    public DataPersistence(string dataFile){
        this.dataFile = dataFile;
        fullPath = Path.Combine(dataFolder, dataFile);
    }
    public DataPersistence(string dataFolder, string dataFile){
        this.dataFolder = dataFolder;
        this.dataFile = dataFile;
        fullPath = Path.Combine(dataFolder, dataFile);
    }

    public T getData(){
        return data;
    }
```

```
public void save(){
    List<ISaveLoad<T>> objects = findISaveLoadObjects();
    foreach(ISaveLoad<T> obj in objects){
        obj.save(data);
    }

    try{
        Directory.CreateDirectory(Path.GetDirectoryName(fullPath));
        string data = JsonUtility.ToJson(this.data);
        using(FileStream stream = new FileStream(fullPath, FileMode.Create)){
            using(StreamWriter writer = new StreamWriter(stream)){
                Debug.Log("Saving data to: " + Path.GetFullPath(fullPath));
                writer.Write(data);
            }
        }
    }catch(Exception e){
        Debug.LogError($"Cannot save data file to ({fullPath}): {e.Message}");
    }
}
```

Appendix D: Loot-drop Configuration JSON File

```
{
    "assetPaths": {
        "entries": [
            {"key": "OPHelmet.asset", "value": "Assets/InGameLogicalObjects/equipment/OPHelmet.asset"},
            {"key": "OPChestplate.asset", "value": "Assets/InGameLogicalObjects/equipment/OPChestplate.asset"}
            {"key": "OPLeggings.asset", "value": "Assets/InGameLogicalObjects/equipment/OPLeggings.asset"},
            {"key": "OPBoots.asset", "value": "Assets/InGameLogicalObjects/equipment/OPBoots.asset"},
            {"key": "OPBow.asset", "value": "Assets/InGameLogicalObjects/equipment/Bow.asset"}
        ]
    },
    "entitiesDrops": {
        "entries": [
            {
                "key": "Enemy",
                "value": {
                    "list": [
                        {"id": "OPHelmet.asset", "type": "e", "prob": 0.5},
                        {"id": "OPChestplate.asset", "type": "e", "prob": 0.5},
                        {"id": "OPLeggings.asset", "type": "e", "prob": 0.5},
                        {"id": "OPBoots.asset", "type": "e", "prob": 0.5},
                        {"id": "OPBow.asset", "type": "e", "prob": 0.5}
                    ]
                }
            }
        ]
    }
}
```

Appendix E: Creating A New Quest with the Developed Quest API

```
// SampleStory: AreaCondition -> Welcome msg -> Why me? -> KillCondition -> Done
Storyline sampleStory = new Storyline("Sample Story");
StoryTask task0 = new StoryTask("Walk to Location");
task0.rewards.xp = 200;
task0.rewards.items.addItem(weakHelmet);
task0.rewards.currencyStat.money = 9999;
task0.conditions.Add(new AreaCondition(new Vector3(6.63f,0.13f,3.65f), 2));

StoryTask task1 = new StoryTask("Welcome msg");
task1.addDialogue(new Dialogue(npc1, "Welcome to The Road to Castle 3D."));
task1.addDialogue(new Dialogue(npc1, "What you are looking at is a sample dialogue."));
task1.addDialogue(new Dialogue(npc1, "The dialogue will subject to change."));
task1.addDialogue(new Dialogue(npc1, "Please find Npc2 for more information."));
task1.addDialogue(new Dialogue(npc2, "Hey, I'm here!!!!"), true);

StoryTask task2 = new StoryTask("Why me?");
task2.addDialogue(new Dialogue(npc2, "Welcome to The Road to Castle 3D again."));
//...
StoryTask task3 = new StoryTask("Revenge");
task3.conditions.Add(new KillCondition(1));
task3.addDialogue(new Dialogue(npc2, "I want you to beat one of those monsters up."), true);

StoryTask task4 = new StoryTask("Thank you");
task4.addDialogue(new Dialogue(npc2, "Thank you for testing out the quest system."), true);

sampleStory.addTask(task0);
//...
stories.Add(sampleStory);
```