# CS 3540 – MCA3 FPS Game

In this assignment, you will conceptualize and develop an FPS game. While your game should utilize the same FPS game mechanics shown in class demos, you should come up with a new concept for your FPS game, which can be literally anything other than the in-class game “Shooting Dementors”.

Ideally, you would use some of the assets that are available on Canvas. For your reference, two plausible packages are attached to the assignment. I’d encourage you to use one of these attached packages for this game, particularly the Sci-Fi Dungeon package! That said, you can use a completely different set of assets.

## Requirements

### Player (2pts)

1. For this game, please use the same FPS character controller mechanic for your player, with a Character Controller.
2. We should be able to change our FPS view using the mouse (implement Mouse Look).
3. We should be able to control the player using WASD or arrow keys.
4. The jumping mechanic should be added to the player (so when the Space key is pressed, the player should jump). The player should not jump when it is in the air, but we should be able to control the player’s direction while up in the air (similar to the in-class game).

### Player Health and UI (2pts)

1. Write a PlayerHealth script that controls the player’s health. The starting health should be modifiable through the Inspector with a default value of 100.
2. The player should lose health points when contacted by the enemies.
3. Add a UI component to visualize the player health. Use a UI slider for this purpose, similar to the in-class example. Scale it appropriately.
4. When the health points value is 0, the player should die.

### Enemies (4pts)

1. There should be enemies that follow and try to get to the player. When one of the enemies collides with the player, the player should lose health points.
2. Your game should have at least two different types of enemies. The enemies should differ in the amount of damage they cause to the player.
3. These enemies should be spawned using an Enemy Spawner script.
4. Create a Particles effect using the Particles System and use it when the enemy dies (similar to the in-class example).

### Interactables (4pts)

1. Add some probs to your game with which the player can interact. For instance, you could add buckets, bottles, collectibles, etc. (anything you can find on the Asset store) and the player can hit them by shooting a projectile at them using their weapon (which you should decide on as well).
2. Don’t worry about making objects break apart (similar to the in-class example). Simply pushing the prob away from the player should be enough for this interaction.
3. As described below, you should use a different projectile for these interactables.

### Shooting Projectiles (4pts)

1. Depending on your game idea, implement a shooting mechanism consistent with your game design. Shooting projectile should be the key mechanic of your FPS game. You don’t need to kill something in your game to meet this requirement. Think outside the box!
2. A reticle, or crosshair, should be used to facilitate aiming.
3. Use a different reticle/projectile for enemies and other interactable objects.
4. Add a Trail Renderer to the projectile.

### Game Levels, States, and UI (9pts)

1. You should have at least three different levels in your game. Once a level is completed, the game should proceed to the next level, until all levels have been completed. Each level should have at least one new thing (e.g., a different type of enemy/collectible item, a different level design with more blocks, etc.). Level difficulty should progressively increase with the first level being the easiest/simplest.
2. The “Game Over” state should play an appropriate sound effect and should display an appropriate “Game Over” message for 2 seconds and then should reload the same level. The same level should be reloaded until the player beats the level, in which case the player should move on to the next level.
3. The “Level Beat” state should also play an appropriate sound effect and should display an appropriate victory message (“You Win!”) for 2 seconds and then should load the next level, until the final level has been played. When the final level (Level 3) is beat, the game should not automatically attempt to load a new level (because there is none).
4. The UI should display both the Player Health and Score (# of enemies shot), both of which should reset when a new level is loaded. A timer is optional. The UI elements should be properly aligned and formatted.

### Game Polish (5pts)

20% percent of your grade will come from how polished your game is. Game polish refers to the look and feel of the game and reflects how much effort you’ve put into making the best game possible given the requirements. For the most part, this is a subjective quality of your game; you’ll know it when you see it. Here is the scale that will be used to assess the polish of your game:

**Excellent:** presents a new game environment not used during in-class demos but is able to use the same mechanics; goes above and beyond the requirements to improve game mechanics; provides aesthetically pleasing look; adds extras when applicable to make the game interesting/innovative; uses realistic behaviors and appropriate graphics instead of simple shapes.

**Satisfactory:** uses the same game theme as the one used during in-class demos; meets the requirements; graphics look good but can be more polished; no extra effort to make the most interesting/innovative/different game; uses simple shapes sometimes to meet the requirements

**Half-baked:** one or more requirements is missing; problematic game mechanics/weird controls/behaviors; graphics/colors/assets don’t look very good; no extras; gets away with simple shapes for the most part; doesn’t convey an effort to produce the best game possible

**Dull:** bare minimums in all aspects; multiple problems with mechanics/graphics/assets. No effort to make the game look and feel good at all.

## Submission Requirements

1. Complete the last page of this document.
2. Publish your game for WebGL.
3. Upload your WebGL build to Unity Play, itch.io, or another website (similar to MCA1). **Make sure it’s playable**.
4. You should also submit a zipped folder of your **Unity Project** (**not** the WebGL build). Please delete redundant assets as they will increase file size. When zipping your Unity Project, include the following folders **only**:

Text

Description automatically generated

Simply, go to your project folder (you can open it while in Unity). Then select these three folders and zip them. This is different from zipping the entire Unity folder (Unity files aren’t included in this method, which will mean smaller files). This folder isn’t expected to be very large (at 20-50 MBs at most). Use the following naming convention for the file: MCA3\_LastnameFirstname

This will give you a compressed file with the zip extension. Because we are using GradeScope for handling submissions, there is a little catch: it automatically unzips zipped folders! To work around this limitation, **we ask that you upload this zipped folder to Google Drive and provide a public link to the zipped folder.**

1. **Attach all your script files to your submission separately, similar to previous MCAs. The number of script files will depend on how you structure your game. All must be attached separately outside the zipped folder! Failing to do this may result in a grade of 0.**
2. Complete the last page of this document and submit your High-Concept Document as a PDF. Delete all the prompts in this document when submitting it.
3. Submissions will be handled through GradeScope, so you may need to follow along with their custom instructions. Still, you must ensure that you submit all required files.
4. In summary your GradeScope submission will include the last page of this document completed and saved as a PDF and all your script files attached separately to your submission. **Do not combine the PDF and script files into a zipped folder!**

**Failing to meet submission requirements will result in up to 25% penalty above and beyond other point deductions.**

## High-Concept Document

**<your PDF should contain only this page!>**

Before you develop your game, take some time to conceptualize your vision for the game by answering the following questions:

### Concept

What is the core concept of your game? What is the theme?

### Goal and Description

What is the goal of the player? What is the game challenge (i.e., collect all gems before the time is up?)

### Core Mechanics

What are the core game mechanics implemented in your game?

### Controls and UI

How do we control the player? Are there other actions other than moving the player around the world?

What are the UI elements implemented in your game? What do they do/mean (e.g., the centered textbox keeps track of time)?

### Levels

Provide a description of what is different in each level of your game here.

### **Link to your WebGL Game**

### Provide the link to your playable game here. If we cannot play you game at this link, you will lose points.

### Zipped Project Files

Insert the **public** link to your Drive folder containing the zipped project files as instructed earlier.

If we cannot access the files while grading, you will lose points up to 50%. Therefore, please do make sure the link is public (meaning anyone with the link can access it). Also, avoid making changes to the folder after submission.