Programming

1. **Programming Language**
    - C#
2. **Game Engine**
    - Unity Game Engine (**Unity Version 2023.3.13f1**)
3. **Programming Practices**
    - **The version of Unity Game Engine that will be used throughout the development of the game is the 2023.3.13f1. Its very important that every member of the team has the same version of Unity for compatibility reasons.**
    - Commenting is a very essential programming practice for the purpose of every programmer to understand the code that is used for the game's functionality. Commenting is also very useful to know what each line/block of code's purpose for the game's functionality, so in the case in the distant future where that line/block of code has to be revisited for updates or modifications. Please refrain from the practice of not commenting code.
    - Programming script naming has to be clear and descriptive for the purpose of knowing what that programming script does for the game's functionality. For instance, if there's a script that is in charge of camera controls, the preferred naming convention for that script is CameraController, the two words stuck together with no spaces and the first letter of the word is capitalized. Please refrain from the practice of naming the script files as Script1, Script2, and Script3 for the reason is that naming convention is not descriptive, and it will be confusing as the development of the game progresses.
    - Make sure that the variables (int, float, char, etc.) in the script have a clear and descriptive naming convention for the purpose of knowing what each variable's purpose is in the script. For instance, if a variable is for player movement speed, then the correct naming convention is Speed or MovementSpeed. Please refrain from using the variable naming convention of a, b, c, d because its not very descriptive or clear of what each variable's purpose is.
    - If there's a need of making necessary changes to the script, always comment over the old version of the code in case if the new script is not working as intended which will lead to the use of the old version of the code until the new script is functional. Refrain from the practice of deleting previous versions of scripts for the purpose of using old code for guidance for new code or for the purpose of reusing the old code.
    - An old version of a script can be stored can be stored on GitHub for archiving purposes and can be revisited in a later time.
    - Have all the scripts be in one folder for the purpose of having the scripts being easy to find. In the future, the number of scripts will keep increasing as the game gets more complex, the best recommendation is by having subfolders for Player folder that holds all the player scripts, Enemy folder that holds all the enemy scripts, etc. Please refrain from placing scripts in different folders for the reason of not wasting time looking for a script instead of working on it.

- To prevent any duplications of Models, Animations, and Scripts, the Prefab functionality will be used throughout the game's development. The Prefab functionality has the benefit to store all of the components of asset, so for instance if the game has multiple levels, instead of re-entering all what is done when it comes to programming, animation, models and more into the engine. The Prefab will save so much time because its an asset that contains all of the scripting, 3D-Model, animation, and sound will be already set up for the asset. Simply just click the prefab, drag into the scene, and it works. There will be a folder specifically for prefabs.
- Any scripts written in Unity Game Engine must be uploaded to GitHub for backup purposes. In any case of hardware failure, loss of data, or any other extreme circumstances that prevents from accessing the game, uploading files to GitHub is essential for having a backup for those files. This is a very crucial rule, if there is any risk of data loss by any circumstance, it will halt the process of the game's development. **THIS IS ONE OF THE REASONS WHY WE UPLOAD THINGS ON GITHUB. FAILURE TO DO SO WILL COST US A LOT OF TIME.**

4. **File Naming**
    - File naming system is a very important practice during the development of the game. 2D/3D Art, Sound, Music, UI Elements, and Programming Scripts are supposed to be named in a very clear and descriptive way so that the programmers who will putting everything in Unity Game Engine, have idea what each asset file consists of. For instance, if there are assets for the enemy, the correct way of naming the assets is like this:
        - EnemyModel
        - EnemyAnimationRun
        - EnemyAnimationAttack
        - EnemySound
        - EnemyMusic
        - EnemyScript
    - Please refrain from this practice:
        - Sound1, Sound2, Sound3
        - Model1, Model2, Model3
        - Script1, Script2, Script3
5. **Filing System**
    - Please have every asset organized in its perspective folder, for instance, if an asset consists of sound, then it must go to the Sounds folder, if an asset consists of 3D Models, then it must go to the 3D-Models folder. The purpose of the filing system is to make it easy to find an asset, it will save time and effort which rather be instead used for the game's development than looking for an asset that is pretty difficult to find.
    - The filing system works like this
        - 3D-Models -> Models folder
        - Animation -> Animations folder
        - AnimationController -> AnimationController folder

    - 2D-Textures -> 2D-Textures folder
    - Prefabs -> Prefabs folder
    - User Interface (UI) -> User Interface (UI) folder
    - User Experience (UX) -> User Experience (UX) folder
    - Sounds -> Sounds folder
    - Music -> Music folder
    - Scripts -> Scripts folder
- The Unity Game Engine files will be on GitHub for easy access for every member of the team and the files will be updated regularly so everyone member of the team gets access to the most up-to-date version of the game.
- **PLEASE HAVE EVERY ASSET UPLOADED ON GITHUB FOR THE PURPOSE OF ARCHIVING AND DATA BACK UP. FAILURE TO DO SO WILL COST US A LOT OF TIME.**

6. **File Formatting**
    - These are the file formats:
        - 3D-Model File Format -> .FBX (for instance, PlayerModel.fbx)
        - Animation File Format - > .FBX (for instance, EnemyRunAnimation.fbx)
        - 2D-Texture File Format -> .PNG (for instance, WallTexture.png)
        - Sound File Format -> .WAV (for instance, PlayerDeathSound.wav)
        - Music File Format -> .WAV (for instance, MainMenuMusic.wav)
        - User Interface (UI) File Format -> .PNG (for instance, PlayerHealthBar.png)
        - Script File Format -> .CS (for instance, PlayerController.cs)
7. **List of Programming Scripts Needed (Subject to Change)**
    - **Player Controller Script** -> in charge of player movements and player controls.
    - **Player Health Script ->** in charge of the player's health and how it reacts when the player collides with obstacles and the enemy of the game. This class will inherit from the **Enemy Attack Script** and **Obstacle Script**.
    - **Player Death Script** -> in charge of when the player runs out of time while navigating the maze. The script should be able to reset the entire level and reset the timer by inheriting from the **Level Manager Script** and **Timer Script.** This script should be able to disable **the Player Controller Script** for a specific amount of time until the level reloads**.**
    - **Player Interact Script** -> in charge of the event when the player interacts with an object in the game. Inherits from the Player **Controller Script** and **Level Exit Script**.
    - **Camera Controller Script** -> in charge of the camera movements because this game is first person where the player controls the camera movements with the mouse.
    - **Enemy Controller Script** -> in charge of the enemy movements, and behaviours like chasing and attacking the player.
    - **Enemy Attack Script** -> in charge of the event when the enemy attacks the player.
    - **Obstacle Script** -> in charge of the obstacles that the player will encounter and determines how the obstacle will affect the player if the player collides with the obstacle. Inherits from **Player Controller Script** and **Player Health Script.**

- **Timer Script** -> in charge of the timer that the player is going to race throughout navigating the maze.
- **Objective UI Script** -> in charge of displaying the task on the screen to remind the player what the object of the game is.
- **User Interface (UI) Script** -> in charge of the User Interface's (UI) different behaviours depending on what the player is doing in the game. This script will inherit from the **Time Limit Script** so we can display the timer on the screen. This script will inherit from the **Objective UI Script** to display the objective on the screen. This script will also be in charge of displaying if the player won or lose, inherited from the **You Win/Lose Script**. This script should also display 'Wrong Classroom' if the player arrives at the wrong classroom, this will inherit from the **Room Script.**
- **User Experience (UX) Script** -> in charge of the menus that the player will interact with in the game like the main menu, the pause menu, and the setting menu. It will also be in charge of the actions that these menus once the player presses a button on these menus.
- **Map Script** -> in charge of displaying the map of the maze which the player will use to navigate the maze. The map script will have to display the level layout and able to show the current location of the player.
- **Level Manager Script** -> in charge of managing multiple levels. it's the script that will take the player to the next level in the game once they reach the correct room before the time runs out.
- **Level Exit Script** -> in charge of the event when the player interacts with the object that will end the current level and it inherits the properties from **Level Manager Script** to send the player to the next level. This class will inherit from the **Room Script,** so the room that is selected will become the level exit.
- **Level Reset** -> in charge of restarting the level if the player loses.
- **Room Script** -> in charge of selecting the room that the player will have to search for while navigating through the maze. The program should also be in charge of determining if the player is at the right classroom or not.
- **You Win/Lose Script** -> in charge of determining if the player won or lost the game. The script determines if the player lost is when the timer runs out, which will be inherited from the **Timer Script.**
- **Game Over Script** -> in charge of ending the level once the game is lost. This script inherits from **You Win/Lose Script**. Once this script is initiated, it spawns the enemy at the position of the player who instantly "kills" the player, this sequence will inherit from the **Enemy Spawn Script**, **Enemy Controller Script** and **Enemy Attack Script**. This script should also disable the camera controls and camera should play an animation where it automatically turns 180 degrees to face the enemy who will "kill" the player instantly. This sequence will inherit from the **Camera Controller Script.** This script inherits from **Level Reset Script** to initialize the level reset operation.
- **Enemy Spawn Script ->** in charge of spawning the player behind the player once the game is lost. Inherits from **You Win/Lose Script.**

- **Sound Controller Script** -> in charge of the sound effects that will play in specific events that is completely dictated by the player's actions in the game.
- **Music Controller Script (Not Sure About This One)** -> in charge of playing the appropriate music depending on what is going on in the game, whether the time is running out, whether the game is over, etc.
- **Difficulty Setting Script** –> in charge of the difficulty setting of the game depending on the difficulty that the player has selected before the start of the game.

Gameplay/Mechanics

1. **Controls**
    - **Mouse –** look around
    - **W ->** Forwards
    - **A ->** Left
    - **S ->** Backwards
    - **D ->** Right
    - **Space ->** Jump
    - **E –** Interact
    - **M –** Show/Hide Map
2. **Objectives and Tasks**
    - Navigate the maze of Mackenzie Chown and find the correct classroom before the timer runs out.
    - The player will be navigating through the maze while be chased by the enemy. The player should avoid being caught by the enemy.
    - If the enemy catches the player while navigating the maze, the enemy will chase the player and attack by throwing pocket watches at the player.
    - The player will also encounter obstacles like custodians, professors, and teaching assistants that the player can bump into and affects the player's vision and movement speeds. The idea is to avoid the obstacles because if the player collides with the obstacles, the enemy can catch up with you.
3. **Win State**
4. **Fail State**
    - If the player fails to reach the classroom on time, the player will encounter the enemy who will spawn behind the player and "kill" the player instantly.