Persistent Storage Documentation
By Sabastian Na

Define a "save" state for your program/game and demonstrate "load" of that state, e.g.

> The save state for my game is to save the position of the player so that it can be loaded whenever. To save in the game, the player must walk onto the green tile to save the current player's position on the game's map. Having the player move to a different area to save in the game can be classified as changing the game state by altering where the player can load in from/at. To load the previously saved state, the player can click the "load" button on the screen and it will load the player in the saved state. If there is no save file then no load will happen. There is also a "reset" button on the screen that resets the world and deletes the saved file. This save/load mechanism works after the game is closed as it uses custom binary files located locally. Data would be encrypted and decrypted accordingly.

Brief write-up on minimum of 3 storage mechanisms you considered, pros/cons, and why you chose the solution you did

> I considered using the built-in system that Unity offers, a database, or cloud storage to store the game's load/save data. What I decided to do is to use custom binary files to store data. The built-in system in Unity that uses playerprefs is simple to utilize however it is limited in capabilities and cannot save a player's location. The database is secure however it is more complicated to set up and is not needed for the scale of the game that was developed. Cloud storage allows for portability and online connection however the game being developed does not need that complexity to function as well as it should be available offline. What is great about using custom binary files is that they can be optimized for the game and provide good security against data tampering due to binary. Custom binary files were also simpler to implement than the other suggested storage mechanisms.