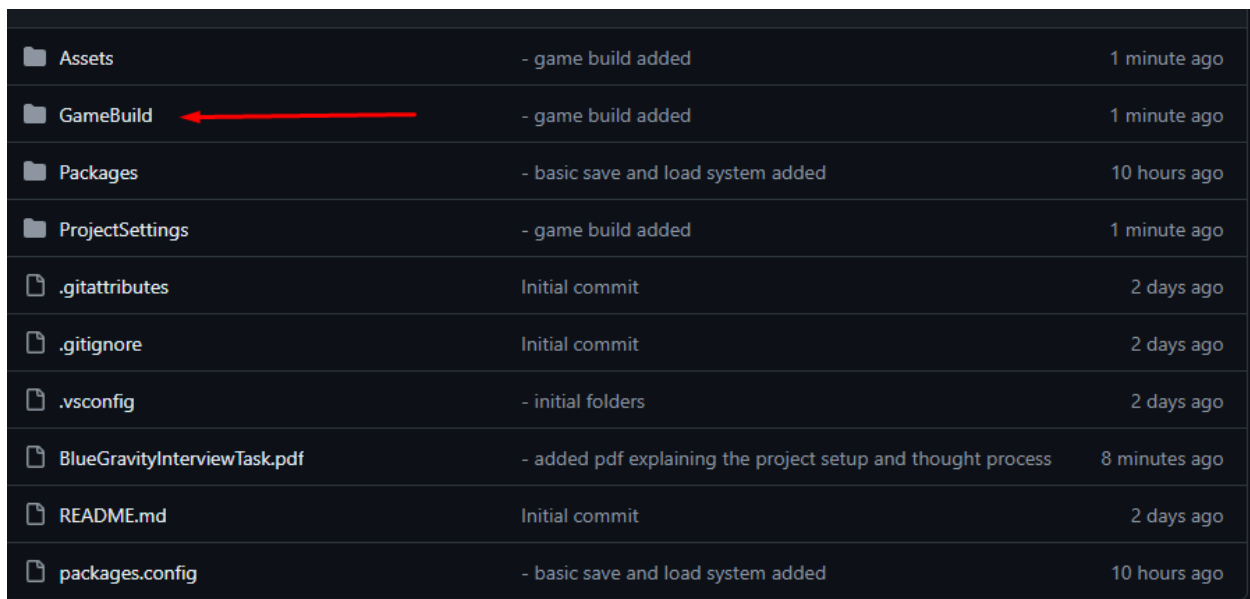


Blue Gravity - Unity Programmer Task

Bojan Vuchevski

During the past 48 hours I implemented all the necessary modules needed for the task. With the time I had I couldn't add the suggested additional features such as world building, because I believed my time was more valuably spent working on the other features. I think I demonstrated my design skills well through my solo projects, notably [King Wizard of the Forest Kingdom](#).

The Game build folder is in the root for the GitHub project itself.



Assets	- game build added	1 minute ago
GameBuild	- game build added	1 minute ago
Packages	- basic save and load system added	10 hours ago
ProjectSettings	- game build added	1 minute ago
.gitattributes	Initial commit	2 days ago
.gitignore	Initial commit	2 days ago
.vsconfig	- initial folders	2 days ago
BlueGravityInterviewTask.pdf	- added pdf explaining the project setup and thought process	8 minutes ago
README.md	Initial commit	2 days ago
packages.config	- basic save and load system added	10 hours ago

Note, most of the classes and functions themselves have written comments and documentation, explaining the logic behind the code.

There is only one scene in the project: Assets/Game/Scenes/GameScene.scene

I had added some helper utility classes that I thought would be necessary for developing a clean code and class structure. They are in the Assets/Game/Scripts/Patterns folder. Most notably the `UnitySingleton` class and the `EventManager` class.

UnitySingleton is used for creating unity components that are easily accessible throughout the game, ensuring that only one instance of a particular class exists at any given time.

The **EventManager** class, on the other hand, is designed to facilitate communication between different parts of the game through a publish-subscribe pattern.

The logic for the Inventory system is in the Assets/Game/Scripts/Inventory folder. The base logic for the Inventory data structure is in the **InventoryController.cs** class. Right now, the inventory data is structured in a basic array, with fixed slots. I went with a simpler approach, because its easier to move up to a more complicated data structure like a Dictionary. Other classes used are:

- InventoryItemSlot – for keeping track of fixed inventory slots that hold items
- InventoryItemSO – Scriptable Object that holds Item data
- ItemDatabaseSO – Scriptable Object that holds all of the Items, like a central database for all of the classes to use.

The UI for the inventory is controlled through the **InventoryView.cs** component. It listens for events through the EventMesenger from the InventoryController class. The class holds logic for dragging and dropping elements in the inventory as well as always rendering the current state of the inventory.

ToolTipController.cs utilizes the Singleton pattern mentioned above, it can be accessed by other classes easily and is used to generate a tooltip message for all the items showing off the item details.

UserControls.cs is a component that manages user input that is sent through the keyboard. It listens for input through the Update method and then sends events accordingly. This makes it easier to manage controls and map keys as we like for the game.

PlayerManager.cs and **Player.cs** are both classes to manage the Player character, for instance the movement and collision with other objects in the world.

PlayerPrefsManager.cs is a wrapper class for the PlayerPrefs Unity class. It manages saving and loading inventory data from local storage. The data is loaded at the start of the game and is saved when the application is closed.