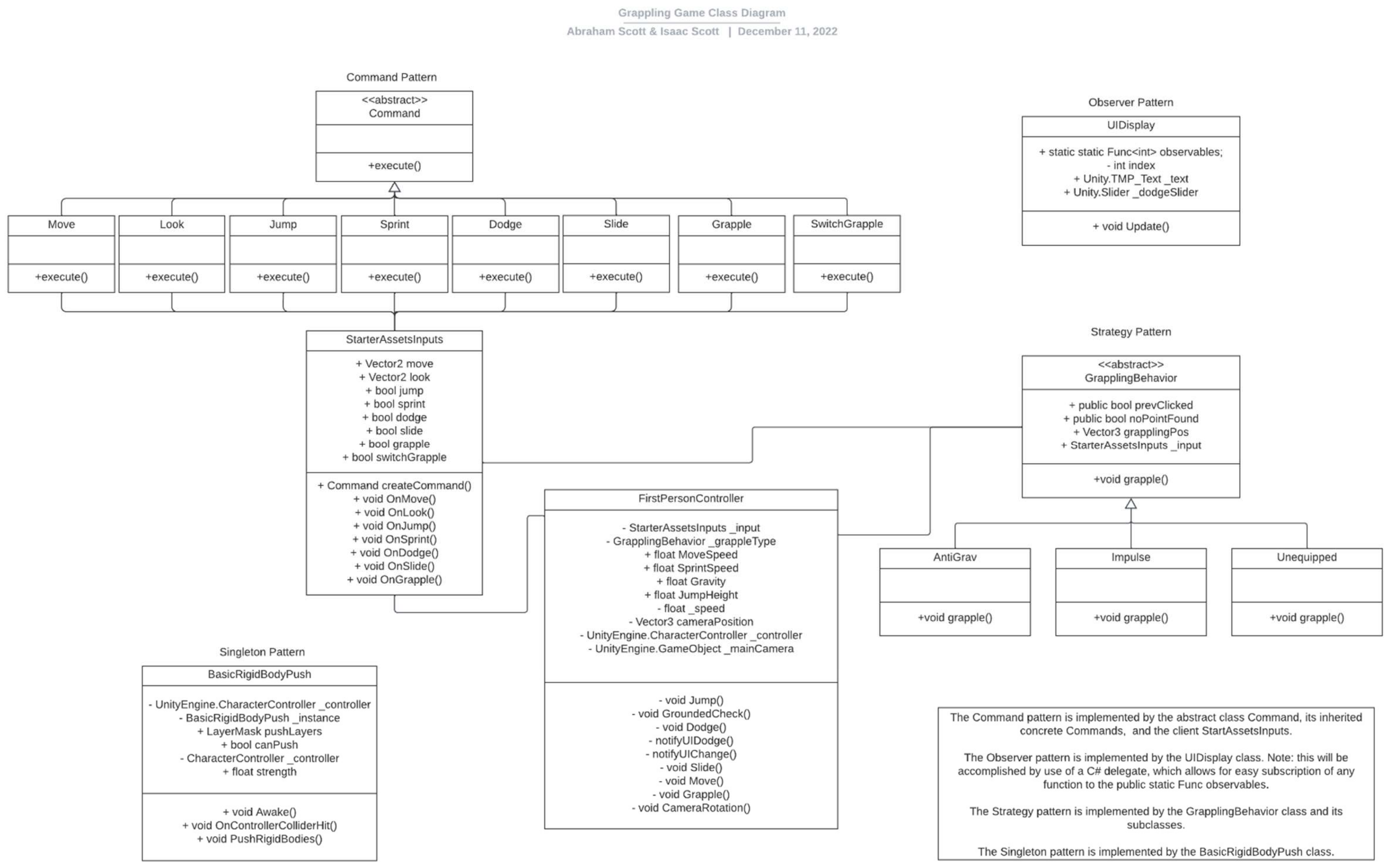
Project 7

Final Project Report:

1. Abraham Scott & Isaac Scott: Grappling Game
2. The final state of the project includes basic and advanced movement for the player, camera movement, interactability with platforms, reset conditions, grappling hooks, a and a simple UI. Troubles with the base behavior of grappling hooks caused us to only be able to deliver two of them as opposed to the promised three. Furthermore, the promised GameManager class was not necessary with the simple scene we designed, so the Singleton pattern was instead used on the BasicRigidBodyPush class.
3. Project 7 UML Diagram



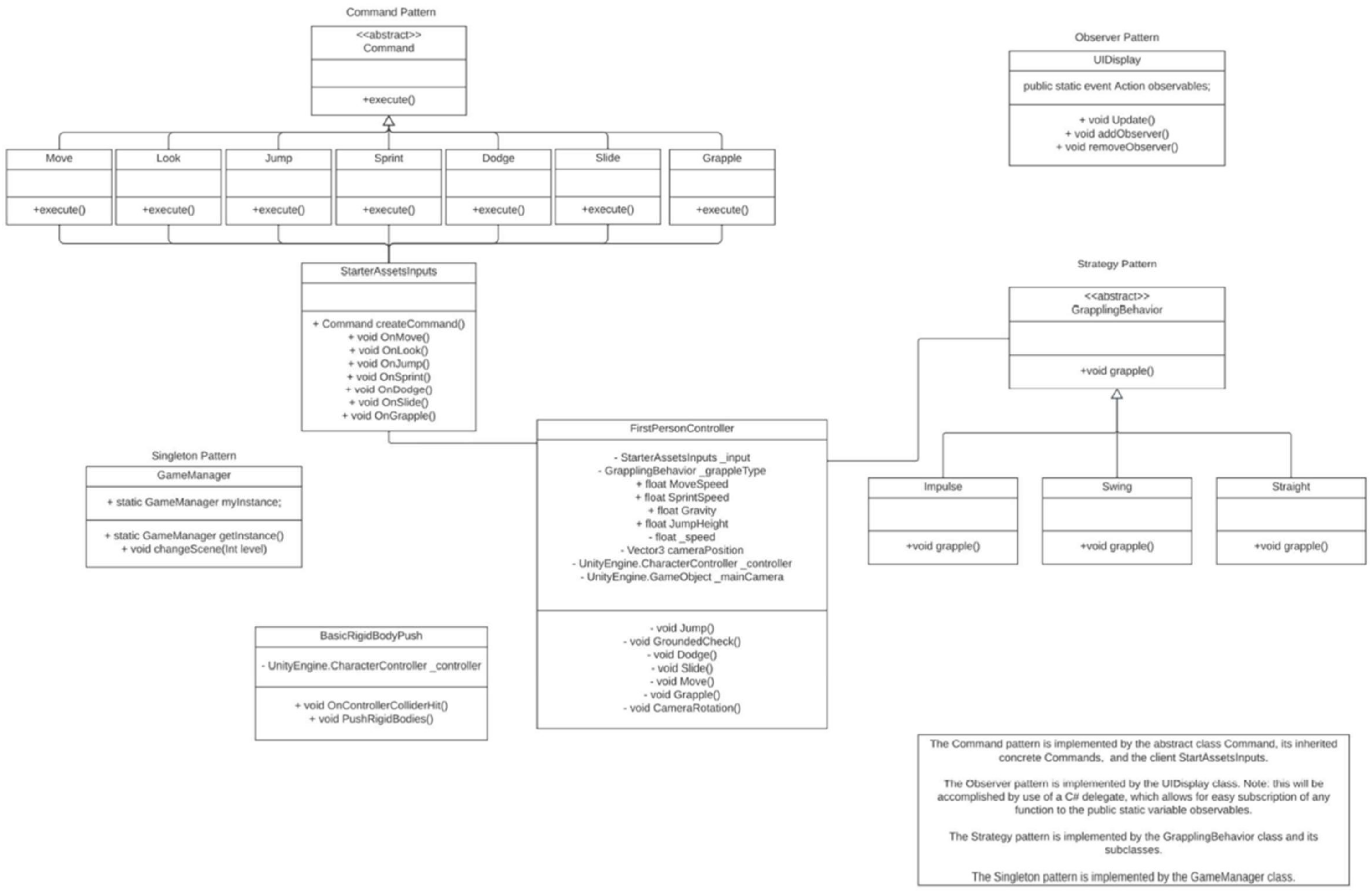
The Command pattern is implemented by the abstract class Command, its inherited concrete Commands, and the client StartAssetsInputs.

The Observer pattern is implemented by the UIDisplay class. Note: this will be accomplished by use of a C# delegate, which allows for easy subscription of any function to the public static Func observables.

The Strategy pattern is implemented by the GrapplingBehavior class and its subclasses.

The Singleton pattern is implemented by the BasicRigidBodyPush class.

Project 5 UML Class Diagram:



The Command pattern is implemented by the abstract class Command, its inherited concrete Commands, and the client StartAssetsInputs.

The Observer pattern is implemented by the UIDisplay class. Note: this will be accomplished by use of a C# delegate, which allows for easy subscription of any function to the public static variable observables.

The Strategy pattern is implemented by the GrapplingBehavior class and its subclasses.

The Singleton pattern is implemented by the GameManager class.

The biggest change from Project 5 is the removal of the GameManager class, and adding the Singleton pattern to the BasicRigidBodyPush class. The UIDisplay class did not need the functions addObserver or removeObserver because the C# Action delegate object allows for doing this easily. Beyond these two big changes, some classes (FirstPersonController, BasicRigidBodyPush, GrapplingBehavior, UIDisplay) got more attributes added.

4. The project uses the Unity Engine to display images and receive inputs. At the beginning of the project, a lot of third party code was sourced from here: https://assetstore.unity.com/packages/essentials/starter-assets-first-person-character-controller-196525

This source provided some of the base code for the classes FirstPersonController, BasicRigidBodyPush, and StarterAssetsInputs. This code was built upon and improved heavily.

Other than that, the following resources were used a lot during the development of the project:

https://gameprogrammingpatterns.com/

https://www.youtube.com/watch?v=V75hgcsCGOM&list=PLB5_EOMkLx_VOmnIytx37lFMiajPHppmj

https://www.youtube.com/@OneWheelStudio

https://www.youtube.com/watch?v=OecJvh8Zvc4&list=PLKERDLXpXl_hN_3tPJdLgjWJ12VH6igy1

5. One issue that our team encountered in the design and analysis of the project was the addition of the Singleton pattern. In the end we decided to use this pattern with the BasicRigidBodyPush class, but initially we were planning on using it with the GameManager class. When we decided to remove the GameManager class, in order to still fulfill the 4 patterns requirement, we used the Singleton pattern on the BasicRigidBodyPush. Serendipitously, this was a good application of the Singleton pattern in the end.

   One element that turned out very well was the implementation of the Command pattern. The built-in Unity Engine input system worked very well with the Command pattern. This made inputs to the player very easy to do, which ended up saving us a lot of time and effort.

   Another element that turned out positively was how we implemented the Observer pattern. C# has an object called a delegate that can hold a reference to methods. The observables variable in UIDisplay allows for subscribers to add methods at any point they want. The Update() method in UIDisplay is always checking for subscribers every frame, which allows for really quick updates to the UI.