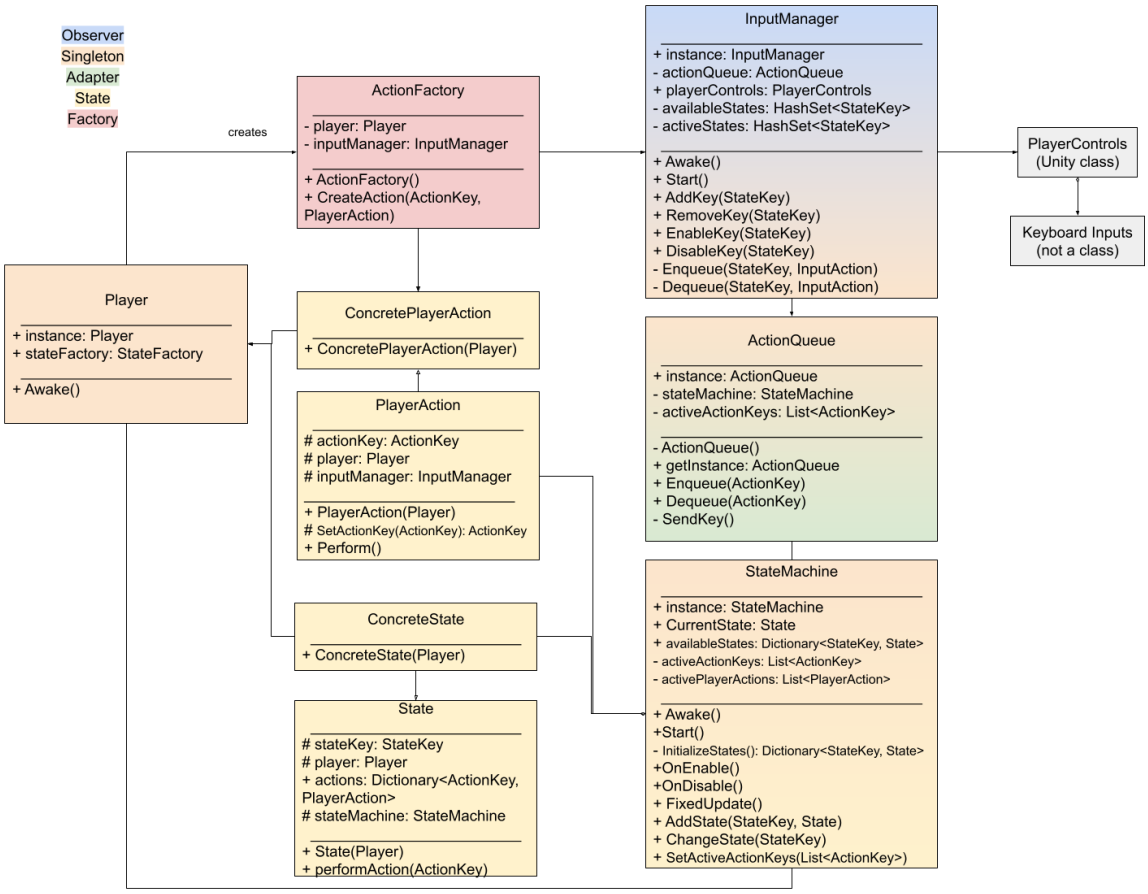
I. **Project Name:** Dino Doomsday: 2D Platformer Game
Team Members: Erica Lowrey, Thomas Deaner, Emma Goodwill

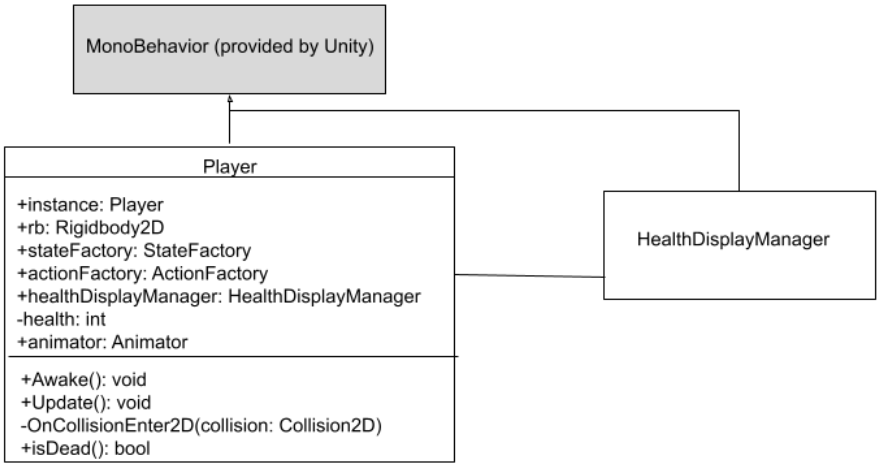
II. **Final State of System Statement**

We now have a dinosaur character that the user can control through keyboard input. The user navigates through a world with three types of obstacles: shaking platforms/unstable terrain, meteorites, and enemy dinosaurs. Enemy dinosaurs and meteorites can reduce the health of the dinosaur character, and the state of this health is displayed to the user via three hearts in the top left-hand corner that follow the dinosaur character throughout the levels. All of this gameplay is accessible through the main menu. We took out the cut-scene features showing the aliens picking up the dinosaur at the beginning of the game because of the time it takes to do such animations. We also decided not to have customizability for the dinosaur character as a result of this requiring significant time to adjust the art of the game. We also changed the game from 10 discrete levels to 1 continuous level, as this continuous level already increases in difficulty, so having discrete levels is just added friction for the user in their completion of the game. Not to mention, having health reset at the beginning of each level, significantly decreases the difficulty of the game for the user.

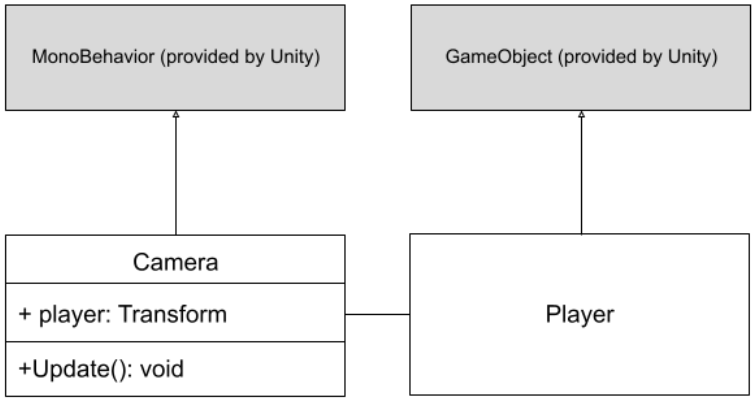
III. Final Class Diagram and Comparison Statement

A. Player Class Diagrams

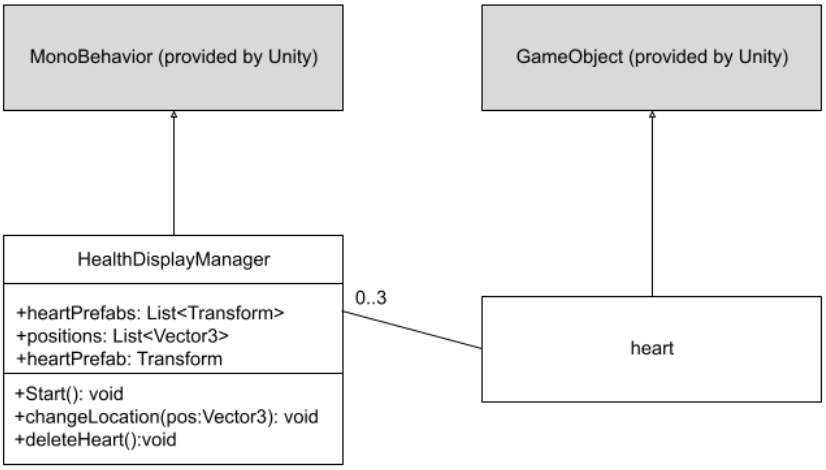




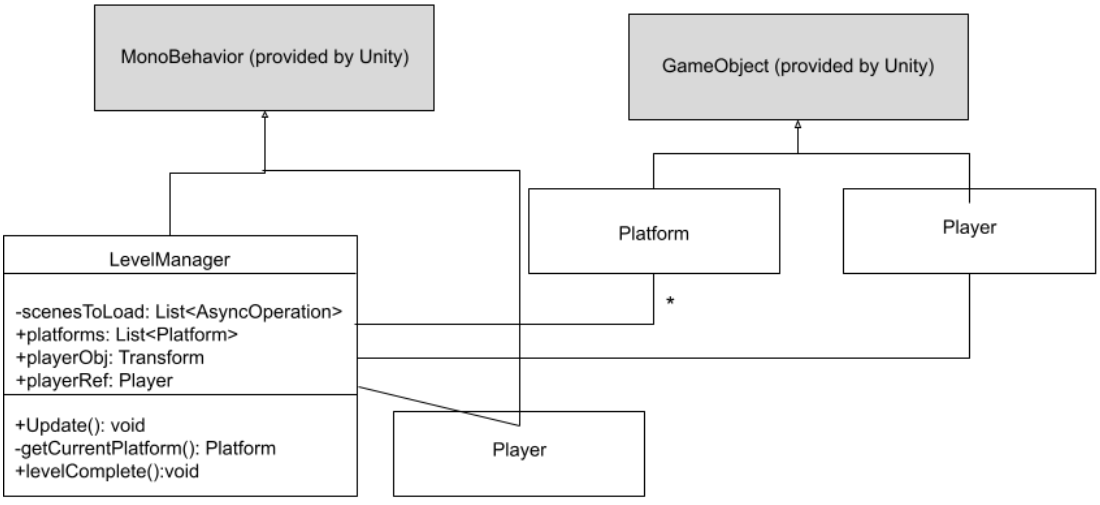
B. Camera Class Diagram



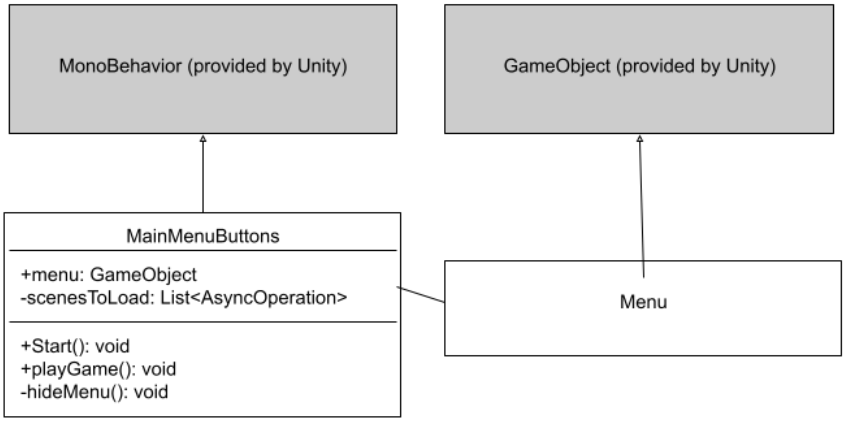
C. Health Management Class Diagrams



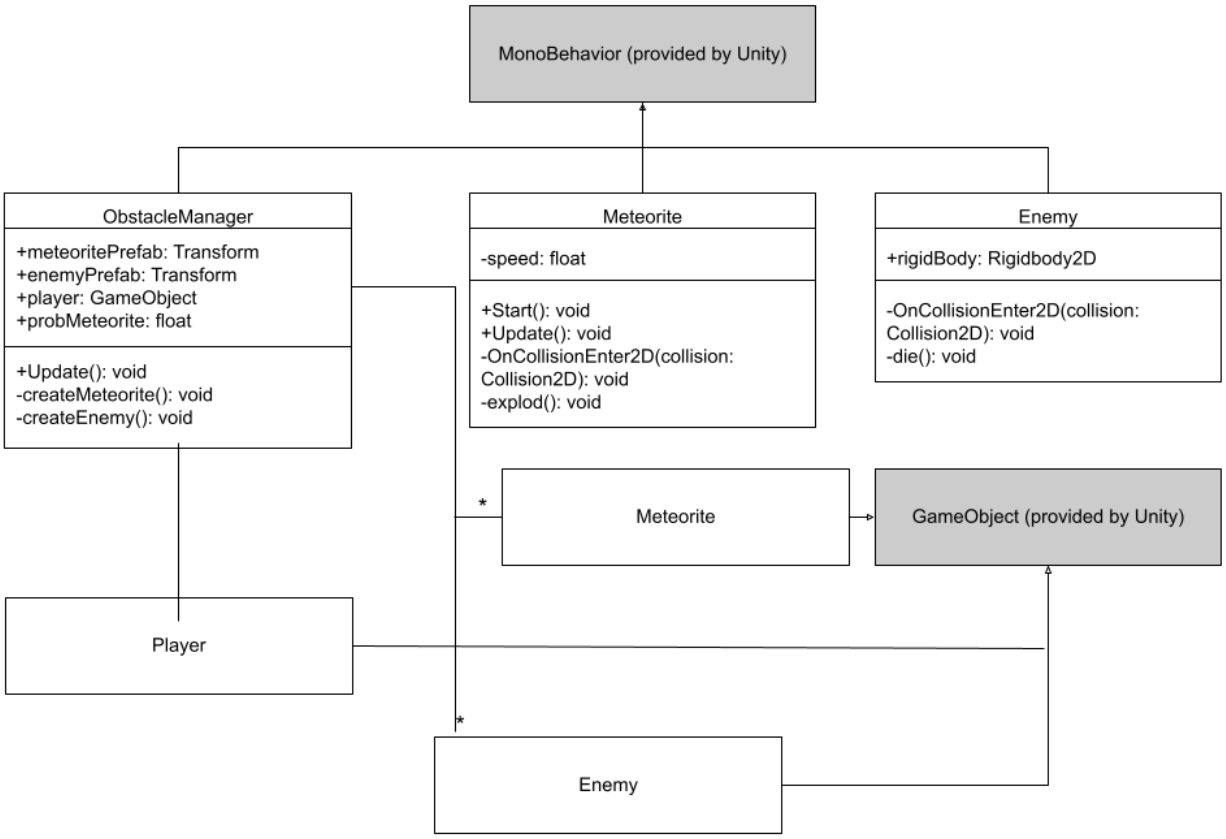
D. Level Manager Class Diagrams

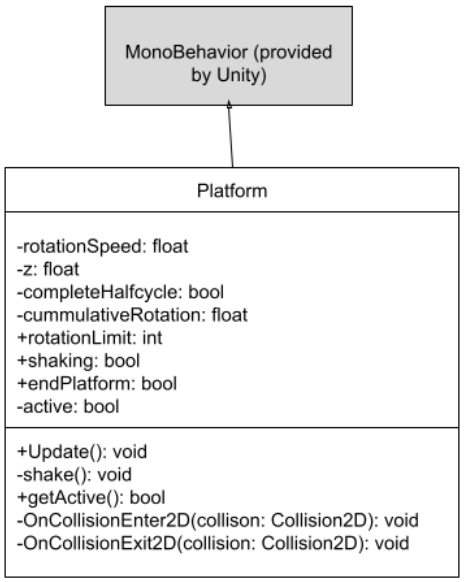


E. Main Menu Class Diagrams

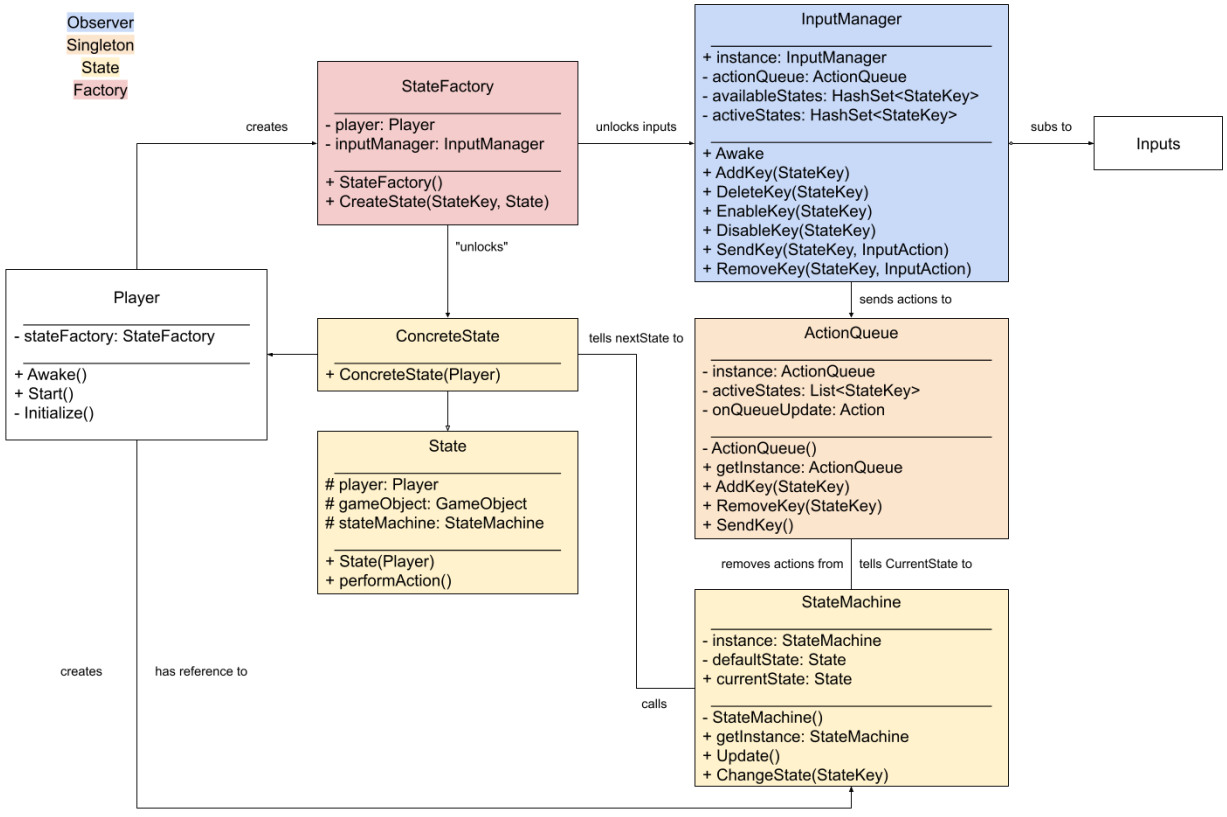


F. Obstacle Objects Class Diagrams





G. Project 5 Diagrams



What has changed between project 5, 6, and 7:

Initially, our class diagram only included the input system and not the camera, health management, level manager, main menu, and obstacle class diagrams. The input system also differed between its final form in that we added classes for Actions and ConcreteActions the user can take. Player also now has an instance of itself. We now use a factory to create actions and states instead of just states. InputManager also has a reference to PlayerControls to manage keyboard input now. From project 6, we have removed all classes and functions associated with character customization. We also removed the attributes and methods associated with enemy movement to instead have a stagnant enemy. We have made it such that platforms can be assigned as end platforms, and once active (meaning they are being stepped on by the user), the game is won.

IV. Third-Party Code vs. Original Code Statement

We used Unity Game Engine to develop this game which allows us to utilize pre-built game objects from which we extend to create each object in this game. This engine also provides a Transform object for each game object through which we can adjust certain aspects of the state of a game object such as position. Unity also provides components to the game objects to use including rigidbody and colliders, through which we overload a couple of associated methods that are called asynchronously (OnEnterCollision and OnExitCollsion) to handle collisions between objects. A tagging system from unity was also utilized to identify which objects were colliding with each other, which we used to direct our handling of such collisions. Unity also allows us to set the method that a button will call on click, which we utilized in the main menu. Unity gives functionality to change scenes, which we utilized in our LevelManager objects.

We programmed the handling of object collisions, obstacle behaviors, scene management, health tracking, button pressing handling, keyboard input, and user state which constitutes the actual functionality of the game. We simply used Unity Game Engine as building blocks for this functionality.

We also utilized the following tutorials/materials as references while programming these features (Note: these are provided in comments throughout the code as well):

- Camera movement reference: https://forum.unity.com/threads/making-camera-follow-an-object.32831/
- C# lists reference: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0
- Reference for loading new scenes: https://www.youtube.com/watch?v=zObWVOv1GlE
- Reference for hiding and creating menu object and loading new scenes: https://www.youtube.com/watch?v=zObWVOv1GlE

- Reference for collision detection: https://www.youtube.com/watch?v=0ZJPmjA5Hv0
- Reference for destroying game object on collision: https://unphayzed.com/2020/08/07/how-to-destroy-an-object-on-collision-in-unity/#:~:text=To%20destroy%20an%20object%20on%20collision%20within%20the%20using%20ty,()%20method%20for%203D%20games
- Random Range reference: https://docs.unity3d.com/ScriptReference/Random.Range.html
- Gradual movement reference; https://docs.unity3d.com/ScriptReference/Transform-position.html and https://docs.unity3d.com/ScriptReference/Vector3.MoveTowards.html
- Rotation reference adapted from https://docs.unity3d.com/ScriptReference/Transform.Rotate.html and https://answers.unity.com/questions/1602053/localrotation-and-transform-rotate.html
- Referenced for on collision exit method name: https://answers.unity.com/questions/1220752/how-to-detect-if-not-colliding.html
- Converting json string to object reference: https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html
- File reading reference:https://support.unity.com/hc/en-us/articles/115000341143-How-do-I-read-and-write-data-from-a-text-file-
- Referenced to add animations: https://www.youtube.com/watch?v=hkaysu1Z-N8

V. Statement on OOAD Process

1. Although keeping the open-closed principle in mind when writing code slowed down our development progress, we were able to write good code and minimize "code smells". Using the principle also made it easier for us to work in iterations, as our code was open to extension, not modification.
2. Using the Single Responsibility principle made sure our code was clean and easy to read. However, using this principle had us making a LOT of classes, and keeping our file directory clean was a challenge in itself.
3. Using class diagrams helped us convey to each other what classes were supposed to do. Instead of describing every single aspect of a class, we were able to say things like "This is an observer class", and we all understood what the class was supposed to do.