Name of Project:

- Goose Hunt

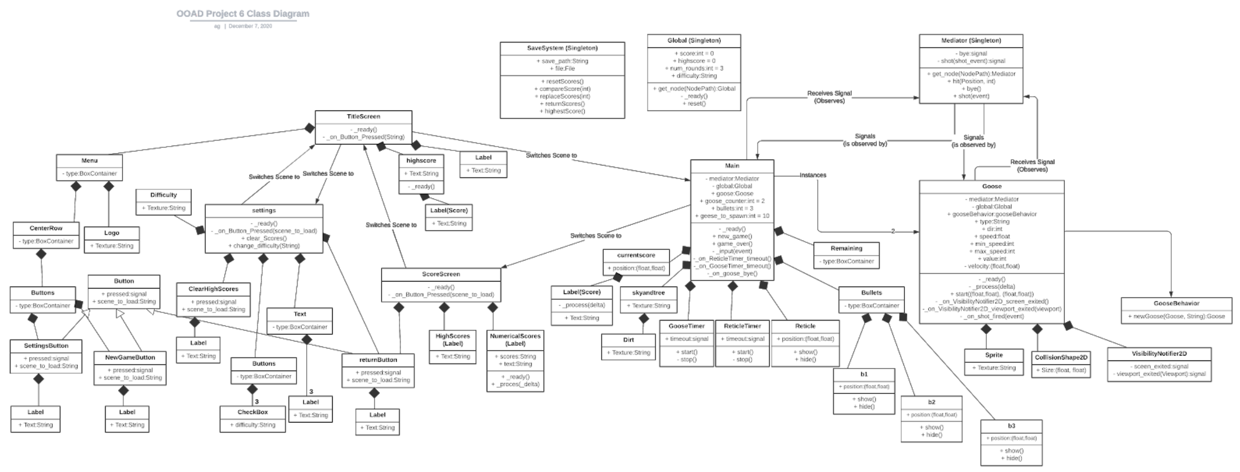
Team Members:

- Caleb Chatham, Andrew Gilfillan

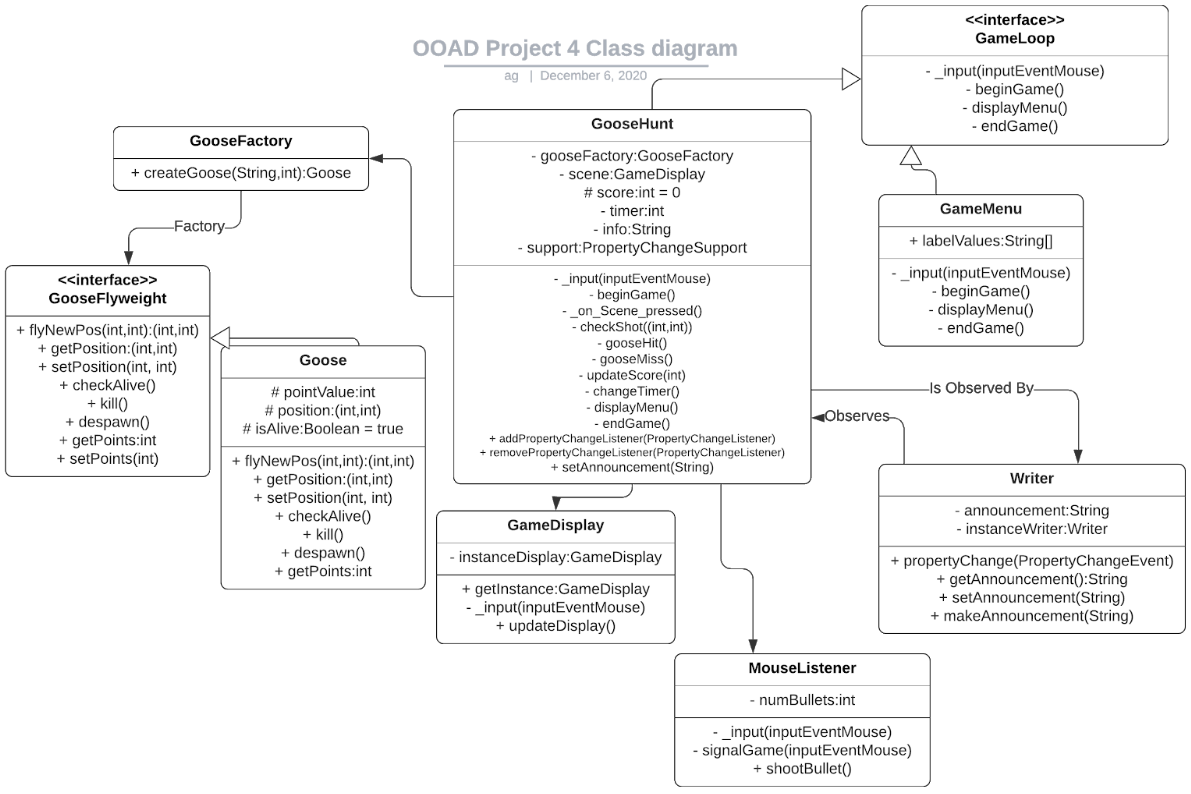
Final State of System Statement:

- Our system is a full-functioning game that is a clone of Duck Hunt, a NES game from the 1980's. In it the player can select a new game. The player then can play the game where they shoot (by clicking on) the geese that appear on screen. The geese spawn 2 at a time and every goose hit increases the player's score. There are 10 geese in a round and the number of rounds can be set by the player in the settings menu, although the default is 3. There are 3 difficulties of geese, easy, intermediate and hard, which become progressively faster and more difficult to hit, respectively. Once the game is over, the gameplay loop ends, the geese stop spawning and the game is able to compare the player's score with the top 10 previous scores, if the player has earned enough points to have a score, that score will be saved by the game, thus updating the high-score list. In Project 4, we had wanted to implement a second mode called "crazy" mode where the number of geese that would spawn would greatly increase, their flight patterns would become faster and far more erratic, their sprites would change, geese could re-enter the scene, geese could shoot at you, etc. Sadly, due to time constraints this was unable to be implemented, as it was better to get the regular version of the game running fully in time instead of devoting development time to a joke mode that may not be able to be completed.

Final Class Diagram and Comparison Statement



- Since our game uses the Godot Game Engine, we inherently utilized the Composite design pattern, as Godot structures all files in the game (whether it's scripts or sprites) as a collection of nodes and scenes into a tree hierarchy. Scenes themselves are a collection of nodes in their own sub-tree hierarchy. Next, we utilized Signals, Godot's version of the Observer pattern to allow for our classes/nodes to notify other "listening/observing" classes/nodes, we used this direct signaling system for communication between nodes within a given scene. Next, we utilized a Mediator design pattern to handle all inter-scene communication, wherein Mediator would be called upon by a given root node of a scene, and it would emit the necessary signals, thus handling all communication between the nodes themselves and allowing our root nodes to not have their script files filled with a bunch of connect() function calls. Next, we utilized the Singleton pattern for Mediator, as well as Global (which deals with global variables and their functionality) and the SaveSystem. However, while these Classes have global access to an instance of them, it is technically possible to create more instances of them as there is not way to make things "private" in GDscript (just like Python), although we ensured not to create new nodes of these classes, only call the instance. Throughout the project, we attempted to program using general object oriented ideals. For example, the buttons and labels used in the project get their base attributes from a parent class through inheritance.



- From these two diagrams, we can see quite clearly that our project has changed quite a bit since we created the Project 4 diagram. First, the entire program is not centralized around a singular Game/GooseHunt Class, as this centralization does not really reflect how things are structured within or function within Godot, where a more tree-like structure exists, and we can switch into the different scenes, from the title screen into the main gameplay loop. We also did not end up utilizing the Flyweight or Factory design patterns, favoring others that worked better with our plans and within Godot itself.

Third-Party code vs. Original code Statement

- Since the project was written in the Godot Game Engine, we utilized some code that is provided by the engine itself, for example the get_node() method, which returns the node instance. Further, in order to learn the engine we went through the basic tutorial within the Godot documentation (https://docs.godotengine.org/en/stable/index.html). Not only did this teach us our way around the engine and give us familiarity with GDScript, the python-esque in-engine scripting language, it also provided guidance on how to approach the game, for example how to implement movement and how to autoload

scripts (which is how you implement Singletons). For the Buttons, the following tutorial was used as a reference (https://www.youtube.com/watch?v=sKuM5AzK-uA&t=1641s). The Godot Engine itself implements a Composite design pattern with its tree structure of scenes and nodes, so, we did not implement the “code” that does this structuring, like adding and deleting children, as it is functionality inherent to Godot. We did however create the nodes, decided their hierarchy, and implemented the node’s functionality, etc. Other tutorials that were used as references were for saving (https://www.youtube.com/watch?v=ygGaN1EOQEA&feature=emb_logo),
Finally, MS Paint was used to create some sprites, and the website LucidChart (www.lucidchart.com) was used to create UML Diagrams.

Statement on the OOAD process for your overall Semester Project

- During the development process of this project we had a variety of positive experiences that were quite rewarding, but also some issues that caused a few setbacks. Speaking of the positive first, Godot, as an engine, inherently makes use of the Composite pattern through it’s nodes and scene tree hierarchy. Not only does this immediately allow us to utilize a design pattern we learned in class, it helped us get great experience with developing other parts of our system (and choosing other design patterns) that work well with the Composite pattern. As previously stated, the development of the project did have some issues which caused us to have some setbacks. First, we had to scrap an entire implementation of factory because the differences between the different difficulties of geese did not warrant separate constructors and the use of the Factory method. We then changed it to a Decorator pattern which, we also found that there were not enough differences between the different kinds of geese (only attributes differed), to warrant the use of the pattern itself. We sadly lost a fair amount of progress to this. Next, although learning Godot was a rewarding experience for the both of us, it did come with a learning curve that slowed down the development of the project, as we had to initially learn how to use the engine itself before any real progress on our project was made.