Ian Thorne
Edwin Silerio
Will Hudson

Our Core Technology:

- For our game, we decided to implement a shop system. The shop system functions like a sort of skill-tree, where the player can spend the gold they accrue in the rounds (either by collecting coins or by killing enemies) to purchase new weapons, armor, boots, and other miscellaneous items. This also required changing our game to function in a round-based format, where levels would be loaded dynamically and the player and AI agents would be loaded into the level dynamically as well--so the shop system could actually be accessed.
- The game is now broken into rounds, which required the creation of a PlayerStatus singleton to store relevant persistent information about the player's stats. It also required the creation of a GameManager singleton to manage switching scenes upon receiving certain signals, spawning entities (the player, enemies, the familiar, and pickups) into the level, updating GUI's, and ending the game.
    - The rounds still operate on a timer, if that timer runs out, the player will lose.
    - We designed a GameManager script that will parse a comma-delimited file named "roundInfo.txt" (in the same directory as the executable or in Project if the game is being run from the Godot editor) to determine which level to load for each round, where to spawn the player, where to spawn relevant pickups, where to spawn the enemies, and what stats the enemies should have. Once the GameManager has parsed through the entire file, the player has beaten every round and won the game.
        - Each line of "roundInfo.txt" is organized as follows:
            1. A string of the path to the level scene to load in
            2. 3 integers representing the player's spawn coordinates
            3. 3 integers representing the sword pickup's spawn coordinates (if the player doesn't have it yet)
            4. 3 integers representing the glider pickup's spawn coordinates (if the player doesn't have it yet)
            5. 1 integer representing the number of enemies to spawn
            6. 1 integer representing the HP of enemies in this round
            7. 1 integer representing the attack power of enemies in this round
            8. 1 integer representing the gold reward for killing enemies in this round
            9. n 3 integer triplets representing the spawn coordinates for the n enemies in this round
        - 3 integer triplets that are parsed as Vector3's are written like "x|y|z".
        - "roundInfo.txt" can be changed without recompiling the game to change the way the rounds play out or modify their difficulty. It will be in the Game directory alongside the executable for the game.

    - We designed a PlayerStatus singleton that took a lot of the player's internal variables representing its state and stored them in itself instead. This allowed us to make the player's stats and current HP persist between the rounds of our game and is how the shop is able to modify the player's stats. It also maintains a list of the items that the player has purchased, for keeping the sub-shops' progress consistent between visits.
        - The main shop screen as well as the pause menu use the PlayerStatus singleton to display the player's current stats.
            - The pause menu can be accessed (in single player mode only), by pressing the escape key. Pressing the "Q" key while in the pause menu will close the game.
    - We were able to route almost all of our signals through the GameManager and PlayerStatus singletons, which made our code a lot cleaner, easier to follow, and easier to work with.
- The shop system starts on a screen which displays the player's current stats as well as buttons to return to the game and switch to different sub-shops which sell different items. Those sub-shops include:
    - A Weapon Shop, which contains an item-tree (starting with the base sword that the player must pick up to complete the first round) and has three offensive stat upgrades, followed by five options for a final weapon.
    - An Armor Shop, which contains an item-tree starting with base armor that raises HP a small amount, followed by another piece of armor that raises HP a little more, followed by two options for a final piece of armor. (It also contains an item called "Cheating," which costs zero gold and gives the player one hundred health. This item is useful for just exploring the game, without needing to worry about losing.)
    - A Boots Shop, which contains an item-tree starting with a base pair of boots that gives the player a third jump, followed by another pair of boots that gives the player a fourth jump and slightly increases their movement speed, followed by another pair of boots that gives the player a fifth jump and further increases their movement speed.
    - A Miscellaneous Item Shop, which contains a bell item that can be purchased to give the player the Familiar AI agent, which will follow them around and collect coins and a potion item which costs ten gold and restores five HP to the player (the potion can be bought as many times as the player has gold for, if they choose to do so).
        - Since the last assignment, the Familiar has changed to no longer collide with things, so the player can't get stuck on it.

- The items in the sub-shop can be moused over to see their name, cost, and a description of what the item does.
- Each of the sub-shops listed above has its own item-tree, which are treated like graphs and obey the following rules:
    - An item is marked “purchased” if the player has picked it up in the overworld (like the base sword) or if they have already purchased it. Items that are “purchased” have a yellow background to indicate this.
    - An item is marked “purchasable” if all of its prerequisite items (the items which have lines going from them to the item in question) are marked “purchased.” Items that are “purchasable” have a red background and an image in full color to indicate this.
    - An item is marked “unpurchasable” if it meets neither of the criteria listed above. Items that are “unpurchasable” have a red background and a darker image to indicate this.
    - Items can be marked “final.” “Final” items obey all of the rules above, with the exception that once the player buys an item that is marked “final,” all other “final” items of the same type are marked “unpurchasable,” regardless of the rules above. Items that are “final” have the keyword (Final) in their description box and they are also typically much more expensive than other items of the same type.
        - For example, if the player buys a final weapon, they can’t buy any more final weapons, but they can buy final items of types that they haven’t purchased a final item for yet.
        - Buying a “final” item only prevents the player from buying other “final” items, they can still buy non-final items of the same type, if any still remain unpurchased.
    - All items can only be purchased once, with the exception of the health potion in the miscellaneous shop.
    - If an item is “purchasable” and the player has the gold, they can buy the item, which will update the player’s stats and update the item-tree.
- The shop system was designed to be modular so it existing shops could easily be modified through the Godot editor, without having to touch any code and new shops could easily be created through the Godot editor, with minimal exposure to code (simply adding a new type value in SubShop.cpp and ShopItem.cpp).
    - All sub-shops within the overarching shop use the same script, which looks at the graph of items inside them and updates it visually to follow the rules laid out above. The sub-shop simply has a Type field exposed to the Godot editor, which determines the type of item sold by that shop.

        - The Type field is used to compare the sub-shop's available items to the list of the player's purchased items that match that type.
    - All shop items use the same script, which exposes a cost to the editor; a list of stat modifiers to the editor for attack, attack range, attack speed, max hp, number of jumps, and max HP; a "final" flag which can be set through the editor; an array of prerequisite shop items that can also be set through the editor; as well as exposed properties for the dark and light versions of the item (for displaying purchasability).
        - When a shop item is purchased, it will use this information to subtract its cost from the player's gold count and apply all of its stat modifiers to the player's stats stored in the PlayerStatus singleton. It also signals the sub-shop to redraw the item-tree, taking into account the fact that it has been purchased.
        - Shop item nodes are also named based on their in-game name, when one is purchased, it observes the sub-shop's Type field to determine which of the player's inventory lists to add its name too, so the sub-shop can be drawn correctly the next time it is visited.
    - While designing the game, it took us a while to get this functionality between scripts set up for the weapon shop, but once we had that figured out, we were able to use the same infrastructure to set up the other shops in considerably less time.
- The shop has a minute timer that will return the player to the next round when it runs out.
- We didn't add any of this functionality to multiplayer, however none of our additions affected multiplayer's current functionality.

Things that went well:

- We planned out all of our goals and started early, which allowed us to finish everything without any stress.
- We communicated effectively, which really helped us design everything and get it done.
- We had reasonable goals and were able to cut back on features when we realized that we were running low on time.
- The code for the shops and shop items was designed to be modular, and once the code was finished, everything fell into place and it was fairly easy to actually create the shops.

Things that did not go well:

- We had difficulty finding times when we could all meet as a group.
- A handful of the small quality of life changes we wanted to make weren't finished, because we didn't leave enough time after finishing our main features.

In the future:

- We would continue to plan everything out and start early.
- We would do a better job of dividing tasks, because some tasks felt larger than others.