The Game

Our game is a 2D Maze game called Treasure Hunter that involves collecting coins/treasure while avoiding obstacles and enemies to achieve the highest score in the fastest time possible. Every coin and treasure chest will add to the player's overall score while collecting seaweed will decrease it. If the player's score falls below zero from collecting too much seaweed, or if the player touches any shark enemies, the game ends and the player loses. Once all the coins are collected, the player may access the exit to win the game. The player character, the scuba diver, is controlled using the arrow keys, and the menus can be navigated using the mouse.

Design/Plan Changes

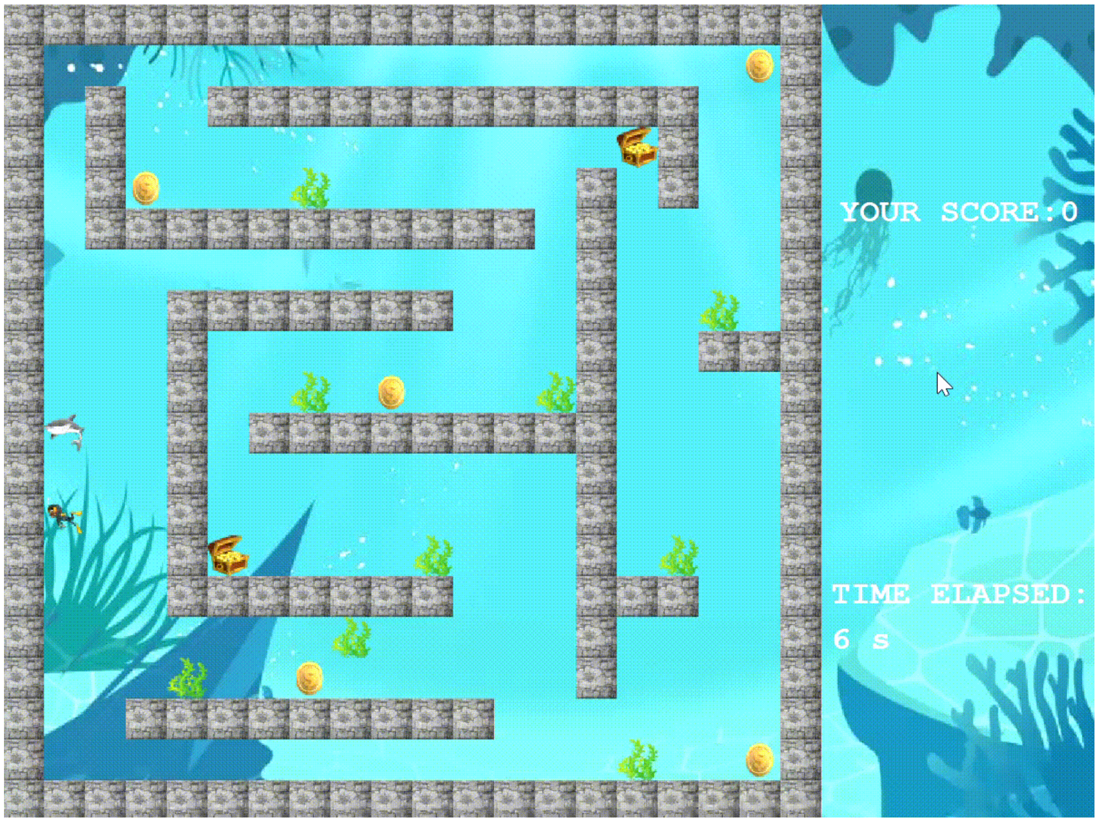
We generally remained faithful to our original design intentions overall. Namely, in the architecture of the code as it's based around a 2D array of objects used to drive the interaction and logic between entities. We also maintained a very similar hierarchy of entities in the final code compared to our original UML Diagram, with only a few caveats. We did not end up needing the MovingObject, StaticObject, or Obstacle subclasses of the Entity class. Upon implementing the design, these distinctions became unnecessary and the Entity hierarchy was abridged to reflect this. The name of some classes were also changed as design decisions were finalized. These included Door becoming Exit, as well as RegularReward and BonusReward becoming Coin and TreasureChest respectively. Where our final code differed considerably from our design, was on the right side of the tree involving things like the UI and game state. Because we weren't sure what library/toolkit we would be using, we outlined a more general layout for what we thought the GUI and IO elements would require. After settling on our toolkit and laying out the base logic for the game, we essentially overhauled this portion of the design to suit what the implementation called for. Functionally and aesthetically our vision for the game remained largely the same, with some minor tweaks when and where we felt they were needed.

Lessons Learned

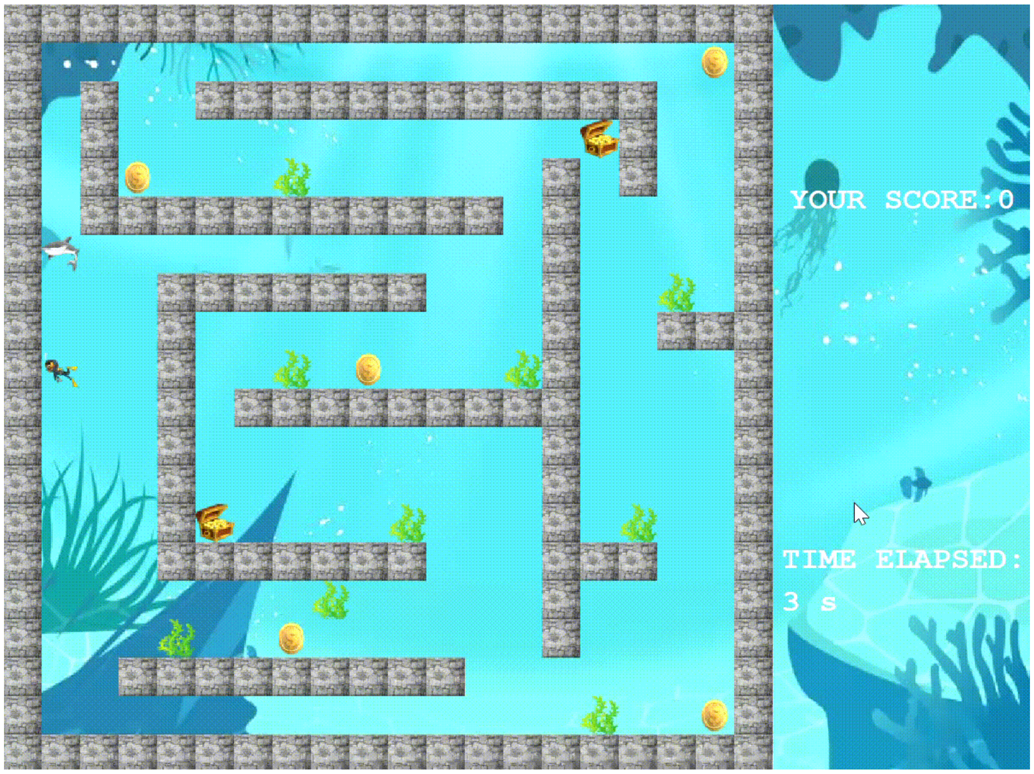
There were several key takeaways from this project that became apparent over the entire process of its creation. A big one was time management. All of us are obviously students with dense schedules during the semester, so learning to schedule adequate time to work on this project as a group and individually was a challenge. Another lesson learned was the importance of proper software modularity and design patterns implemented during the early stages of the project. We were able to implement a Factory design pattern for the Map objects which made laying out the initial map and testing much easier. It would also allow us to easily make more layouts for additional levels should the desire to expand the game ever arise. The final big takeaway was how crucial understanding and planning around requirements is. As our time was limited, we needed to agree on what elements to focus on and implement first, and which were of lesser importance. This relied a lot on us discussing our interpretations of the requirements and how best to fulfill them in a reasonable amount of time. As we weren't too strict on delegating tasks, we had to trust that we were all on the same page for the phase we were in and work on the parts of the software as needed. I think a better plan for assigning tasks and how each requirement was to be fulfilled would have alleviated much confusion over the development of the project.

Tutorial/Demo

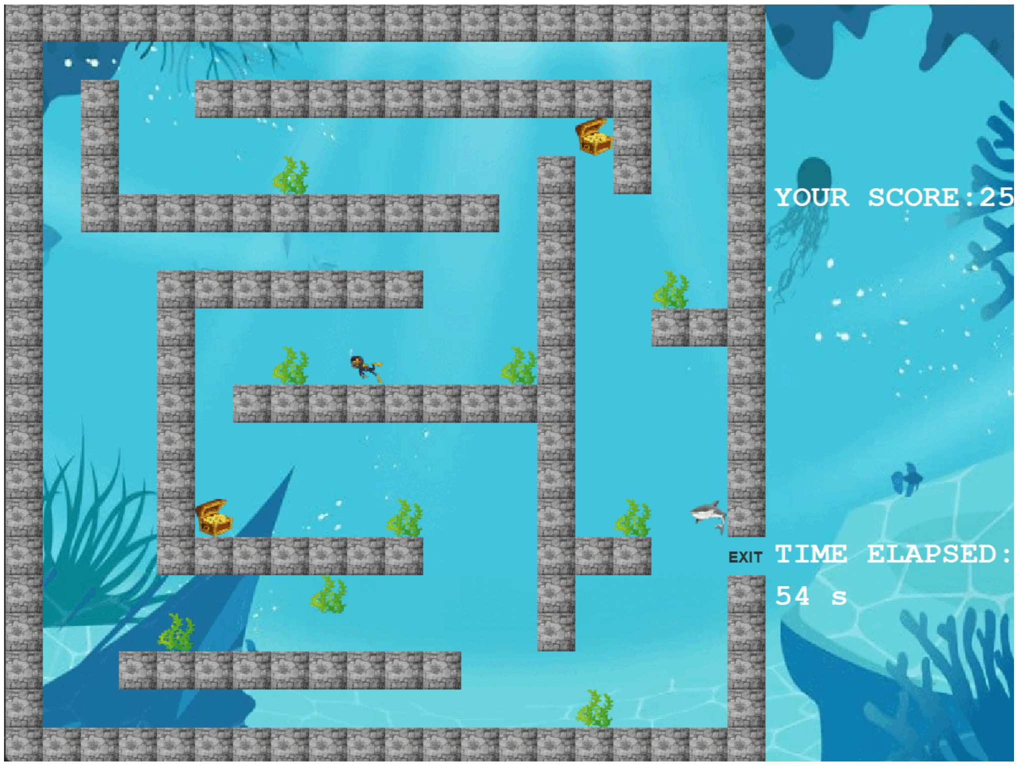
Control the Diver by holding the arrow keys in a direction. The shark uses a BFS pathfinding algorithm to chase the player around the map.



Collecting coins adds 5 to the player's score, running into seaweed causes it to decrease by 10.



Exit appears only when all coins are collected.



Demo Video: https://www.youtube.com/watch?v=rE62PFih8c4&t=2s&ab_channel=EricChan