How to play:

Use the **Left Mouse Button** and click the UI buttons to select a tower, and them place it with the **Left Mouse Button**. Use **Cancel** button to resume the game. Click in a tower with the **Left Mouse Button** outside of the "place mode" to open the Upgrade menu. Click to **Upgrade** if available, or **Cancel** to resume. Hold the **Right Mouse Button** and move the mouse, so the camera moves around the arena. In the leaderboard, add your name and click **save**, to add you score.

The red enemy tries to reach the center. The Orange enemy at start gives speed for those who are close to him. The Pink enemy is a boss with a lot of HP.

About:

I started analyzing what problem I had and what I could use to solve it. My first thought was that, for a Tower Defense, using the Unity DOTS and ECS where the way to go as this could provide a most performant result for the final product. But, as the aim is to deliver a maintainable code, and I don't have so much experience with it as with Game Objects, and of course it's a new API for Unity, which could let to a lot of problems, I decided to stick with Unity 2020.3, as recommended.

At start, I prioritized crating the base of the game and the systems that would make it work, so I could extend these and add new content as it would work by default. For example, I created the base of the Towers, and the UI to place them. At the end of the project, I just created new Towers, and they worked perfectly.

My approach to achieve a maintainable code was following SOLID principles, with interfaces and abstractions. I created a modular system for Enemies, Towers, and game logic, so they can be extended as needed, and without the need of refactor, as the code only knows interfaces, and not the implementation. As example, the UI, the Enemies and Towers are easy to extend and implement new versions for new content in the game.

To address enemy pathfinding, I used the Unity's default NavMesh system. The enemies use the NavMesh to find the path to the target. The placed Towers carve the NavMesh creating holes that block the current path, making them recalculate. To prevent the enemies from getting stuck or unable to find a way to the target, as it can get completely blocked by towers, I implemented a path-check, so after adding a Tower but not enabling it, I try to find a path to the destination. If it's not available, the tower is removed. If it is, the Tower is enabled. Additionally, a Tower can only be placed at some distance from enemies and another tower, to prevent bugs as making Enemies stuck inside a tower.

One significant performance concern was the potential slowdown when adding numerous towers and enemies at runtime. To mitigate it, I considered some approaches as implementing multi-threading or asynchronous code for the NavMesh and pathfinding. However, given the limitations, and time, I opted for introduce a tactical mode that pauses the game when the player intends to upgrade or build a new tower. This prevents real-time performance issues as gives the game time to process carving or recalculating pathfinding, without the player noticing. Also, this helps in the gameplay, by giving the player time to strategize.

And for complement, I minimized the need for instantiations and destructions in the game, that could heavily impact on performance, using a generic Pool system, for the big number of enemies and also the projectiles used by Towers.

In conclusion, I believe my Tower Defense game architecture achieved a good code maintainability, and to sustain this, I can use my experience to extend my own systems by adding new enemies, Towers, UI, game logic, without problems.