First, game controls:

- Camera displace: a,w,s,d.
- Camera rotation: mouse with right button down.
- Type 0 button to build blue towers; Type 1 button to select tower to build.
- Move mouse over any free board cell and left button click to build the tower. Right button anywhere to discard selection.

The design of the game is intended to have a very loose coupling and to be easy to scale up adding new features like different towers and also to receive major modifications (e.g. easily change the BFS path finding algorithm to a different implementation). To achieve this I made two major design decisions at the beginning of the work:

1. Centralized message flow. I tried to centralize the message flow in the GameManager component. Even if this project has limited message flow requirements the use of GameManager for orchestration seems a good decision (in most projects I had to be very careful with message flow because Unity projects tend to become chaotic and the application flow almost impossible to follow if the design allows uncontrolled message pass from one script directly to another). The use of a orchestration agent means that any pair of scripts willing to communicate will do it through the agent which will keep the overall project coupling at minimum. There are two major exception to this:
    a. Objects/scripts in the same prefab. Generally all prefabs elements are tightly coupled but being each one an indivisible entity working as black box the coupling is not a big issue.
    b. Some special cases, as the missile script invoking the target 'ReceiveDamage' from DamageDeal. In these cases direct reference would add extra complexity and there is no need of it (i.e. in the missile case the reference to the target is provided by the OnTriggerEnter and this is exploited to simplify the message flow).

2. Each script implements a particular and independent behaviour. This is particularly clear in the Enemy prefab which includes three scripts, 'Enemy' for the game rules behaviour (receive hits, react them, keep the target reference), 'Lifebar' to keep the lifebar updated and faced to the camera and 'FollowPath' to move the enemy through the scene, change waypoints and recalculate paths to the target. This means that each behaviour can be easily changed or completely replaced by a new script favoring scalability.

Towers hierarchy is designed to make all Tower children work in a similar way. This is the reason of having a parameter in the TowerType1 for the line renderer and for having a prefab for it line renderer is used as a projectile, in the same way than the missiles in the TowerType0 and the only change is how the attack is launched. Once instantiated, the ray itself will deal the damage to the tower. It's remarkable how, in all cases, is the projectile the element which deal damage to the enemy and not the tower which makes the application flow natural and easily understandable. This also makes quite easy to add different behaviours just creating new projectile prefabs to be launched from new towers prefabs or, depending on the projectile, from the current ones, with only minor modifications or none at all (i.e. a new kind of projectile acting as 'fragmentation missile' which makes damage to its target and nearer enemies would be very easy to add just building the new missile prefab inheriting from the current missile and then this missile could be launched from a new prefab tower which would be, essentially, a copy of TowerType0. Another change, a tower randomly firing one or other missile type on each attack would also be very easy to build just extending the current Type0 tower).

Most of scripts make use of intermediate variables rather than having long complex code lines. This is deliberated for making the script easier to read and for debugging purpose. Also, most of the script variables are declared as public even if it's not needed; I use to do such thing for debugging purposes grouping them with comments such as 'keep public', 'set private' or 'hide in inspector'. I use to make these indicated changes after debugging process but I decided to left those variables as public for your convenience.

BoardGenerator script builds the board. Spawner frequency is used for randomize the generation of enemy instantiators in the first row of board cells. In a similar way BlockerFrequency is the probability of a board cell to be blocking the path. The presence of blockers is convenient to test the path finding algorithm but keep in mind that no check is made to ensure that after instantiating the blockers all crystal are reachable by enemies so don't' set a too high value (in test values around 20% seems to work fairly well.

BuildManager script is remarkably easy to extend for new towers just adding new tower types in the AvailableTowers array. After that new buttons should be added to the GUI with the proper parameter.

GameManager also keep track of the game ending condition and MaxEnemies variable will be the total number of enemies to be instantiated on the board.

There is a second hierarchy in the board elements (blockers, instantiators and crystals). At first I thought of them as different objects but when I was working on the BoardGenerator I found that with such approach I was required to create three different instantiate methods, one for each kind of element. At that point I decided to create the current hierarchy which allowed me to implement the InstantiateBoardElement which receive the board element to instantiate as parameter. This remarkably reduced the work in BoardGenerator script.

The BFS search is performed by all enemies only in certain cases: when a new enemy is instantiated its perform the search; when a tower is set or a crystal is destroyed all enemies made a path finding to guarantee that all of them still follow valid paths to an existing crystal.

The project is deliberately built to allow quick and easy creation of scenes. You just need to drag and drop into the scene window the GameManager prefab and CameraHolder prefab, set your parameters and everything will be ready to begin playing!!

The project itself didn't was particularly difficult. I had only two remarkable issues, first failing to include the board elements hierarchy in the initial design rather than adding it at the end of the project which was more difficult because involved making some changes in the hierarchy elements. Second, the camera control scripts because I'm used to take the controllers provided in Unity standard assets but for this project I thought it was better trying to make one by myself and involved the biggest research effort even as simple as they are in the end.

I would like to have had more time for testing the scripts and add some more elements (I thought in fragtowers, towers which deal damage to its target an the near enemies and bufftowers with have a minimal damage output but increase the near towers damage attribute and other elements) but unfortunately this had been a very busy week for me and I'd barely could put half of my time on this project

Developed with Unity 5.0.1f1