CastleWar

CastleWar is an online player-vs-player 2D RTS game. Each player starts with a simple castle, overlooking the opponent's castle across a field. The objective is to destroy the opponent's castle, using to this purpose several weapons which the player must build with various resources. At the same time, the opponent shall try to destroy the player's castle; the player can also build fortifications to his own castle, in order to prevent this (or at least delay it for long enough). The destruction of the castles is ruled by a physics engine, as are the different weapons: a catapult shot may have more effect if fired at the base of a tower than at the top of it.

If a certain percentage of a castle is destroyed, the player to whom the castle belongs is defeated. By building new fortifications and weapons, one is contributing directly to the resilience of the castle, since it counts towards its total percentage.

Looking into a few more details:

In addition to having a starting castle, each player also has an area around the castle on which he may build whatever structure he wants (save a few exceptions). We shall refer to this zone henceforth as the "courtyard". The starting castle is composed of the castle (referring to a single building in this case), two small buildings: one for resource storage (the "storage") and another serving as the workers barracks; surrounding walls with a tower on each end and a "gate wall" in the direction of the opponent's castle. Since the game is displayed in 2D, a button shall allow switching between interaction with the walls and towers, and interaction with the courtyard, where the castle and the other buildings are located.

A player may extend his walls (and towers) to the limit of the courtyard, which allows for more protection of the buildings in the courtyard. When the player extends the walls, the gate wall is automatically moved to the nearest position relative to the opponent's castle. The gate wall is the opponent's knights only point of entrance, thus it should always have some form of defense, such as archers atop the wall (the player may choose to neglect its protection and move the archers, however).

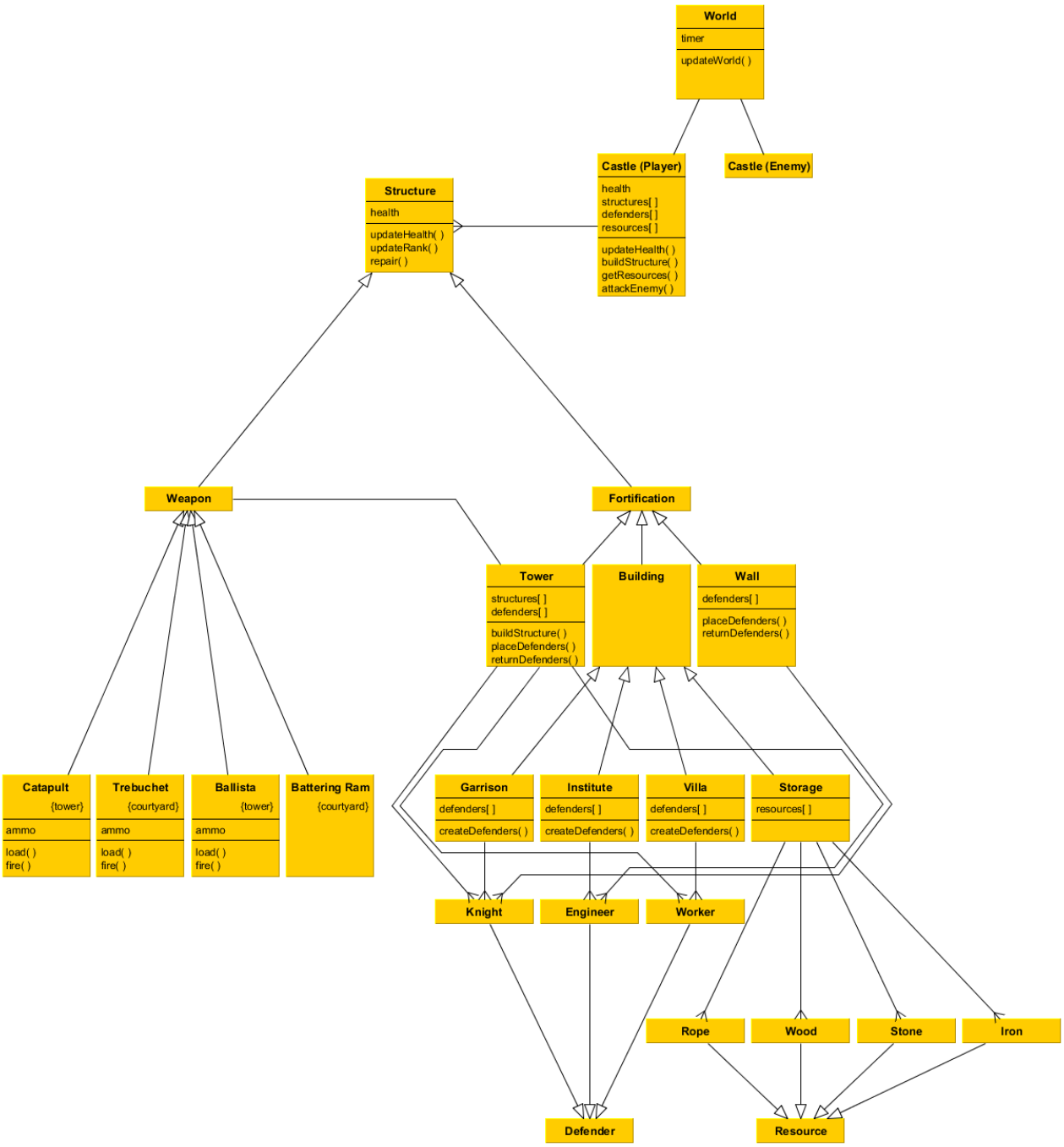
Some weapons may be built on the top of the towers, others can only be built on the courtyard.

A tower may be built with whatever height (up to a limit) the player chooses, as long as he has enough resources and workers (and time), and weapons built upon such towers may have added benefits; each wall can only be as tall as the shortest of the two towers it is built between. Obviously, the taller the wall, the harder it is for the opponent to hit the buildings in the courtyard. Some weapons, however, may be able to bypass these limitations…

Each structure in the game can be ranked up, improving its resistance and getting a few other stats buffed up (depending on the structure). However, always keep in mind that any construction may quickly meet its demise with a good (sometimes lucky) shot from a catapult, so one must consider the options carefully.

This concludes the introduction; now we shall look into some technical aspects of the implementation.

This UML diagram shows the classes that represent the objects that shall be present in this early version of the game, along with their most important attributes and methods:



It should be noted that these classes describe the logic of the game, not being responsible for the interaction of the player with it (which may depend on external libraries and/or require additional classes).

The “World” class shall be responsible for the coordination of the game between the two players, therefore it contains two and only two instances of the class “Castle” (which governs the objects inside the castle, as well as the castle itself), one for each player; every time a player’s castle suffers changes, either made by the player (create a building, fire a weapon) or by the opponent, the “World” class shall pack the updated information about the player’s castle state and send it to the other player smartphone (through the network); the class is also responsible for receiving any updates from the other player smartphone and treat them accordingly. The timer is used to synchronize the received information, comparing the timer of the player who sent the data pack with the current timer and changing the data accordingly before updating the game itself; the timers are synchronized at the beginning of the game.

Both “Castle” classes are identical (only one has been detailed, to avoid redundancy), and they contain references to all objects of its corresponding player’s castle. A game loop shall control the update of both castles’ objects, i.e. the progress of the construction of a tower, and any action made by either player is treated asynchronously. Once a player changes the state (through the process described in the previous paragraph), the “Castle” shall collect the updated information from a special variable and change its own data accordingly.