1. How can a player interact with the game? What are the possible actions?

The player may begin the game. Once the game begins, players can place their workers and then the rounds will start. Each player will be able to move a worker to an unoccupied, adjacent location and then build a tower/dome at an unoccupied, adjacent location during their turn. After each turn, the player checks if they have won the game. If so, the game is over, else, the game will switch it to the other player's turn. The players' defined actions exemplify the design principle of a low representational gap, since the actions are exactly what the instructions allow the player to do. Therefore, this accomplishes the design goal of understandability and change.

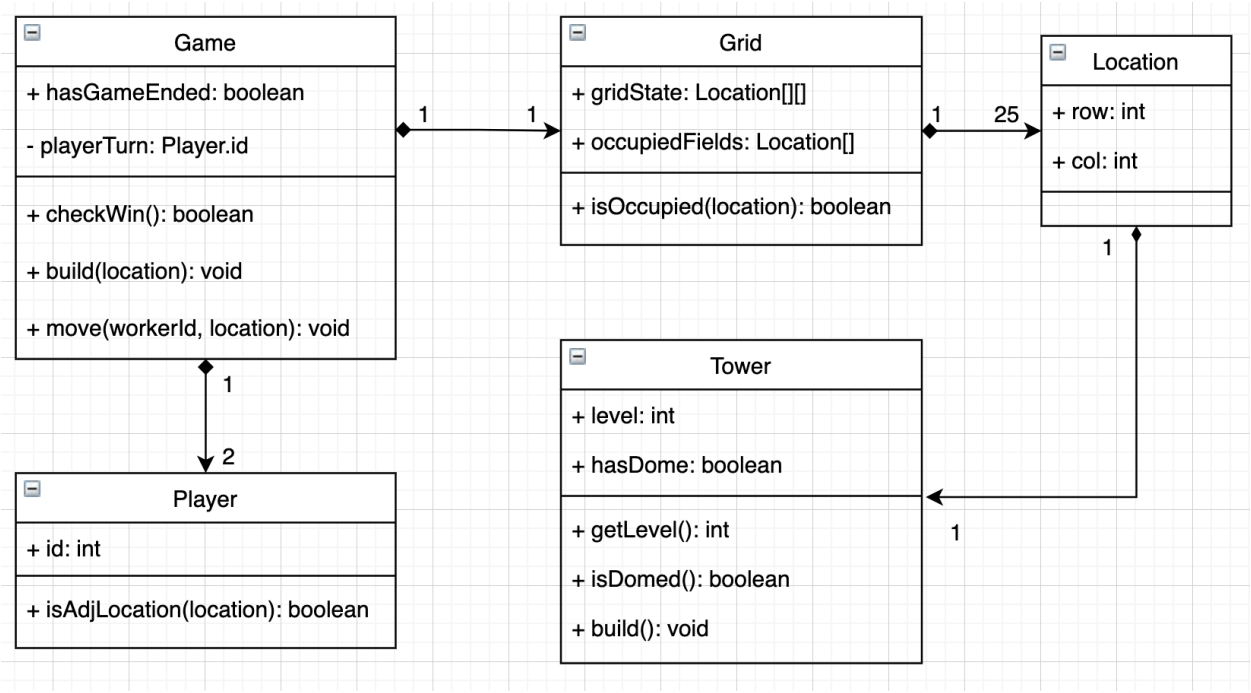
2. What state does the game need to store? Where is it stored? Include the necessary parts of an object model to support your answer.

The game has attributes hasGameEnded and playerTurn to track the game status and current player. It also stores the grid, represented in a Grid class, with attributes gridState and occupiedFields. The grid itself is represented by a 2d matrix of locations objects with row, col coordinates. Each location can have 1 tower object, which contains information about its level and if it has a dome. For efficiency, the Grid class also has attribute occupiedFields which represents locations that workers cannot move to. This includes locations with domes and locations with other workers.

The game, grid, location, and tower classes are separated to use the design principle of high cohesion and to accomplish design goals of understandability, reuse, and change. Generally, this separation is more understandable, and if any class was changed to add features, those changes would not have drastic effects on other classes.

Additionally, I chose to design occupiedFields as one attribute instead of splitting into separate domeFields and workerFields, since they have similar functionality where a worker cannot move to that field. This design choice was made for the design goal of understandability and reuse, since there will be less attributes cluttering the Grid class and the isOccupied function can be used in many different contexts for the game.

Object Model parts (Game, Grid, and Location objects)



3. How does the game determine what is a valid build (either a normal block or a dome) and how does the game perform the build? Include the necessary parts of an object-level interaction diagram (using planned method names and calls) to support your answer.

To determine a valid build, the game uses the player method isAdjLocation(location) to determine if location is an adjacent location to either worker of the player. After, the player calls grid methods isOccupied(location) and getLevel(tower) with the potential build location, and if the location is not occupied and has a level less than 3, then it is a valid build location. If any of these conditions are not met, there will be a failure message returned to the Player/GUI.

To perform the build, the game uses the tower method build(), which increases the level of the tower at that location by 1. The exception is when the level is already 3; in that case, the tower's hasDome attribute is turned to true.

I decided to incorporate the dome into the tower class, since the dome is part of the tower. Therefore, the design goal of understandability is accomplished because keeping them together is more intuitive. In addition, I decided to put the build method in the Tower class, since this incorporates the design principle of low coupling, as building a block/dome does not depend on attributes in other classes such as Grid or Location. This accomplishes the design goals of reuse and change with a more modular approach.

Object-level Interaction Diagram