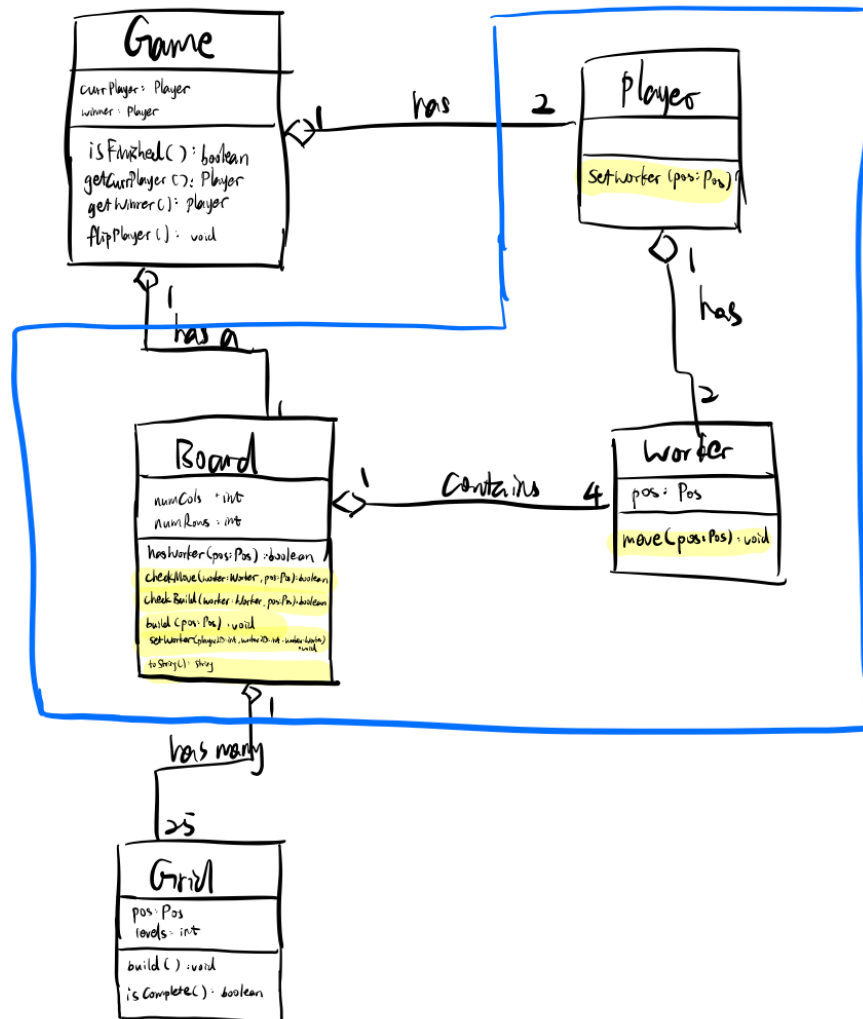


Task 3: Justification

1. Player's interaction with the game and possible actions

The player can interact with the game in the following ways:

- (1) Set the initial position of his/her workers at the beginning of the game
- (2) Move his/her workers
- (3) Build using his/her workers
- (4) check the current board situation



The above actions are implemented and explained in Object Model as below:

- (1) When setting the initial position for the worker:
 - the Player class call method `setWorker(Pos pos)`

to initialize and set the position of the worker

- the Board class call `setWorker(int playerId, int workerId, Worker worker)`
to add the worker that is initialized by the player to the board

(2) When a player wants to move his/her worker:

- the Board class call `checkMove(Worker worker, Pos pos)`
to check if the move to do is a valid move;
- If the move is valid, the corresponding Worker class call `move(Pos pos)`
to update the position of the worker

(3) When a player wants to build using one of his/her workers:

- the Board class call `checkBuild(Worker worker, Pos pos)`
to check if that build is a valid build
- the Board class call `build(Pos pos)`,
which further invoke `build()` at the corresponding Grid class
to actually perform the build and update the grid tower levels

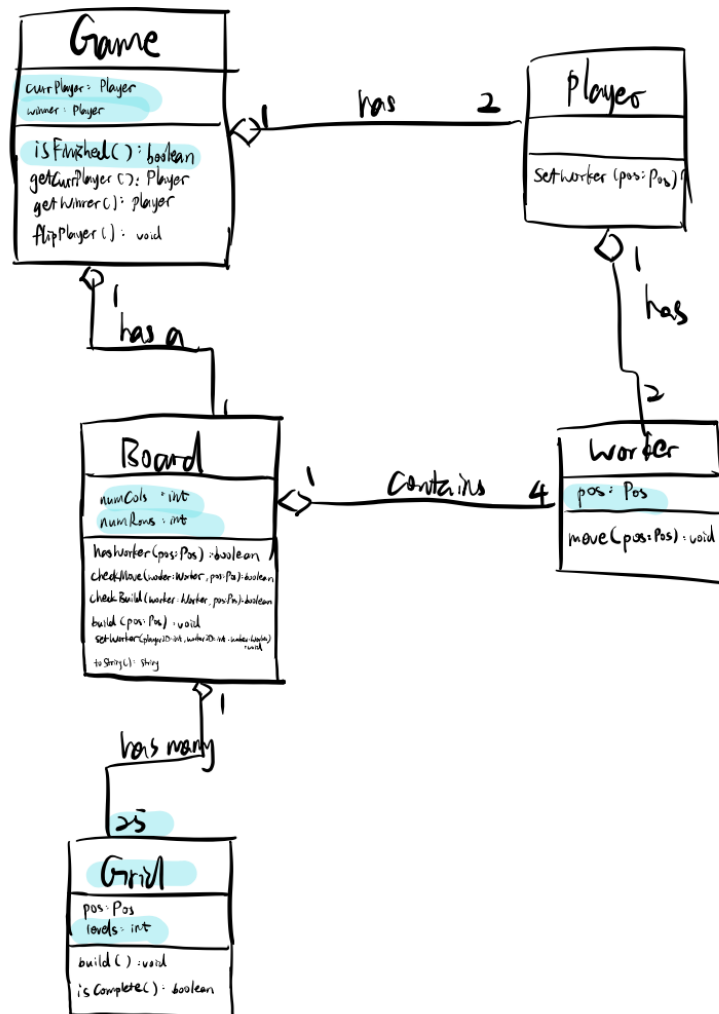
(4) When a player wants to check the current board situation:

The Board class has a override `toString()` method that prints the current board (with the grid's levels and workers' positions for both players in a human-readable way)

2. What state does the game need to store and where are they stored?

The worker's positions, the tower levels (we can think of the Dome of the tower as level 4 of the tower) of each grid on the board, and who is the next player needs to be stored.

Also, if the game is still going or has ended, and who is the winner if the game has ended also need to be stored.



As shown in the Object Model,

- Worker class store the current position **pos: Pos** of the worker.
- The tower levels of each grid are stored as **levels: int** in the Grid class.
- Each grid on the board is stored as a 2-D Grid array in the Board class.
- The next player is stored as **currPlayer: Player** in the Game class.

- The game status (if is finished) can be accessed by **isFinished(): boolean** function in the Game class.
- The winner of the game (if the game has finished) is stored as **winner: Player** in the Game class. This is set when the **isFinished()** function is called.

3. How does the game determine a valid build? How does the game perform the build?

When a player wants to perform a build, the game has the following operations:

- first, the game checks if current next action should be "Build". This is verified from accessing `game.nextAction()`. Also, in a god Demeter mode, the game will also call `game.usePower()`, inside which `god.canDoAction()` will be checked to see if the current situation allows the player to use the Demeter power and perform another build
- Then, the frontend gets the worker id that the player wants to perform the build, and the position of the build from the player via IO;
- next, the Board class calls `checkBuild(Worker worker, Pos pos)` to check if the build is valid
- inside the `checkBuild()` function,
 - the Board calls `isInBound(Pos pos)` to check if the position is inside the board
 - the Worker calls `getPosition()` to get the current position of the worker
 - the Board calls `isAdjacent()` to check if the build position is adjacent to the worker position
 - the Board calls `hasWorker()` to check if the position is already occupied by another worker
 - inside `hasWorker()`, all 4 workers stored in the board calls its `getPosition()` function to check if they are occupying the target position
 - the Board calls `getGrid(Pos pos)` to get the corresponding grid of the target position
 - the grid calls `isComplete()` to check if the tower of the grid can be further built (if contains a doom)

If all of the above checks succeed, the `checkboard()` returns true

Otherwise if any of the above checks fail, the function will return null, indicating the build() is failed

- If the check passed, then, the Board calls `build()` function to actually build the block.
If the build() is coming from another build result in using Demeter power, the game will set the next action to `Game.BUILD` from `Game.MOVE` and switch the nextplayer from the other player to the current player (since after the previous normal Build, the next action SHOULD HAVE BEEN the `Game.MOVE` performed by the other player)

Then, Inside the `build()` function,

- the Board calls `getGrid(Pos pos)` again to get the grid of the target position
- The grid calls `build()` to finish the building operation

The corresponding Interaction diagram and Object Model diagram is shown below:

[UPDATE: for the milestone for hw5, the diagrams are not updated yet in case that the design needs to be changed for the hw5 final]

