Justification

Q1:

At the beginning, each player can interact with the game by providing the starting positions for two workers to initiate it in the *Game* class using the *initiatePlayer(x1, y1, x2, y2)* method. And the game will store a list of Players, a *currPlayer* field to indicate current operating player, and the *winner* field also.

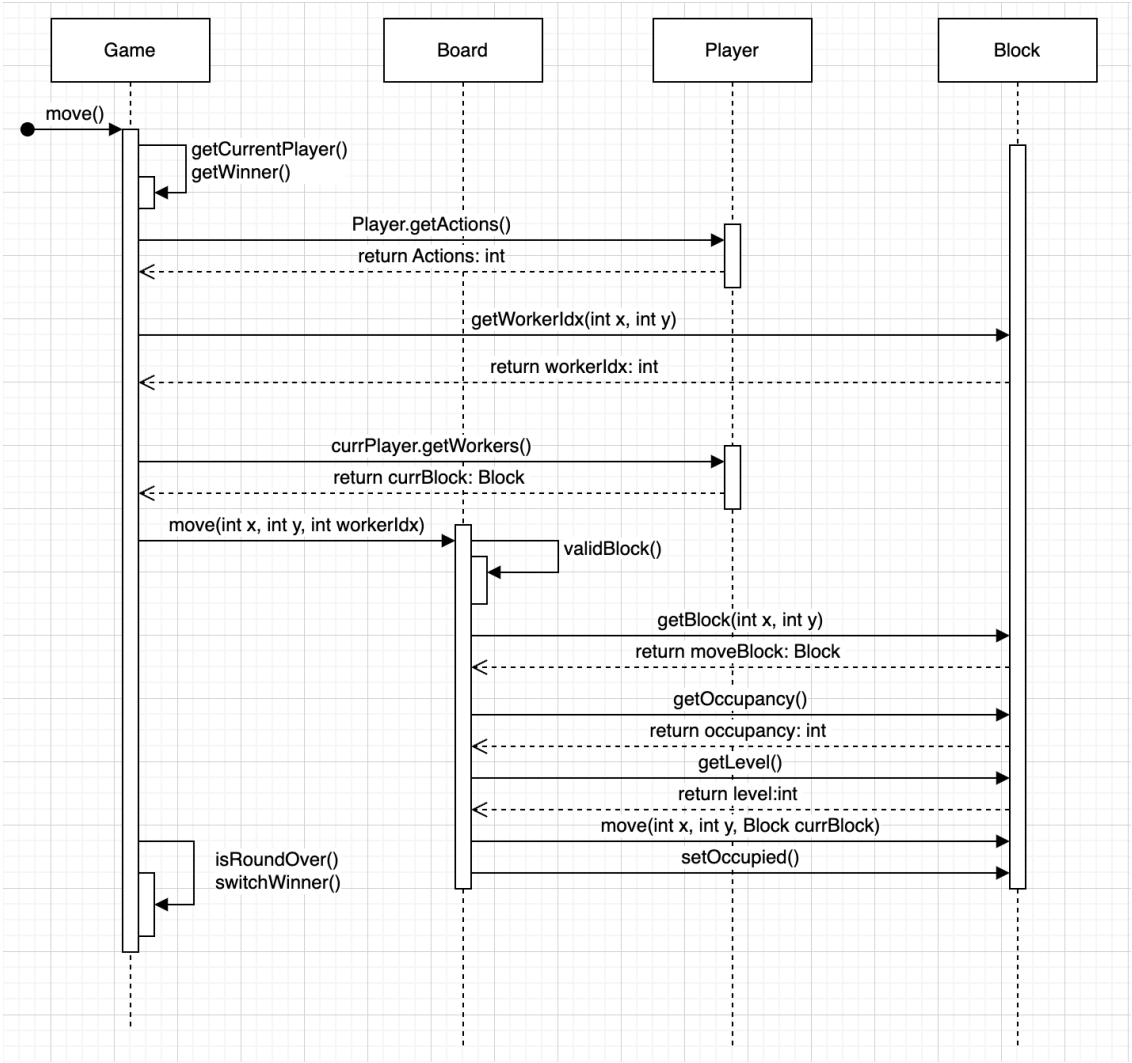
As the game progresses, players can continue to interact with the game by performing a variety of actions. Possible actions include moving a worker and constructing a tower or a dome. The player can execute the move and build block or dome functions within the *Game* class and the *Board* class by supplying the coordinate values, as well as the worker index of the current player.

The moving execution requires the player to call the *move(int x, int y, int worker_idx)* method in the Game class, and the Game class will then call the *move(int x, int y, Block currBlock)* in the Board class automatically.

To execute a successful move, the board checks the validity of the movement by verifying the destination coordinates against the board's boundaries using *validBound(int x, int y)* and ensuring they are adjacent to the worker's current location using *validSurround(int x, int y, block)*. The board also checks the current occupancy and level of the destination block using *getOccupancy()* and *getLevel()* from the Block class, respectively. If the move is valid, the Game class updates the actions and associated block position of the current player using *setActions()* and *setOccupancy()*.

The Game class will update the actions and associated block position of the current player using *setActions()* and *setOccupancy()*, if the worker moves successfully.

Building a tower or a dome follows a similar process, but it is more crucial to ensure the validity of the building in terms of level.

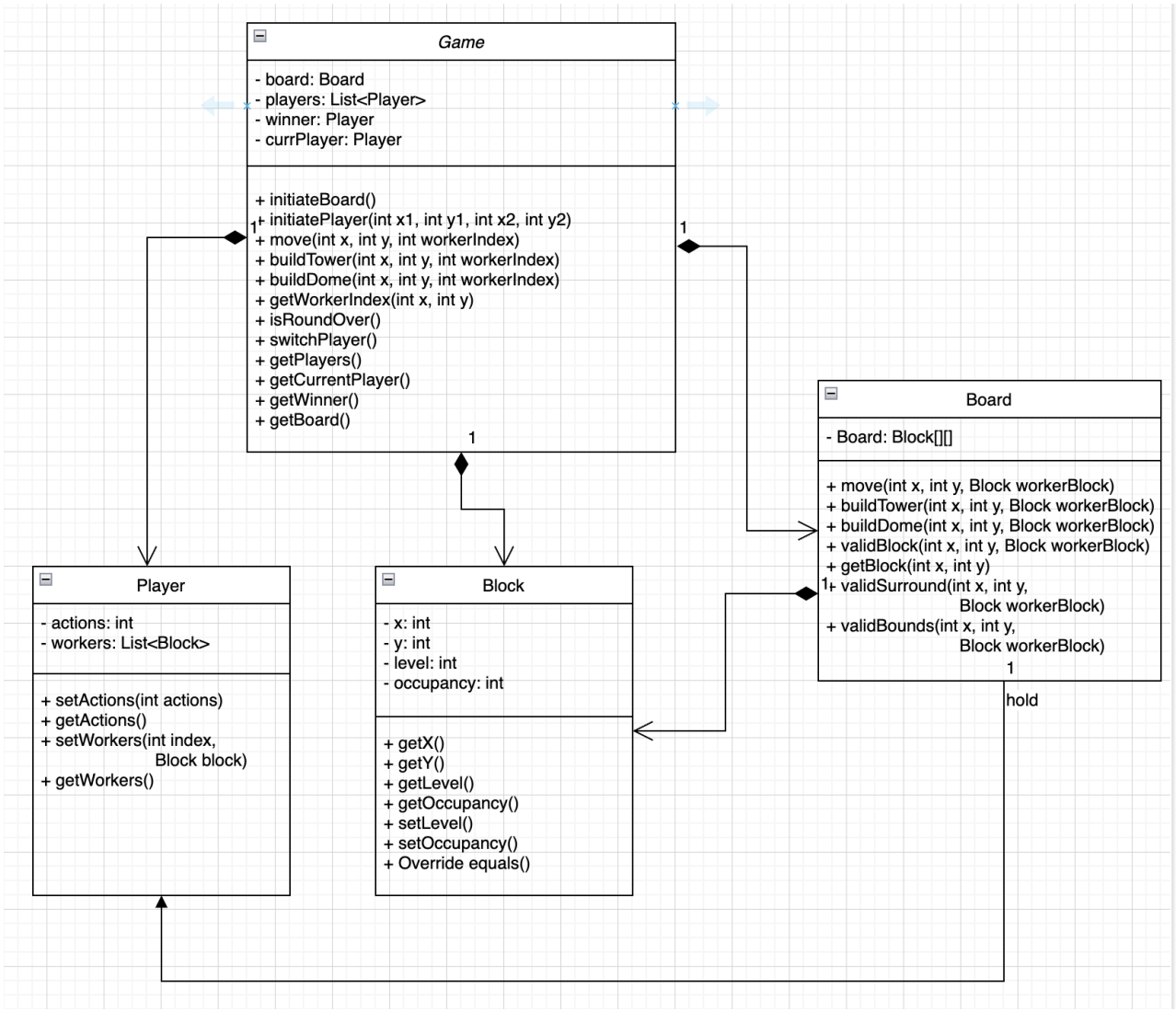


Q2:

To keep track of the current player and their available actions, the Game class needs to store a list of two players, the current operating player field, the winner field, and the reference of the whole board. By maintaining a list of two players, the Game class is able to alternate players after each round, and the Board class is responsible for implementing action functionalities. The Game class acts as a manager to ensure coordination between the players and the board.

To keep track of the action state, the Player class stores an integer representing the player's remaining actions, starting at 0. If the player wishes to move then build, the action field will increase to 1 and then 2, ensuring that workers move and build in the correct order. In addition, the Player class stores a list of workers and their occupied blocks, enabling easy access to each worker's current position. Since the block reference is stored, there is no need to update twice as updating the occupancy and the level of a block in the board.

Each block on the board stores its coordinate positions, level, and occupancy. The Board class maintains the entire board's information, enabling it to validate moves and builds and update the appropriate blocks as necessary.



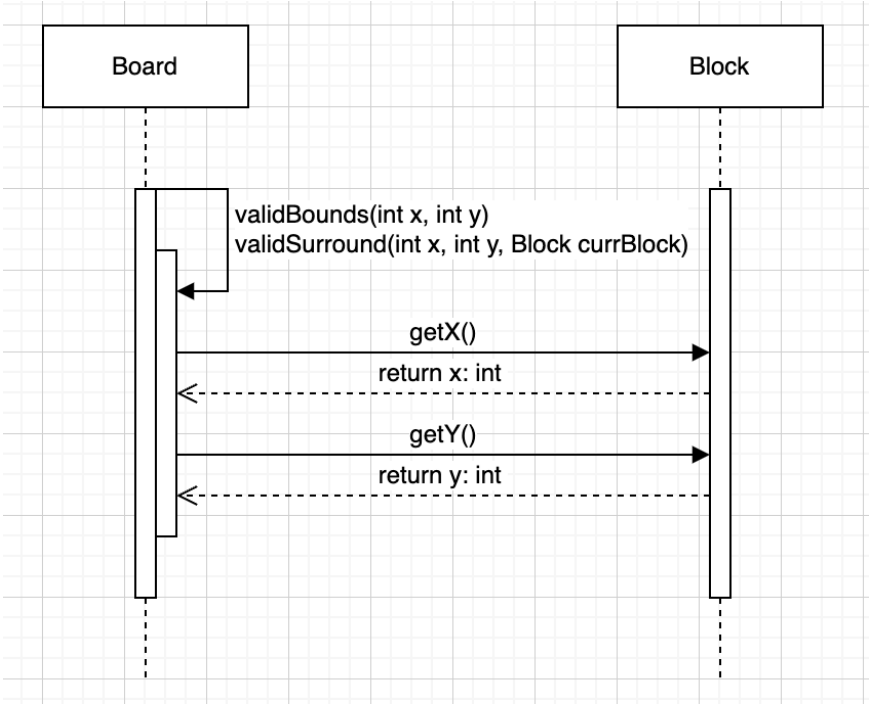
Q3:

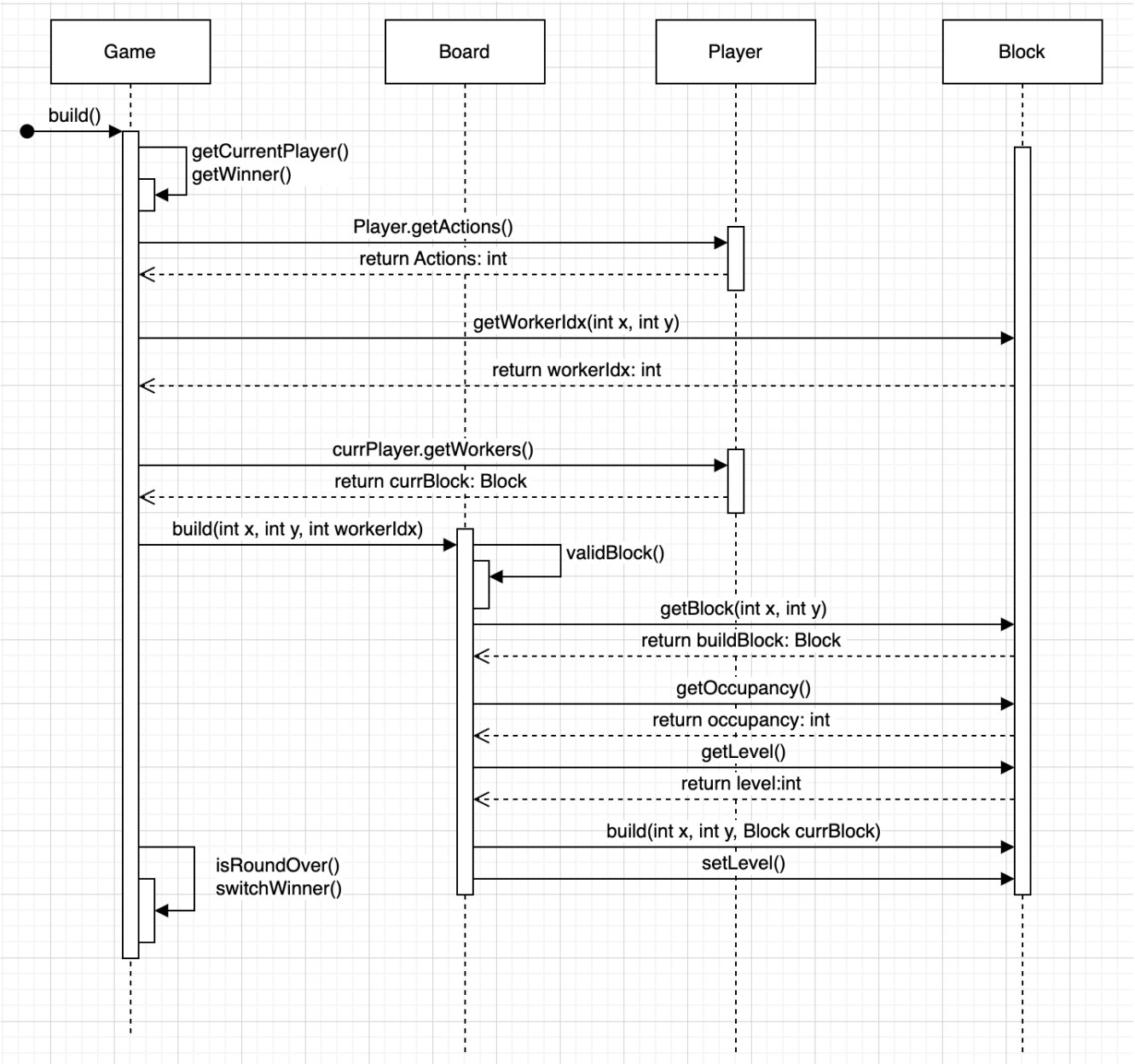
To follow the principle of better maintaining the code and ensuring high cohesion, the game uses separate methods to check various requirements for validating a build. The building execution requires the player to call the *build(int x, int y, int worker_idx)* method in the Game class, and the Game class will then call the *build(int x, int y, Block currBlock)* in the Board class automatically. And then the game determines whether this is a valid build execution by checking the following.

- The build position must be within the board boundaries.
- The build position must be adjacent to the worker performing the build.

- The build position must be unoccupied.
- The build position must not already have a dome.
- The build position must be at an appropriate level (it is not a complete tower for building a tower, and it has to be a complete tower for building a dome).

The worker's block can be used to check whether the given position to build is inside the board, adjacent to the selected worker and not occupied using validBlock(int x, int y, Block workerBlock) which includes *validBound(int x, int y)* and the *validSurround(int x, int y, Block workerBlock)*. The occupancy would be checked by accessing the specific block's *getOccupancy()* method.
The above method also verifies if the build position is constructible, which means it does not already have a dome, by accessing the specific block's *getLevel()* method. If the level is not equal to three or equal to four, it would not be allowed to build a dome. If the level is equal to three or four, it would not be allowed to build a tower.
If the build position is valid and does not have a dome, add one level to the build position using the method *setLevel()*. And the current player has completed all actions in this round. The Game class will update the actions of the current player using *setActions().*
The following interaction diagram is to demonstrate how validBlock(int x, int y, Block workerBlock) work internally.




Game
Board
Player
Block
build()
getCurrentPlayer()
getWinner()
Player.getActions()
return Actions: int
getWorkerIdx(int x, int y)
return workerIdx: int
currPlayer.getWorkers()
return currBlock: Block
build(int x, int y, int workerIdx)
validBlock()
getBlock(int x, int y)
return buildBlock: Block
getOccupancy()
return occupancy: int
getLevel()
return level:int
build(int x, int y, Block currBlock)
setLevel()
isRoundOver()
switchWinner()