Justification – Homework 5

1. **How can a player interact with the game? What are the possible actions? Please include necessary parts of the Object Model to explain. (Homework 3 version)**

   The player interacts with the game by calling the functions in Island board class which is the main body of the game. There are several possible actions. *showStatus()* could print the current status of the game board including the position of workers and tower configuration in each grid. *initialPlayersandWorkers()* could initialize the game and set the original players and workers. When the new game starts, there are four phases for each player. Phase 1 would call function *moveWorker()* and the player needs to move one of the worker to one of its neighboring grids. Phase 2 would call function *buildToken()* and the player needs to build one token (block or dome) in one of its neighboring grids. Phase 3 would call function *checkscore()* which would check whether the victory conditions are met for the player. If the player does not meet the victory requirements then the player would call the function *changecurrentplayer()* and enter Phase 4 which is the last phase. After that, it will switch to the other player's turn and perform the same phase. The object model of interaction between player and game body would be presented as follow:

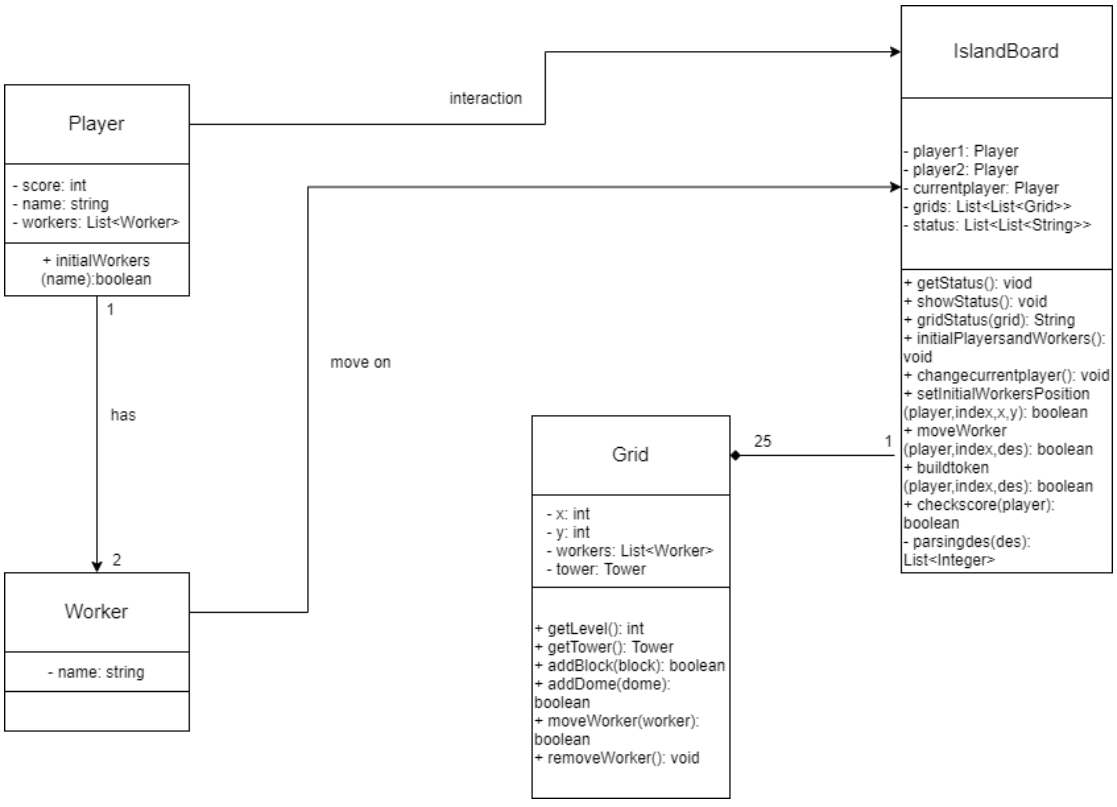


Fig 1 Object model of interaction between players and game

2. **What state does the game need to store? And where should it be stored? Please include necessary parts of the Object Model to explain. (Homework 3 version)**

There are several states need to be store. For player, the *score of each player* needs to be stored to determine whether there is a winner. For token (block, dome, tower and worker), the *position (coordinates) in the game board of each type of token* needs to be stored. For grid, *the state of worker and tower in each grid* should be stored. For example, whether the grid is occupied by a worker and the level number of tower in each grid. And for the island board which is the main body of the game, *the state of each grid in the game board* should be all stored in this class. For example, the initial position of each worker before the game gets started should be stored. And the status of each grid at the end of each player's turn needs to be recorded as the game progresses.

For player, the state of *score of each player* would be stored in the integer variable named *score* in each player class. For token, the state would be stored in the private integer variable named *x coordinate and y coordinate* in each token class. As for each grid, the state of *worker and tower in each grid* are stored in two separate lists in grid class. For island board, there is a nested list named *grids* which is in two dimension (5×5) and stored *the state of each grid in the game board.*

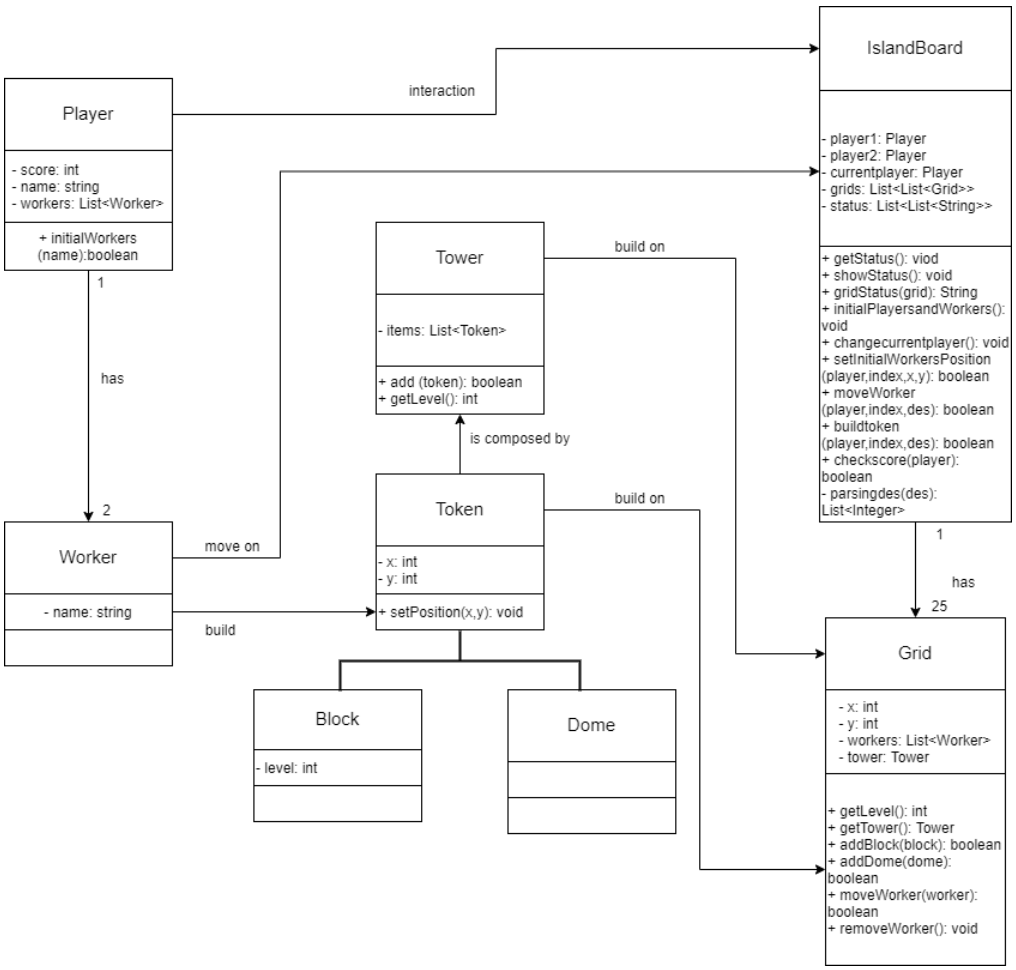


Fig 2 Object model of game state storage

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? Please include necessary parts of an object-level Interaction Diagram and the Object Model to explain. (Homework 5 version)**

   As we know, the Demeter could build twice. We assume the operations as build phase 1 and build phase 2.

   In the build phase 1, when the player needs to build a block there are three requirements it should meet:

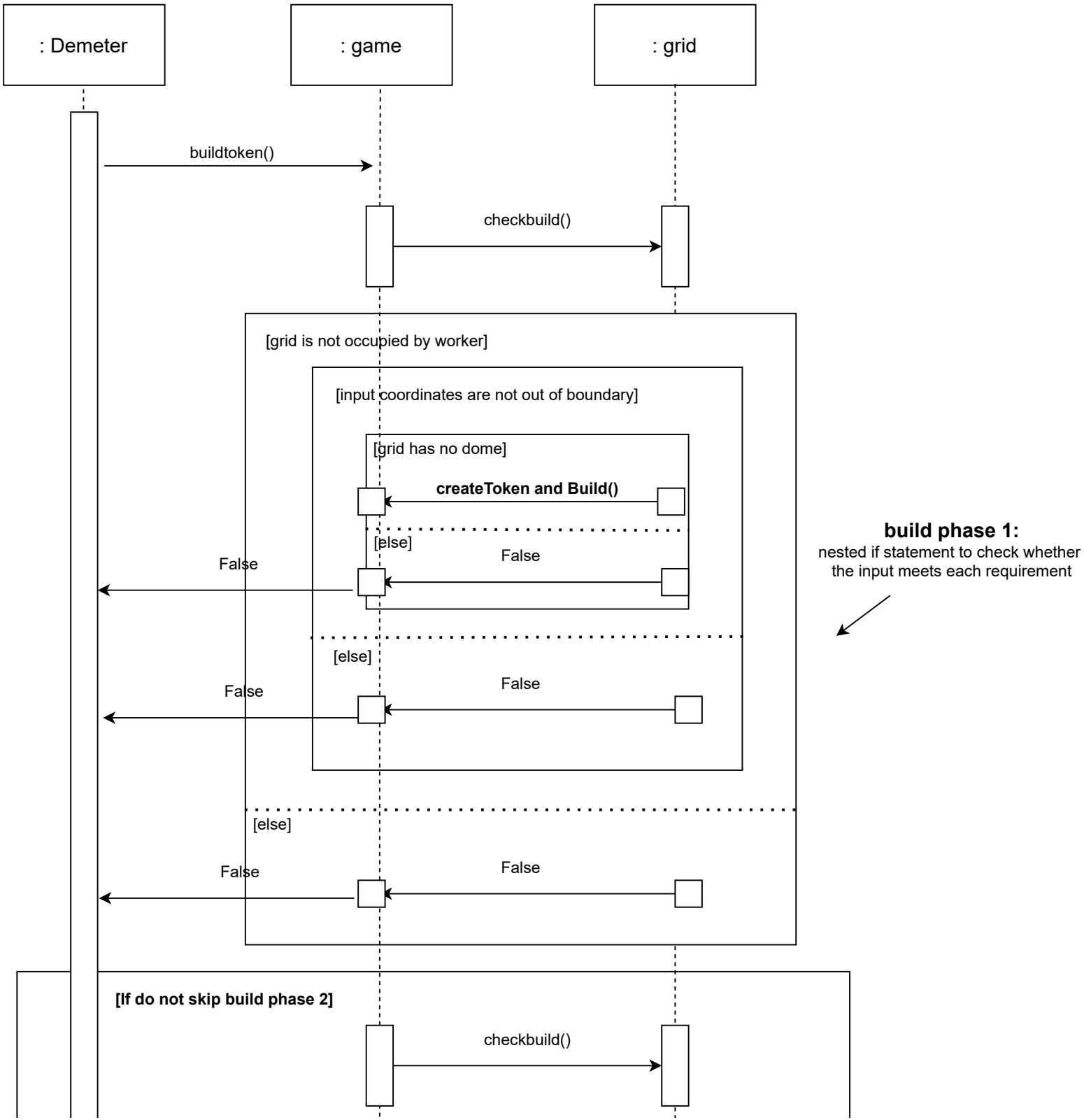
   1. The destination grid is not occupied by another worker.
   2. The coordinates of the destination grid are not out of the boundary of game board.
   3. The destination grid does not have a dome.

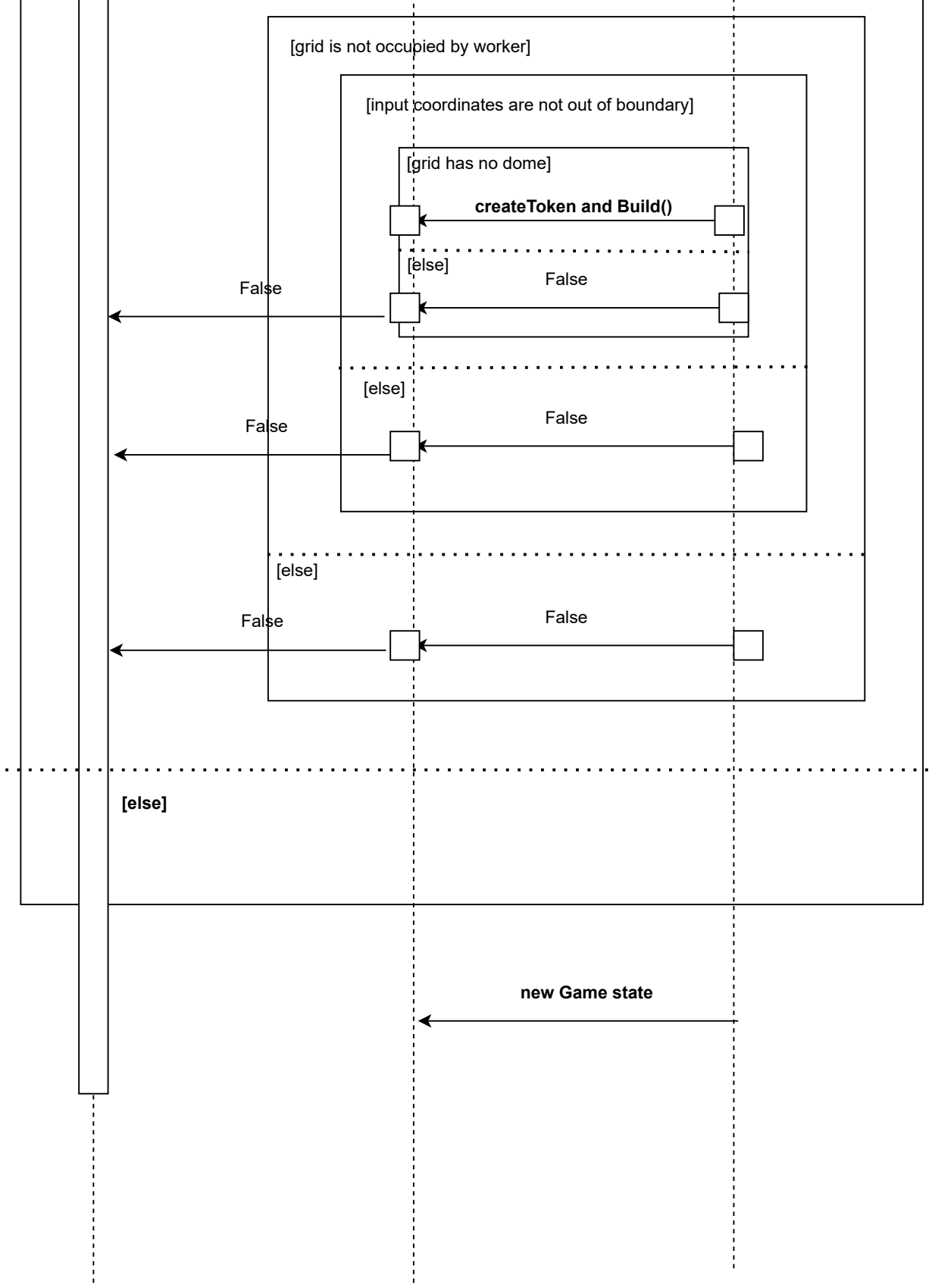
   In the build phase 2, if the player does not want to build any more, he/she could skip this phase. Otherwise, there are five requirements it should meet. In addition to the above three requirements, it is also necessary to meet that the grid build in phase 2 should not be the same with the grid build in phase 1. In other words, Demeter can't build in the same place twice.

   If all the conditions above are met, then the game would determine it is a valid build and perform action.

   When the game to performs the build action, in the phase 1, it acts as the normal worker build. In the phase 2, if the player would not skip this phase, it would mark the grid the player built in phase 1 as false. Then the player could not build on this grid in this phase.

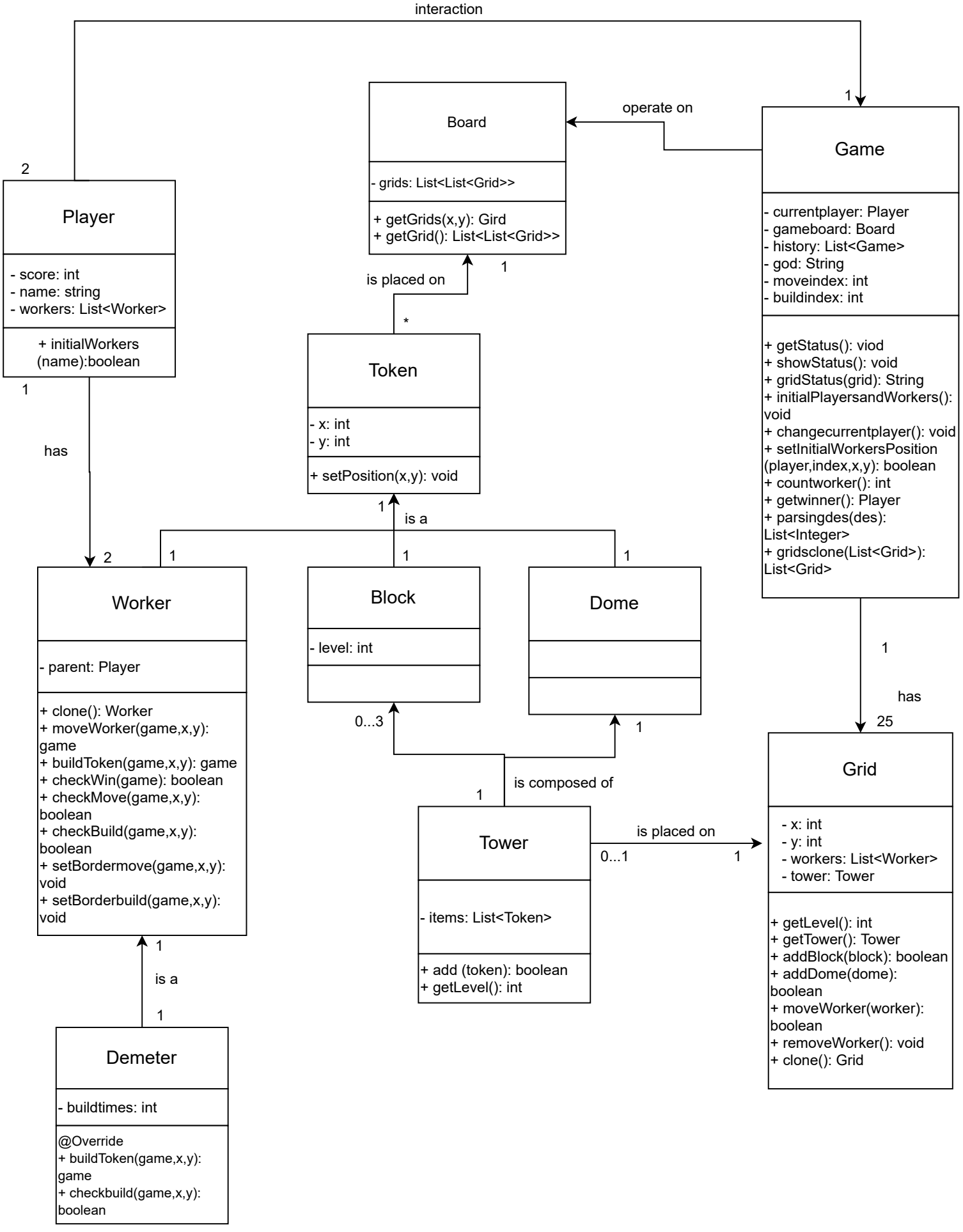
   The object-level Interaction Diagram would be shown as follow to illustrate how the game performs the build action.


: Demeter
: game
: grid
buildtoken()
checkbuild()
[grid is not occupied by worker]
[input coordinates are not out of boundary]
[grid has no dome]
createToken and Build()
[else]
False
False
[else]
False
False
[else]
False
False
[If do not skip build phase 2]
checkbuild()
build phase 1:
nested if statement to check whether
the input meets each requirement



build phase 2:
nested if statement to check whether the input meets each requirement

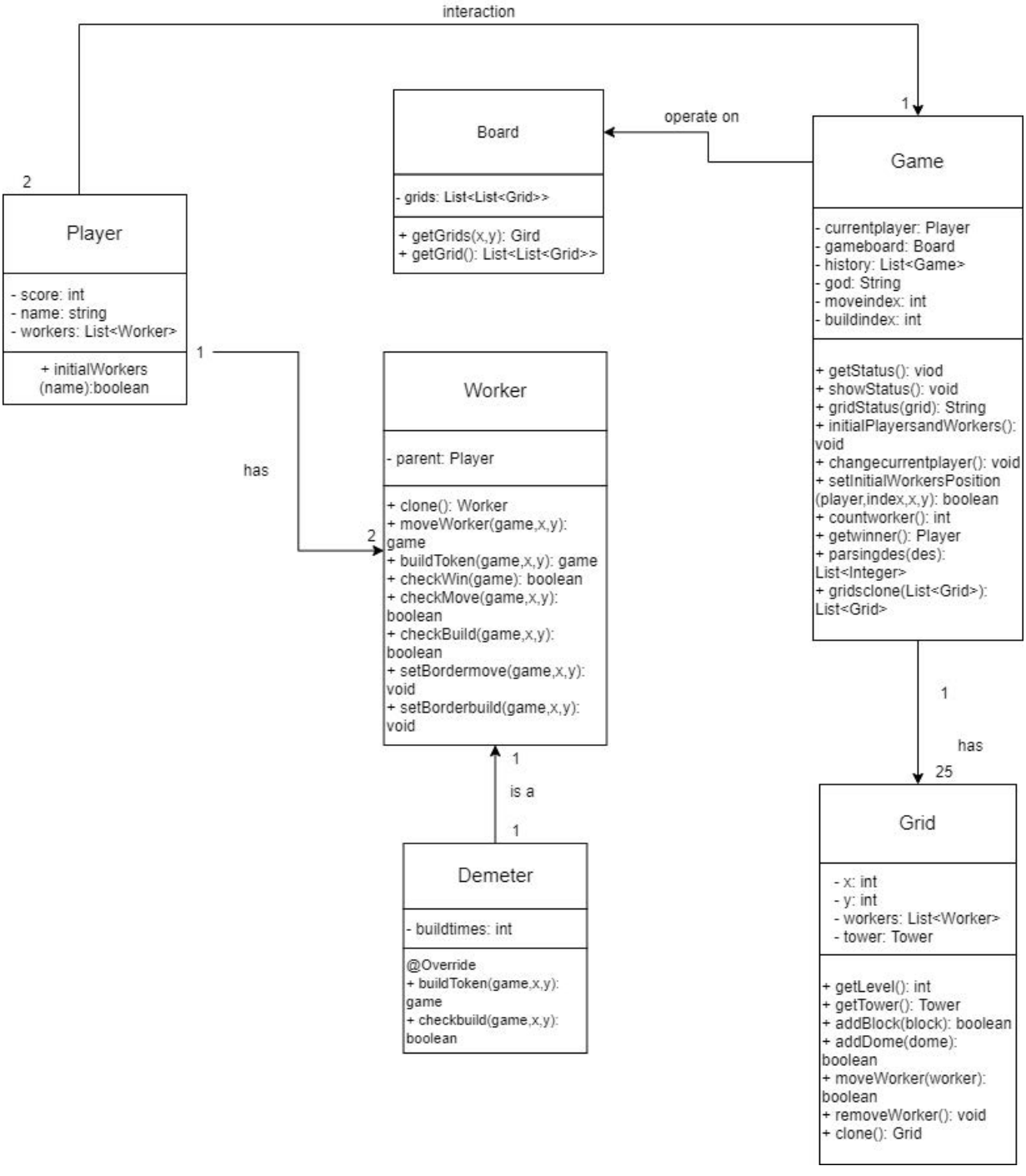
The object model of build token is shown as follows:



4. How does your design helps solve the extensibility issue of including god cards? Please write a paragraph (including considered alternatives and using the course's design vocabulary) and embed an updated object model (only Demeter is sufficient) with the relevant objects to illustrate. (Homework 5 version)

To increase extensibility and minimize coupling between objects, I design the god class as a subclass of the class worker. In the superclass worker, there are many functions to achieve movement and building token operations. The function to determine victory is also written in the worker class. Therefore, when I implement new god card, I only need to override the function to realize different function of operations. For example, to implement Demeter, I only need to override the function of buildtoken() in worker to change the build times from once to twice. And I don't need to change any other programs which considerably avoiding unnecessary coupling in program.

The object model is shown as follows:



5. What design pattern(s) did you use in your application and why did you use them? If you didn't use any design pattern, why not? (Homework 5 version)

I mainly implemented controller pattern, template method pattern and state pattern.

In controller pattern, the program is composed by three parts: model, controller and view. In this project, the frontend web is the view part and the backend is the model part. The class of game work as a controller. The game class would receive the operation from GUI and converts it to the model. The controller would respond to the user and perform interactions on model object.

The implementation of god class uses the template method. The god class only override some functions of worker class. The whole algorithm in worker is selecting worker, moving worker, building token and checking winner. The god class only need to simply override any of these sections as required.

For the implementation of undo operation, I use state pattern. For each operation, the game model would have a state which records all the attributes in a game. When we take the next step, the state of the game is transferred and the previous state is stored in history variable which is a list of game class. When it implements undo, it only needs to update the game to the latest state in the state history list.