Justification

There are three classes in this game, which are Board, Cell and Player. Board has 25 Cells which represent the 25 places that the worker can stand and build tower. In the Cell, it has two fields: height and Player. If there is a Player in the Cell, that means the Cell is occupied. The height represents how many blocks are in the filed, if the height is 4, it means that there is a dome in the cell. In the Player class, Player has three fields: name, godName and GameLogic.

I use Decorator Pattern to present main logic of the Game which includes checkMoveValidity(), CheckNextMovablePosition(), checkBuild(), move(), build(), checkWinner(). The main reason I use decorator pattern is that it enhances the extensibility of the GameLogic(). Different god card can have different additional logic on top of the default game logic. I can add additional logic on each individual god card without changing the default game logic. It will also not affect logics of other god cards. Besides, if I need to add more god cards, I can just have a new class extending the GameLogicDecorator(). Modifying any godCard logic will not affect other godCards.

Template patterns is also a good candidate for the implementation of different godCards. Template method can have an abstract class which defines the basic game logic without god card. Its subclasses can override the method implementation without affecting the basic game logic, so template method can also achieve extensibility. The benefit of using decorator pattern instead of template method is that decorator pattern can allow more possibility. A great analogy of using decorator pattern is wearing clothes. When you're cold, you wrap yourself in a sweater. If you're still cold with a sweater, you can wear a jacket on top. If it's raining, you can put on a raincoat. All of these garments “extend” your basic behavior but aren't part of you, and you can easily take off any piece of clothing whenever you don't need it. Therefore, decorator pattern can be more flexible in adding functionalities. Besides, inheritance in the template method has several problems: 1) Inheritance is static. You can't alter the behavior of an existing object at runtime. You can only replace the whole object with another one that's created from a different subclass. 2) Subclasses can have just one parent class. In most languages, inheritance doesn't let a class inherit behaviors of multiple classes at the same time.