Design goals:
The Santorini game is mainly based on several key classes, including Game, Player, Worker, Board, and Space. Main func() should call the startGame() to initialize the game and Game.class is designed to take control of this game.

Principles:
There are several states exist in this game and each state corresponds to a different object. So we decouple them into several classes: players state are stored in Game.class, current player state stored by Player.class, worker location stored by Worker.class, towers stored by Space.class, winner stored by Game.class. Game.class will ask the worker to check the next valid move/build positions, and the specific method is inside Worker.class. The Game will perform state updates, check if the game is terminated, and decide the winner by calling methods in Board.class.

Design Patterns:
The overall design follows the Strategy Pattern, which intends to let the Game.class decides which methods to call next based on different scenarios. The condition is mainly implemented as "state" emulation in this design.

Extensibility:
My design decision is to let all God class extends the Worker class and implement the God interface. Since God class also has the basic functionality, i.e., move() and build(), it should use the methods in worker class to avoid code duplication and increase code reusability. To enable subtype polymorphism, we let all God classes implement the God interface; hence, we can avoid using instanceof() to manually decide which God class we are trying to use. If we need to add more God classes or modify one of the God class's skills, in this design, we can achieve high cohesion and low coupling. To be specific, developers can complete all these by only modifying or adding specific God classes without impacting the rest of the project.

Design Pattern (new):
1. Strategy Design. Reason: We don't know players will choose which God class, so we need to let the program select an algorithm at runtime based on the player's choice.

2. Decorator Pattern. Reason: We don't know whether the player will choose God card or not. And if they choose it, their workers will have more powers than before, which means we need to extend the functionality of these workers. The decorator pattern enables high extensibility which makes the workers' powers easy to extend.