Justification for handling state

Below, describe where you stored each of the following states and justify your answers with design principles/goals/heuristics/patterns. Discuss the alternatives and trade-offs you considered during your design process.

Players

Players is stored in the Game class. In Game class, we will have a playerList, which stores the player object of each player.

- Justification:

High Cohesion: playerList allows the Game class to track all players involved in the game. This is crucial for managing player turns and other player-related game logic.

Easy to Extend: If the game rules allow for more players to join, or if there are variations in the number of players, playerList can be easily modified to accommodate these changes.

- Alternatives & Trade-offs:

we can creating a PlayerManager class to manage players. This can help further decouple game logic from player management. However, I think the main task of the Game class is to handle the rotation between players and set the winner. Introducing a new PlayerManager class will increase the overall complexity of the system. We will need to understand and maintain more classes and their interactions.

Current player

Current player is stored in the Game class, and is identified by by an player object.

- Justification:

High Cohesion for Game class: The Game class is responsible for controlling the game flow, and the currentPlayer is a key component of the game flow. Placing the currentState in the Game class allows the Game class to focus more on its responsibility, which is managing the game process.

Low Coupling: The logic related to player rotation is encapsulated within the Game class. Other classes do not need to know the detailed information about player rotation; they only need to interact with the public interface of the Game class. For example, the Player class does not need to directly manage or query who the current player is, reducing direct interaction between classes.

- Alternatives & Trade-offs:

We can set a boolean value isCurrentPlayer in the Player class to indicate whether it is the current player. However, with this design, every time when Game wants to get the current player, it needs to interact with the Player class. Also,isCurrentPlayer value in each player objects needs to be modified everytime we change the player, making the code complex and not achieving low coupling.

Worker locations

Worker locations is stored in the Worker class, I use two integer variables x and y to represent the current location.

- Justification:

Cohesion: Placing the location information of a Worker within its own class helps to operate the location data and behavior related to the Worker together. This ensures the privacy and integrity of the object's state, allowing modifications to its location information only through methods of the Worker class.
Information Expert: Worker is the information expert regarding its own location, so it should be responsible for handling the logic related to its position, such as movement.

- Alternatives & Trade-offs:

An alternative method is to track the positions of all his workers in the Player class. This may help the Player class better coordinate the actions of its workers. However, it may make the Player class overly large and complex, handling logic that does not belong to it, thus reducing its clarity and cohesion.

Towers

The tower state is recorded in the Grid class, using the height parameter in each grid to represent the tower's height. If height=1, it means that no tower has been built on that grid. If height=4, it indicates that the tower has already added a dome and cannot be increased in height further.

- Justification:

High Cohesion: By managing the tower height within the Grid class, we can maintain high cohesion within the class. This class encompasses all operations and information related to grid cells, making it easy to maintain and understand.

- Alternatives & Trade-offs:

It is possible to create a separate Tower class specifically to manage the tower's state.
The trade-off is that this would increase the complexity of the system and could potentially lead to interface dependencies between Tower and Grid. Additionally, having a Tower object instance for each grid cell might increase memory usage.

Winner

Winner is stored in the Game class, and is identified by an player object.

- Justification:

High Cohesion for Game class: Managing the Winner in the Game class helps maintain high cohesion, as all relevant game state and control flow are centralized within a single logical unit.
Information Expert: The Game class is an information expert in managing the game state, so it should know who the Winner is. This is because the Game class itself is responsible for monitoring and enforcing the game's winning conditions

- Alternatives & Trade-offs:

We can set a boolean value isWinner in the Player class to indicate whether it is the winner. The disadvantage of doing this is that it may violate the principle of encapsulation because now the winning logic is scattered across each Player object, and we need to check all players to determine the winner.