Build Action Justification

This document outlines the rationale behind the design decisions made for handling the build action within the game.

Implementation Process

The build action is initiated by the Player object, which sends a build request to the Game object. The Game object then interacts with the Board object to retrieve the target Cell using getCell(buildX, buildY). Upon receiving the target Cell, the Game object invokes the getWorker(workerId) method on the Player object to retrieve the specific Worker that will perform the build. Once the Worker is obtained, the Game object calls the buildAt(targetCell) method on the Worker, which in turn calls buildBlock() on the Cell object to execute the build.

Objects and Methods Responsible

- Player Object: Initiates the build action.
- Game Object: Acts as a controller, orchestrates the build process.
- Board Object: Provides access to the Cell objects.
- Cell Object: Contains the state and logic for building blocks or domes.
- Worker Object: Performs the build action on the Cell.

Design Justification

The assignment of responsibilities adheres to the following design principles:

- Information Expert: The Cell object, as the expert on its state, is responsible for changing its state when a build action occurs.
- Low Coupling: The Game object mediates the interaction between the Player, Worker, and Cell, minimizing direct dependencies.
- High Cohesion: Each object is only responsible for actions closely related to its role in the game, ensuring that the classes remain focused and manageable.

Alternatives and Trade-offs

The following alternatives were considered in the design process:

- Direct Interaction: Initially, allowing the Player to interact directly with Cell was considered. However, this was discarded to prevent high coupling between Player and Cell.
- Game Manages Builds: Having the Game object manage the build actions directly on the Cell was also considered. This was rejected in favor of having the Worker perform the action to keep the Game object's responsibilities from becoming too broad and to maintain the integrity of the domain model where the Worker is the entity that interacts with the Cell.

The selected design provides a balance between encapsulation, ease of understanding, and flexibility for future extensions, such as adding new types of build actions or rules.