Build Action Justification

This document outlines the rationale behind the design decisions made for handling the build action within the game, including the behavior when the active player has the Demeter god card.

Determining Valid Builds

Initial Check for Build Action Availability:

The process starts with the Game object initiating a build action by checking if the player has available build points. This is achieved by calling the checkBuildPointsAvailable() method on the Player object. The use of this method ensures that build actions are only attempted when they are permissible, aligning with the principle of ensuring action validity before execution.

Validation of Build Location:

The validation process continues with the Worker object's validateBuild(targetCell) method. This method checks if the targeted cell is adjacent, unoccupied, and does not contain a dome. The responsibility for this check is logically assigned to the Worker object because it is the entity attempting the build, adhering to the design principle of encapsulation and single responsibility.

Executing the Build (Base Game)

Building on the Target Cell:

Upon validation, the Worker object instructs the Cell object to execute the build action through the buildBlock() method. This demonstrates a clear separation of concerns, where the Cell object is responsible for modifying its own state, aligning with the encapsulation principle.

Handling Cell State Changes:

The Cell object then determines the appropriate action based on its current state: if the cell's height is less than 3, it increases the height by calling increaseHeight(). Otherwise, it builds a dome with buildDome(). This decision-making process is encapsulated within the Cell, ensuring that the object itself manages its state changes, adhering to the information expert principle.

Executing the Build (Demeter God Card):

Handling Additional Build Action:

When the active player has the Demeter god card, the build action is modified to allow for an additional build. The Game object checks if the player's associated god card is an instance of the DemeterGodCard class by calling the getGodCard() method on the Player object. If it is, the Game object invokes the onBeforeBuild(), onBuild(), and onAfterBuild() methods on the DemeterGodCard object.

Modifying Build Points:

The DemeterGodCard object overrides the onBeforeBuild() method to grant an additional build point to the player by calling the setBuildPoints() method on the Player object. This adheres to the **open-closed principle**, allowing the behavior to be extended without modifying the base game logic.

Validating and Executing the Second Build:

The DemeterGodCard object also overrides the onBuild() method to handle the additional build action. It performs the same validation checks as the base game using the validateBuild() method on the Worker object. If the second build is valid and targets a different cell than the first build, the buildBlock() method is called on the targeted Cell object to execute the build.

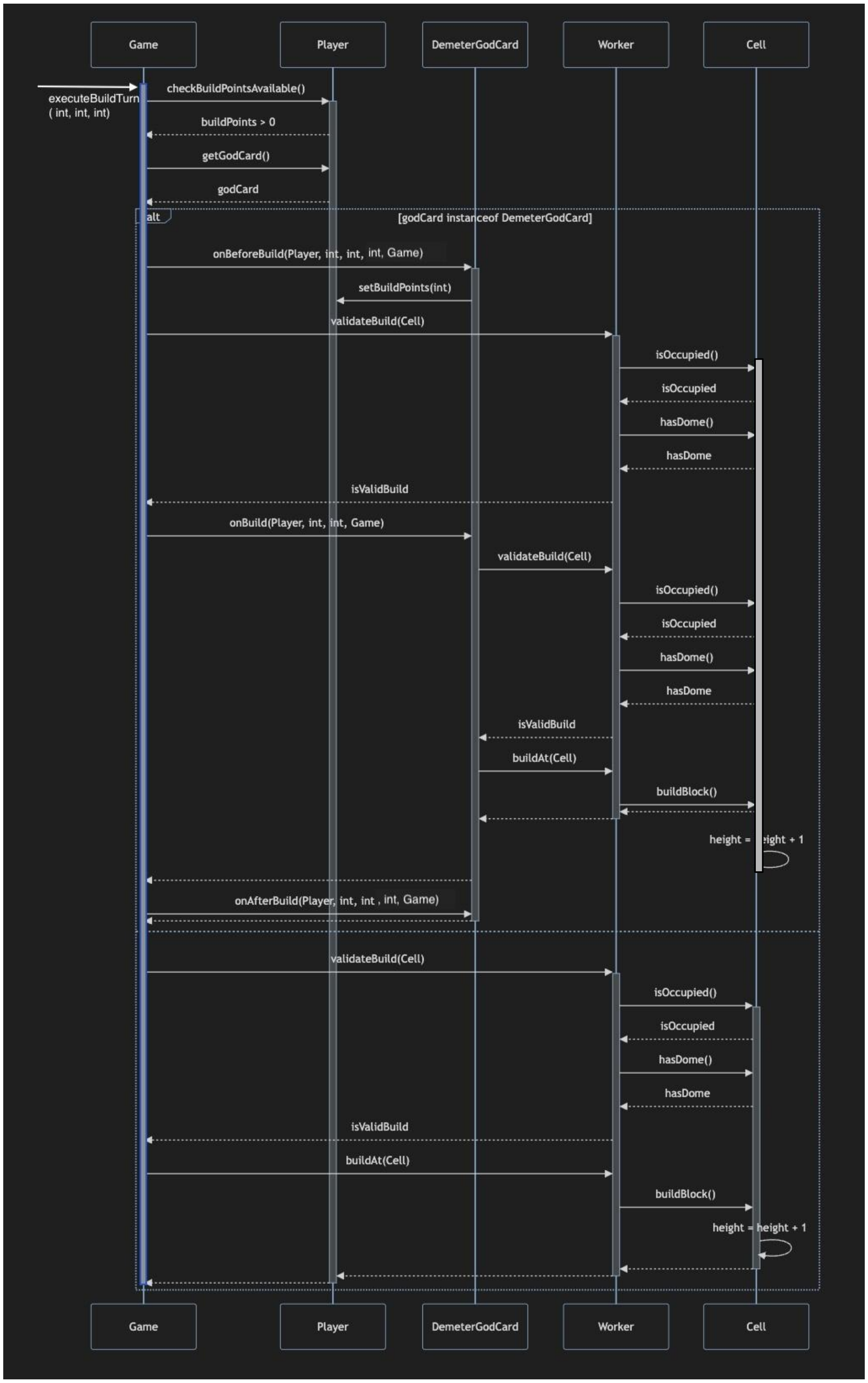
Discussion of Design Decisions

- The responsibility assignments and method interactions are justified using design principles such as encapsulation, single responsibility, and information expert. These assignments ensure that each class or object manages information and actions that are directly related to its purpose, minimizing dependencies and simplifying the overall design.

- When the active player has the Demeter god card, the build action is modified to allow for an additional build. The Game object checks if the player's associated god card is an instance of the DemeterGodCard class by calling the getGodCard() method on the Player object. If it is, the Game object invokes the onBeforeBuild(), onBuild(), and onAfterBuild() methods on the DemeterGodCard object.

- The interaction between the base game and the Demeter god card is clearly defined through the **use of template methods** in the GodCard abstract class. The Game object invokes the appropriate methods on the DemeterGodCard object, allowing for seamless integration of the god card behavior into the build action flow.

- Alternatives, such as centralizing the validation and execution of build actions within a single class, were considered. However, distributing responsibilities among the Player, Worker, and Cell objects was chosen to enhance modularity and ease of maintenance. This decision illustrates engagement with design trade-offs, prioritizing a design that supports future extensions and modifications with minimal impact on existing code.


Game
Player
DemeterGodCard
Worker
Cell
executeBuildTurn (int, int, int)
checkBuildPointsAvailable()
buildPoints > 0
getGodCard()
godCard
alt
[godCard instanceof DemeterGodCard]
onBeforeBuild(Player, int, int, int, Game)
setBuildPoints(int)
validateBuild(Cell)
isOccupied()
isOccupied
hasDome()
hasDome
isValidBuild
onBuild(Player, int, int, Game)
validateBuild(Cell)
isOccupied()
isOccupied
hasDome()
hasDome
isValidBuild
buildAt(Cell)
buildBlock()
height = height + 1
onAfterBuild(Player, int, int , int, Game)
validateBuild(Cell)
isOccupied()
isOccupied
hasDome()
hasDome
isValidBuild
buildAt(Cell)
buildBlock()
height = height + 1
Game
Player
DemeterGodCard
Worker
Cell