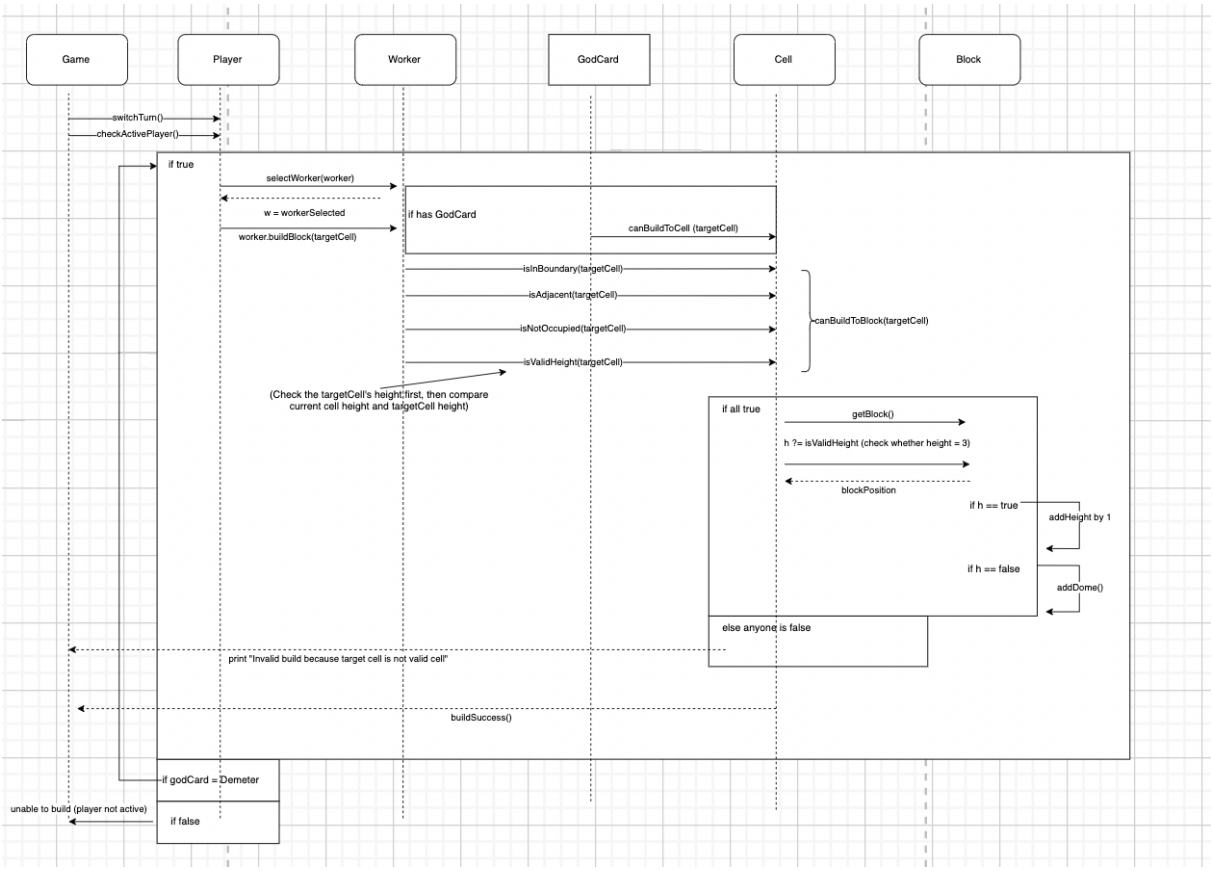
Valid Build Determination

When the active player has the **Demeter** card, the process for determining a valid build follows the same base criteria for all workers (e.g., neighbor block check, height limit, dome addition). However, the **Demeter** card introduces a special rule: the worker may perform an additional build action, but the second build must occur on a different cell than the first build during the same turn.

1. **Demeter GodCard Class**: Tracks the lastBuiltCell after the first build action during a player's turn.
2. **Validation Check**: The overridden canBuildToCell(cell) method ensures that the second build cannot occur on lastBuiltCell while incorporating the standard criteria:
    - **Neighbor Block Check**: The target cell must:
        - Be within the boundaries of the board (5x5 grid).
        - Be a neighbor of the worker's current position (adjacent within one cell).
        - Not be the worker's current position.
    - **Height Limit**: The target cell's block height must be less than 3 for a standard block.
    - **Occupied Cell**: A cell with a dome or another worker is invalid for building.
    - **Second Build Rule (Demeter)**: The second build must occur on a different cell than the first.



Perform Block Build and Update of State

The **Worker** initiates the build process using buildBlock(cell), but if the player has the Demeter card:

1. **First Build**: Proceeds as per standard rules.
2. **Second Build**: After the first build, the Demeter card sets lastBuiltCell to the first target cell. The **Game** allows the second build, calling canBuildToCell(cell) with the added condition enforced by Demeter to exclude lastBuiltCell.
3. The **Block** class manages height updates:
    - Calls getBlock() to retrieve the block state.
    - If the height is less than 3, increments the height.
    - If the height equals 3, adds a dome to the block.

If the cell does not pass validation, the system provides feedback and prompts the player to select another cell.

Responsible Objects and Methods

1. **Block Class**:
    - buildBlock(cell): Increases height or adds a dome based on current state.
    - addDome(cell): Updates the state to indicate a dome is present.
2. **Worker Class**:
    - buildBlock(cell): Initiates the building process on a target cell.
3. **GodCard Interface** (implemented by Demeter):
    - applyBuildRule(worker, cell): Enforces Demeter-specific rules for the second build.
    - canBuildToCell(cell): Validates the cell based on GodCard rules.
4. **Game Class**:
    - Coordinates player turns and enforces GodCard-specific rules by using the assigned GodCard.

Justification of Design Choices

1. **Encapsulation**: Each class handles its specific responsibilities:
    - The **Block** class encapsulates all block state logic, ensuring clear separation of concerns.
    - The **Worker** class manages worker actions, delegating GodCard-specific rules to the GodCard class.
2. **Single Responsibility Principle (SRP)**:
    - **GodCard Interface**: Abstracts unique rules for build and move actions.
    - **Demeter Class**: Implements Demeter-specific rules, ensuring modularity and reusability.

3. **Open/Closed Principle (OCP)**: The system supports the addition of new GodCards without modifying existing logic. For example, adding a new GodCard only requires implementing the GodCard interface.
4. **Delegation**: By delegating rule enforcement to the GodCard class, the system remains modular and easier to test.

Alternatives Considered and Trade-offs

1. **Centralized Logic in Game Class**:
    - Pros: Simplifies coordination between different entities (Workers, Blocks, etc.).
    - Cons: Leads to a bloated and less maintainable Game class. This alternative was rejected to ensure separation of concerns.
2. **Worker-Based Rule Enforcement**:
    - Pros: Simplifies Game logic by pushing responsibilities to the Worker class.
    - Cons: Makes the Worker class overly complex and harder to extend. This alternative was also rejected.

By isolating the GodCard logic, the system achieves extensibility and clarity while adhering to design principles like SRP and OCP.