Deliverable 6:

Implementation of Valid Build Determination:

Objects and Methods Involved:

- GameController: Receives input and invokes build actions on the Player.
- Player: Responsible for selecting a worker and executing build actions through the worker.
- Worker: Performs the build action on a target Cell.
- Cell: Contains the isValidBuild(Cell workerCell, Cell targetCell) method to determine if the build action on the target cell is valid.
- Grid: Provides the getAdjacentCells(Cell cell) method to retrieve adjacent cells.
- Tower: Responsible for managing the tower structure on a cell, with methods addLevel() and addDome() to modify the tower.

Process:

- GameController receives a build command, including coordinates for the build location.
- GameController delegates the action to the current Player, identifying the selected
- Worker based on the game's state.
- The Player instructs the selected Worker to perform the build action on the target Cell.
- The Worker checks with the Cell if the build is valid through Cell.isValidBuild().
- Inside isValidBuild(), the following checks are performed:
  - Check if the workerCell and targetCell are not null.
  - Check if the targetCell is not occupied by a worker using targetCell.hasWorker().
  - Check if the targetCell does not already have a dome using targetCell.getTower().hasDome().
  - Retrieve the adjacent cells of the workerCell using grid.getAdjacentCells(workerCell).
  - Check if the targetCell is in the list of adjacent cells.
- If all checks pass, the build is considered valid.
- Upon validation, the Cell updates its Tower, either by adding a level using addLevel() or a dome using addDome().

Justification and Design Principles:

Single Responsibility Principle:

- Each class has a clear, distinct responsibility. Cell manages cell-related rules, Tower manages structural changes, and Worker initiates the action.
- This separation enhances maintainability and readability of the code.

Encapsulation and Information Hiding:

- The internal state of each tower is hidden and managed through public methods like addLevel() and addDome().
- This design encapsulates behavior and protects the integrity of the game's rules.

Low Coupling:

- Classes are designed to minimize dependencies. Worker doesn't directly manipulate the Tower; it goes through Cell, adhering to the game's rules of interaction.
- The Grid class is used to retrieve adjacent cells, reducing the coupling between Cell and its neighbors.

- This design allows for easier modifications and testing.

Law of Demeter:

- By having GameController initiate actions, and pass to closest neighbor, the system adheres to the Law of Demeter, with each part interacting with only closely related objects.

Alternatives and Trade-offs:

Direct Worker-Tower Interaction:

- Considered allowing Worker to interact directly with Tower for building.
- Rejected because it would violate encapsulation and increase coupling, making future changes more complex.

Direct Player Invocation:

- Directly invoking build actions from the Player was considered but could lead to a tighter coupling between the player input and the player actions, reducing flexibility in command processing.

Grid-Based Validation:

- Another approach was to have the Grid class validate build actions.
- This was seen as overloading the Grid with excessive responsibilities, moving away from the Single Responsibility Principle and low coupling.
- Player could also be associated with grid to have easier move/build() function, but was discarded because I wanted to keep low coupling.

Enhanced Worker Autonomy:

- Giving more autonomy to Worker objects to determine valid actions was evaluated. However, centralizing decision-making in GameController and Player maintains a clearer separation of concerns and game logic encapsulation.

Conclusion:

The chosen design for the building action in Santorini respects core object-oriented principles like Single Responsibility, Encapsulation, and Low Coupling, Law of Demeter. It effectively distributes responsibilities among the GameController, Player, Worker, Cell, Grid, and Tower classes.

The isValidBuild method in the Cell class performs the necessary checks to determine the validity of a build action, including adjacency, worker occupancy, and dome presence. It leverages the getAdjacentCells method from the Grid class to retrieve adjacent cells efficiently. Incorporating GameController as the initiator of build actions introduces a structured approach to handling game commands, aligning with design patterns that enhance maintainability, scalability, and clarity. This design choice, while adding a layer between player input and action execution, provides a robust foundation for game logic and future UI integration, ensuring that the game's architecture remains flexible and adaptable to new requirements.

Object Interaction diagram on next page

Deliverable 6 : object-level interaction diagram