Extensible Design Justification

To support God cards in my Santorini implementation, I refactored the game using a modular and extensible design centered around the **Strategy** and **Factory** design patterns.

I identified three core aspects of player behavior that vary between Gods—**movement rules**, **build behavior**, and **win conditions**—and extracted each into its own interface: `MoveStrategy`, `BuildStrategy`, and `WinConditionStrategy`. Each God card is then implemented as a composition of these strategies. This allows the game logic to interact with generic interfaces while each God modifies behavior through custom strategy implementations. For instance, Demeter extends `BuildStrategy` to allow a second build on a different space, while Minotaur overrides `MoveStrategy` to push opponents. To assign gods at runtime, I introduced a `GodFactory` that centralizes creation of God compositions based on their names.

I chose this design because it promotes **Separation of Concerns** by isolating each behavior dimension. I have considered alternatives such as a monolithic `GodPower` interface that handled all actions in one method, but it proved too rigid and error-prone for Gods with partial or optional overrides. The per-behavior Strategy approach was preferred as it enables fine-grained customization and reuse.