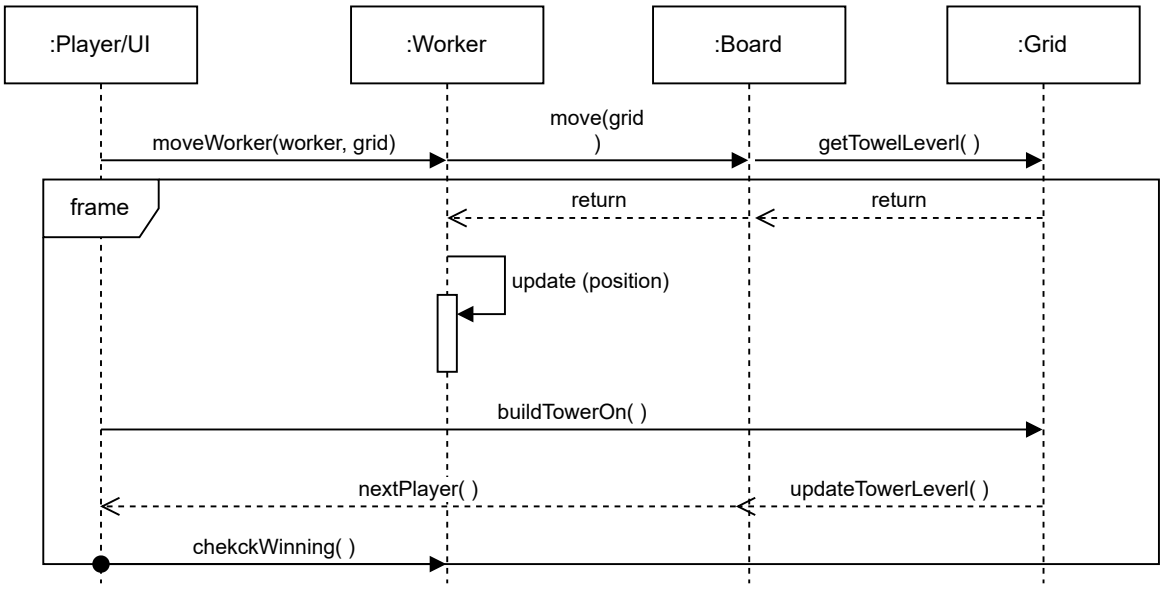
Obeject-Level interaction diagram



Building Action Implementation:
The game determines a valid build through a series of checks within the Grid class. The isOccupied() and hasDome() methods ascertain if a block or dome can be placed. The Player class's buildTowerOn() method is invoked to initiate the action, showcasing encapsulation and single-responsibility principles.

Actions and Responsibilities:

Worker class: Calls move() which interacts with the Grid to change the worker's position.
Board class: Utilizes getTowerLevel() to fetch the current tower level from the Grid.
Grid class: Uses updateTowerLevel() to modify the grid state post-build.
Justification:
Assigning the build responsibility to the Player class aligns with our design goal of keeping gameplay logic centralized within player actions, thus simplifying the interaction between the UI and the game logic. The Grid class remains a passive object, consistent with the principle of high cohesion within classes.

Alternatives and Trade-offs:
An alternative design considered was to place the build logic within the Grid class. However, this was dismissed as it would mix the responsibilities of state maintenance and gameplay mechanics, potentially leading to a violation of the single-responsibility principle and making future extensions more complex.

Design Process:
The chosen design was iteratively refined, starting with a simple implementation that evolved as the need for a more cohesive and maintainable codebase became apparent. Regular reviews and refactoring sessions led to the final design, ensuring that the codebase adheres to object-oriented design principles.

Interaction Diagram:
The accompanying object-level interaction diagram illustrates the sequence of method calls for a build action, detailing the collaboration between objects to achieve a build and highlighting the clear separation of concerns.