# Initializing the Game Board

**Step Overview:**

1. When the game is initialized, the game board is set up with a specified number of opponents, and polyomino shapes are randomly placed on the board.

**Explanation:**

- The `gameplay` class is responsible for initializing the game.
- `gameplay` creates an instance of the `gameBoard` class and specifies the number of opponents.
- The `gameBoard` class, during initialization, generates polyomino shapes using the `generatePolyominos` method.
- Polyomino shapes are randomly placed on the board using the `getRandomPolyominoShape` method.
- `Opponent` objects are created and assigned specific IDs, corresponding to the polyomino shapes on the board.

**Classes Involved:**

- `gameplay` class manages the overall game and interacts with the `gameBoard` class.
- `gameBoard` class handles the initialization of the board and generation of polyomino shapes.
- `Opponent` class represents individual opponents with specific IDs and fort configurations.

# Processing Player's Move

**Step Overview:**

1. The player inputs a move, and the system processes the move, updating the board, checking for hits or misses, and handling opponent fort damage.

**Explanation:**

- The `gameplay` class handles the player's move processing.
- The `convertMoveToMatrixCoordinates` method converts the user's input to matrix coordinates (row, column).
- The `playerHit` method updates the game board based on the player's move and checks for hits or misses.

- If a hit occurs, the corresponding opponent's fort is damaged using the `decreaseUndamaged_forts` method in the `Opponent` class.
- If all forts of an opponent are damaged, the opponent is considered defeated.
- The system alternates turns between the player and opponents, updating the `turn` attribute in the `gameplay` class.

**Classes Involved:**

- `gameplay` class manages the game flow and calls methods to process the player's move.
- `gameBoard` class stores the current state of the board and is updated by the player's move.
- `Opponent` class represents opponents with unique IDs and fort configurations.
- `convertMoveToMatrixCoordinates` is a utility method for translating user input.

This design ensures that each class has a well-defined responsibility, promoting modularity and maintainability. The `gameplay` class acts as a controller, coordinating interactions between the user, game board, and opponents, while the `gameBoard` and `Opponent` classes handle specific aspects of the game state and opponent behavior.