Updated behavioral contract for moving a worker based on the new implementation changes, assuming the active player has the Minotaur card, and the opposing player has no card:

Behavioral Contract for Moving a Worker

• Operation: playerMove(int toRow, int toCol)

• Cross References: Use case is when a player tries to move a worker.

• Preconditions:

o Game Initialized: The game must be initialized with two players, each having two workers placed on the grid.

o Valid Turn: It must be the turn of the player attempting the move. Only the current player can move a worker.

o Worker Selection: A worker belonging to the current player must have been selected based on its coordinates on the grid.

o Target Cell Validity: The target cell for the worker's move must be within the grid bounds (0 to 4 for both row and col in a 5x5 grid).

o Minotaur God Card: The active player must have the Minotaur god card (The hw6b rubric required to have Minotaur for this move worker contract, otherwise the playerMove() function validates the godCard that is passed in).

• Postconditions:

o Worker's New Position: If the move is valid, the worker's position is updated to the target cell.

o Opponent Worker's Position: If the target cell is occupied by an opponent's worker, the opponent's worker is pushed one space straight backwards to an unoccupied space at any level.

o Cell Occupancy Update: The target cell is now occupied by the worker, and the worker's previous cell is vacated. If an opponent's worker was pushed, their previous cell is also vacated.

o Game State Update: The move might trigger a change in the game state, such as checking for a win condition or changing the turn to the next player if the move ends the current player's turn.

Below is the behavioral contract for every sub-method invoked by playerMove(int, int):

1. Handling the Move Request (GameServer.serve(IHTTPSession session)):

- Operation: GameServer.serve(IHTTPSession session)
- Cross References: Use case is when the frontend sends a move request to the backend.
- Preconditions:
  - o The request URI should be "/api/game/move-worker".
  - o The request method should be POST.
  - o The request should contain the parameters "toRow" and "toCol" representing the target cell coordinates.
- Postconditions:
  - o The GameController's playerMove(int toRow, int toCol) method is called with the provided coordinates.
  - o The response is sent back to the frontend with the updated game state.

2. Moving the Worker (GameController.playerMove(int x, int y)):

- Operation: GameController.playerMove(int x, int y)
- Cross References: Use case is when the backend processes the move request.
- Preconditions:
  - o The current player must have a selected worker.
  - o The target cell must be within the grid bounds.
- Postconditions:
  - o The moveWorker(Grid grid, Worker worker, Cell targetCell) method of the current player is called with the grid, selected worker, and target cell.
  - o If the move is successful:

- The afterMove(Grid grid, Worker worker, Cell targetCell) method of the current player's god card is called.

- The checkGameStatus() method is called to check for any win conditions.

o If the move is unsuccessful, an error message is added to the messages list.

3. Validating and Performing the Move (Player.moveWorker(Grid grid, Worker worker, Cell toCell)):

• Operation: Player.moveWorker(Grid grid, Worker worker, Cell toCell)

• Cross References: Use case is when the player attempts to move their worker.

• Preconditions:

o The worker must belong to the current player.

o The move(Grid grid, Cell toCell, GodCard godCard) method of the worker must return true.

• Postconditions:

o If the move is successful, a success message is printed.

o If the move is unsuccessful, an error message is printed.

4. Executing the Move (Worker.move(Grid grid, Cell toCell, GodCard godCard)):

• Operation: Worker.move(Grid grid, Cell toCell, GodCard godCard)

• Cross References: Use case is when the worker performs the move action to the target cell.

• Preconditions:

o The godCard.isSpecialMove(Grid grid, Worker worker, Cell targetCell) method must return true for the Minotaur god card.

o The grid.isValidMove(Cell fromCell, Cell toCell, boolean isMinotaurMove) method must return true.

• Postconditions:

o If the target cell is occupied by an opponent's worker (Minotaur's special move):

- The opponent's worker is moved to the cell behind the target cell using grid.getCellBehind(Cell fromCell, Cell toCell).

- The opponent's worker's moved state is set to false using opponentWorker.setMoved(false).

o The worker's position is updated to the target cell using setPosition(Cell toCell).

o The worker's active state is set to true.

5. Validating the Move (Grid.isValidMove(Cell fromCell, Cell toCell, boolean isMinotaurMove)):

- Operation: Grid.isValidMove(Cell fromCell, Cell toCell, boolean isMinotaurMove)
- Cross References: Use case is when validating the move from one cell to another.
- Preconditions:

o fromCell and toCell must not be null.

o toCell must not have a dome.

o toCell must be adjacent to fromCell.

- Postconditions:

o If isMinotaurMove is false (regular move):

- toCell must not have a worker.

- The level difference between fromCell and toCell must be less than or equal to 1.

o If isMinotaurMove is true (Minotaur's special move):

- If toCell has a worker, the cell behind toCell must exist, not have a worker, and not have a dome.

- If toCell does not have a worker, the level difference between fromCell and toCell must be less than or equal to 1.

6. Checking Game Status (GameController.checkGameStatus()):

- Operation: GameController.checkGameStatus()

- Cross References: Use case is after the worker has moved to a new cell.
- Preconditions:
  - The worker has been moved to a new cell.
- Postconditions:
  - The checkWinCondition() method is called for each player.
  - If any player's win condition is met, that player is set as the winner using setWinner(Player player).

7. Checking Win Condition (Player.checkWinCondition()):

- Operation: Player.checkWinCondition()
- Cross References: Use case is when checking if the player has won the game.
- Preconditions:
  - The player's workers have been updated with their new positions.
- Postconditions:
  - If any of the player's workers is at a position with a tower level equal to the win condition (e.g., 3) and has moved, the method returns true.
  - If the player's god card has a special win condition (e.g., Pan moving down two levels), the god card's checkWinCondition(Worker worker) method is called for each worker. If it returns true, the method returns true.
  - If no win condition is met, the method returns false.

This contract covers the flow of moving a worker from the frontend request to the backend processing, assuming the active player has the Minotaur god card. It includes the preconditions and postconditions for each relevant method involved in the process.