Use Case Description and Diagram

- **Enter Move**
  - The user inputs a peg pair (one peg that moves and one peg that will be jumped or removed) into the system.
  - User submits the peg pair as a move via a button on the GUI.
  - System validates move. If valid, the system updates the GUI and program backend as required.
  - Now, the system remains idle waiting for user input.

- **Optimally Solve Puzzle**
  - The user selects the solve puzzle button
  - The system uses its algorithm and solves the puzzle in an optimal fashion
  - System, now asks the user if he/she wishes to continue or shut the game down.

- **Clear / Reset**
  - The user selects the clear/reset button
  - The system reinitializes itself.
  - System clears display of old game board state and shows initial board state

Alternative Events

- **Invalid Move**
  - The user inputs a peg pair into the system
  - User submits the peg pair as a move
  - The system determines that the move input is invalid (is not an integer, is not a peg number, the move is not possible, etc.)
  - A system error message is displayed to the user
  - The player is asked to re-input the move
  - Player re-submits a new move
  - System validates the move and waits for input if move is valid. Otherwise the process is repeated.

- **Final Move**
  - While validating the move input by the user the system concludes that this move is a final move, meaning that no other moves exists.
  - The system takes note of the number of remaining pegs on the game board. If the pegs remaining equals one, the solution as given by the player was successful. Otherwise if pegs remaining is greater than one, an end to the game has been reached but the solution was not successful.
Exceptional Events

- **Optimally Solve Puzzle During User Game Play**
  - The user has input at least one move into the system.
  - User decides to select the solve puzzle button.
  - System issues an error message to the user stating that it cannot solve the puzzle optimally after a user has begun the game. That is to say that the system cannot guarantee that its solution will optimal and there will be one peg remaining after it has terminated its algorithm.
**Sequence Diagrams**

**Enter Move**

1. User enters force and target peg number, presses move button.
2. GUI validatesMove(force, target).
4. Moves return true.
5. Moves jumpPeg(force, target).
7. Moves updateDisplayBoard().

**Optimally Solve Puzzle**

1. User selects the solve puzzle button.
2. GUI calls solvePuzzle().
3. Gameboard getsBoardSize() returns size.
4. Moves RegSolitaire(size) returns.
5. UpdateDisplayBoard().
6. Replay().
Optimally Solving the Puzzle During Game Play

User

selects the solve puzzle button

GUI

solvePuzzle()

Gameboard

numberofPlays()

return >1

return

updateMsgCenter()