4003-242: Data Structures for Problem Solving, Spring 2013

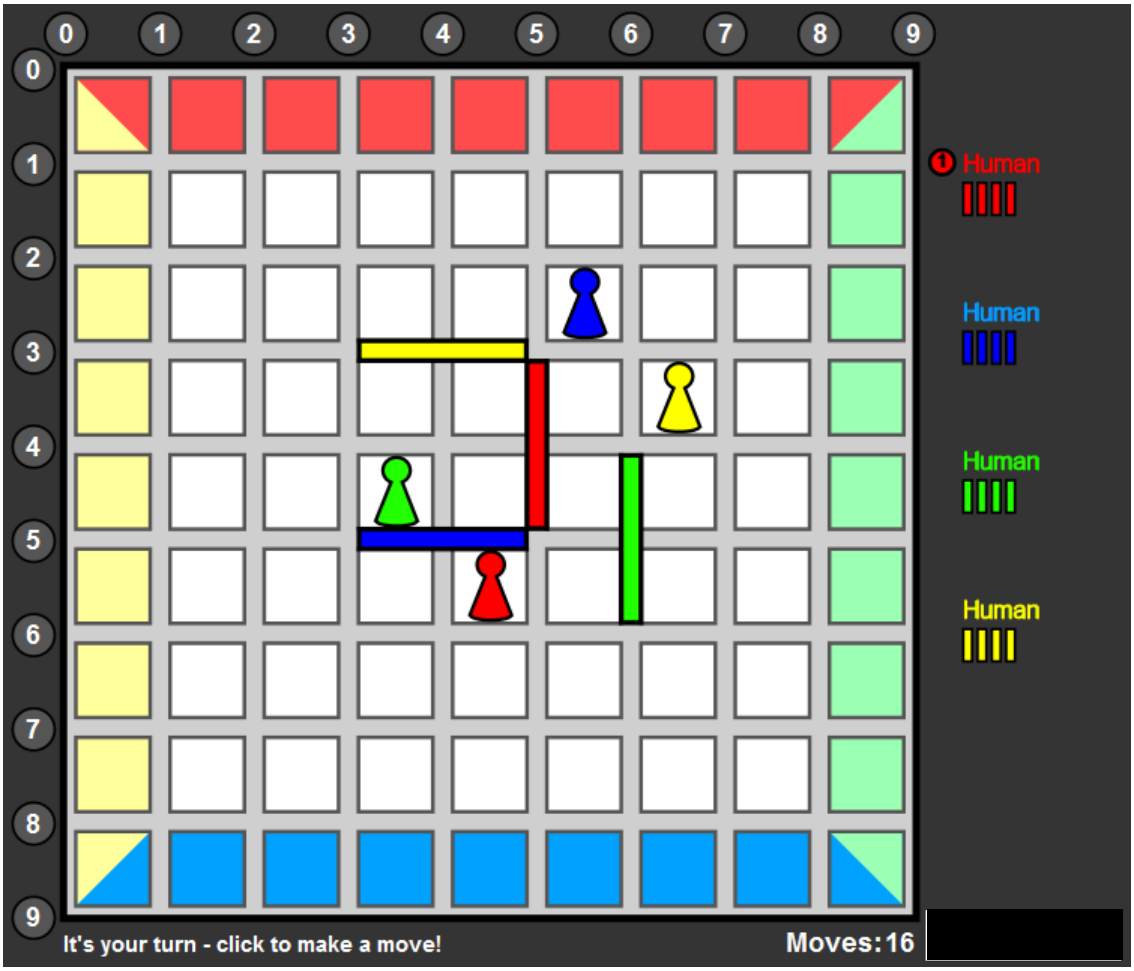
QUORIDOR: Project Part 3

1 Problem Solving Session

Part 3 of the project handles multiple players: your player will play against 1 or 3 other players.

Your design needs to handle pawn "jumps." The game rules allow a pawn to jump over another pawn. A straight jump, if possible, is always allowed. An L-shaped jump is not allowed if a straight jump can be made. See `https://[redacted]royale/`, under Game Rules, for figures of valid jumps. A pawn can jump over only one other pawn, not two or three.

Below you see a board with a 4-player game in progress. The first (red) player's destination is north, the second (blue) is south, the third (green) is east, and the fourth (yellow) is west.



1. Suppose the players make the following sequence of pawn moves:

   (a) Player 1 moves to square $(5, 5)$.

   (b) Player 2 moves to square $(3, 5)$.

   (c) Player 3 moves to square $(4, 2)$.

   (d) Player 4 moves to square $(2, 5)$.

   (e) Player 1 moves to square $(4, 5)$.

   (f) Player 2 moves to square $(2, 6)$.

   (g) Player 3 moves to square $(5, 2)$.

   (h) Player 4 moves to square $(5, 5)$.

   (i) Player 1 moves to square $(3, 6)$.

   (j) Player 2 moves to square $(3, 6)$.

   (k) Player 3 moves to square $(6, 2)$.

   (l) Player 4 moves to square $(4, 5)$.

   For each move, state whether the move is valid or not. If not, give a reason why not. For the purposes of this exercise, if a specified move is invalid, treat the move as a "pass" and keep the player in the game.

   After you finish this problem, your group should split into separate teams. Each team will hand in a separate solution to the remaining questions.

2. Develop pseudo code that finds all valid, single-turn, pawn moves for a given player. This includes jump moves.

3. Describe the strategy you plan to use to help your player win the game in part 4.

Notes

(a) In an actual game, the engine eliminates a player making an invalid move. The game continues but without the offending player. Your design must take the player's pawn out of the game and leave any already-placed walls on the board.

(b) The only time when a player is allowed to "pass" is when there is no choice for a move. This means that there are no legal positions for the pawn to move or jump to, and there are no more walls available to place. A "pass" is indicated by creating a `PlayerMove` that shows a pawn moving to its current location.

Your instructor will collect the answers to these problem-solving problems roughly at the midpoint of the problem-solving and lab period.

Part 3 Implementation

Part 3 is about running multi-player games. The teams must make sure that their modules are aware of the other players in the game.

For the most part you simply modify functions you've already written for part 2.

1. `init` should not need to change except for one important item. In the unfortunate case of a player being eliminated (due to crashing) before the call to your `init` function, that player's home will be `False` instead of its home coordinates. Although you may not need this information until you implement a strategy for winning (part 4), you do need to ensure that your code can handle this alternate type of data in the `playerHomes` parameter.

2. `last_move` is now called after other players' moves as well. Make sure your player data is being properly updated.

3. `move` must be slightly more sophisticated to make legal moves in the presence of other players' pawns on the board.

There is one new function to write: `player_invalidated`. The engine will call this function only if another player has made an illegal move or crashed. You must update your player data to remove the offending player's pawn from the board – but not its walls.

See the pydoc comments in your package's **__init__.py** file for details.

Part 4 Implementation

Part 4 is about strategy. There is nothing new that you are required to implement; you just have to retool your existing code to improve your player's chances of winning.

If your player module beats the test opponent named **BeatMe**, **90%** of the time in a two-player game, and does not crash in either two- or four-player mode, then you will get full functionality credit for this assignment.

Note: There is a NEW BeatMe that you must download from the site. Look for the post "New BeatMe Player" under `https://[redacted]royale/docs/`. Unzip the file and follow the instructions in `readme.txt`.

We will be hosting a tournament at the end of the term, but how your team's player performs in that event has no bearing on your grade. Trials for the tournament within your own section will take place in lab during the final week of the term.