LINGI2261: Artificial Intelligence
Assignment 3: Part I: Quoridor Instructions

Jean-Baptiste Mairy, Cyrille Dejemeppe, Yves Deville
October 2013

Guidelines

- These instructions are provided to give you an overview of the Quoridor game.
- The content of assignment 3 will be described in another document.
- You should become familiar with the Quoridor framework because it will be used in the third assignment.
- If you encounter a bug while using the provided framework, report it clearly to cyrille.dejemeppe@uclouvain.be.
- Before diving into coding your agent, be sure you have answered the first questions of the third assignment and that you fully understand adversarial search concepts.
- Be sure you fully understand the provided framework before you begin to code.

1 Rules

The rules provided here slightly differ from the original Quoridor rules. The original Quoridor game allows two to four players to play. For the third assignment, we consider a simpler version of the game in which only two players are allowed to play. The rules provided here are the rules for the hardware version of the game and are implemented in the provided python framework.

1.1 Components

- 1 game board made of 81 tiles (9x9).
- 2 pawns, 1 for each player.
- 20 walls, 10 for each player.

1.2 Objective of the Game

Bring your pawn at your goal row (the opposite row from where your pawn started) before your opponent reaches his goal row with his pawn.

1.3 Setting Up the Game

1. Determine who will play first (can be done randomly or according to any other discriminating criterion).
2. Place the 2 pawns on the board, each pawn being respectively at the center tile of the first and the last row of the board.

1.4 Gameplay

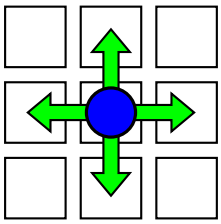
Starting with the first player, the two players take their turn alternatively. A turn consists in moving your pawn or placing a wall. When a player has no walls remaining, he has to move his pawn.

Moving Your Pawn

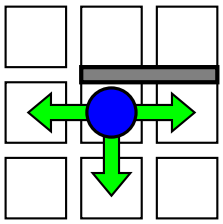
Pawns can move on a neighbouring tile, in the up, down, left or right direction (as seen in Figure 1a). The pawn must move around the walls (as seen in Figure 1b) and cannot move out of the board (as seen in Figure 1c).

Placing a Wall

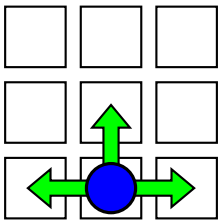
The walls must be placed between 2 sets of two tiles (as seen in Figure 2a). The walls cannot be placed such that they span 3 tiles (as seen in Figure 2b). A wall cannot be placed



(a) Possible pawn moves: up, down, left, right.



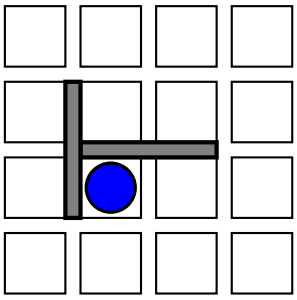
(b) Pawn must get around the walls.



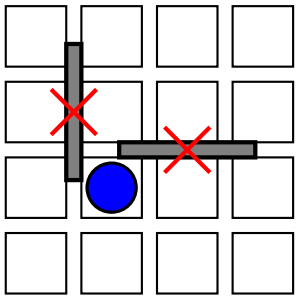
(c) Pawn cannot move out of the board.

Figure 1: Basic moves of a pawn.

such that there is no possible path between a pawn and its goal row. Walls cannot superpose each other.



(a) Possible wall placement: horizontal, vertical.



(b) Not allowed wall placement: spanning more than 2 tiles.

Figure 2: Placement of a wall.

Face to Face

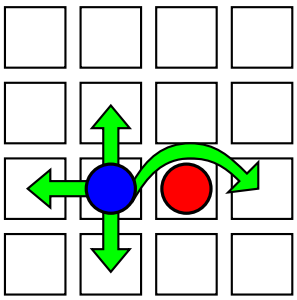
When two pawns face each other on neighbouring tiles which are not separated by a wall, the player whose turn it is can jump the opponent's pawn and place himself behind him (as seen in Figure 3a). If there is a wall behind the opponent's pawn, the player can move on one side of the opponent's pawn (as seen in Figure 3b). If another wall blocks the way to jump on one side of the opponent's pawn, then the player cannot move there (as seen in Figure 3c).

1.5 End of the Game

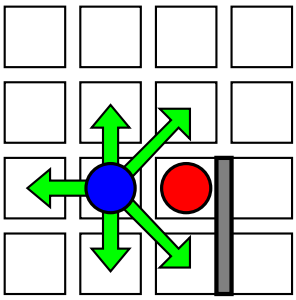
The game ends when one pawn reaches his goal row (i.e. the row on the opposite side of its starting row). The player which pawn is the first to reach his goal row is the winner.

2 Python Framework

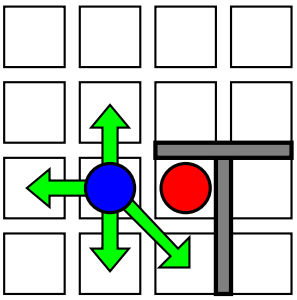
In the third assignment, you will have to imagine and implement an AI agent to play the Quoridor game.



(a) Jumping above the opponent pawn.



(b) Jumping above the opponent pawn with 1 wall.



(c) Jumping above the opponent pawn with 2 walls.

Figure 3: Moves when facing the opponent pawn.

2.1 Files provided

The `quoridor.zip` archive on the iCampus site of the course (Document › Assignments › Assignment 3) contains some Python code to get you started. The following files are provided:

- `quoridor.py`: module containing the board representation, the agent interface and some other stuff. This will be the most useful file for you, so it is recommended to have a look inside and read the specifications thoroughly.
- `minimax.py`: implementation of the MiniMax and Alpha-Beta algorithms.
- `random_player.py`: an example player making dumb random moves.
- `agent.py`: a skeleton of an Alpha-Beta agent you can use as a base for your own.
- `game.py`: main program for launching the game. Run with `-h` to show the usage notice.
- `gui.py`: graphical user interface. This file is used by `game.py`.

2.2 Internal Representation

Internally, the board is represented by different variables. The specification of these variables is described as follows:

size The size of the board. It represents the number of rows and columns on the board. The board contains `size` × `size` tiles.

starting_walls It is an integer defining the number of walls each player has initially.

pawns It is a list containing two elements which are the positions of the pawns. The first element of this list, `pawns[0]`, is a tuple (x_1, y_1) where x_1 is the row and y_1 the column at which the pawn of the first player is ((0, 0) being the top left tile of the board). Similarly, the second element, `pawns[1]`, is a tuple (x_2, y_2) representing respectively the row and the column at which the pawn of the second player is.

goals It is a list containing two elements which are the respective goal rows of the two players. To win the game, the pawn of the first player must reach the row `goals[0]` and the second player must reach the row `goals[1]`.

nb_walls It is a list containing two integers which represent the number of walls remaining for respectively the first and the second player. Each time a player places a wall on the board, its number of remaining walls is decremented by 1. If the number of remaining walls for a player reaches 0, this player cannot place any more wall.

horiz_walls It is a list containing tuples which represents the position of horizontal walls on the board. The tuples (x_i, y_i) it contains represent the row and column of the center of the horizontal walls. The row and column indices of the walls don't correspond to the row and column indices used to define the position of the pawns. Indeed the number of possible rows and columns on which a wall can be placed is inferior by 1 compared to the number of rows and columns on which a pawn can be placed (i.e. the classic quoridor board contains 9×9 tiles on which a pawn can be placed and only 8×8 position at which the center of a wall can be placed). So the (0, 0) position for a pawn doesn't correspond to the (0, 0) position of the center of a wall since you cannot place a wall on a tile.

verti_walls Similar to the `horiz_walls` variable. It contains tuples which represents the position of vertical walls on the board.

Example

An example of the state of a board during a game is provided in Figure 4. Let us now have a look at the variable defining this state:

size = 9 The board contains 9 rows $\times$ 9 columns tiles.

starting_walls = 10 Each player has started the game with 10 walls.

pawns = [(1, 6), (4, 6)] The first player's pawn (blue pawn) is in row 1, column 6 and the second player's pawn (red pawn) is in row 4 column 6.

goals = [8, 0] The goal for the first player is to reach row 8 with his blue pawn and the second player must reach the row 0 with his red pawn.

nb_walls = [4, 6] The first player has already used 6 walls (so 4 walls remain) and the second player has only used 4 walls (so 6 walls remain).

horiz_walls = [(1, 3), (1, 5), (2, 4), (3, 7), (6, 4)] These are the positions (row, column) of the center of the horizontal walls. Note that the order of the walls in this list doesn't matter.

verti_walls = [(2, 2), (2, 6), (3, 5), (5, 5), (7, 3)] These are the positions (row, column) of the center of the vertical walls. Note that the order of the walls in this list doesn't

matter.

As you can see, there is a path between the blue pawn and its goal and between the red pawn and its goal.

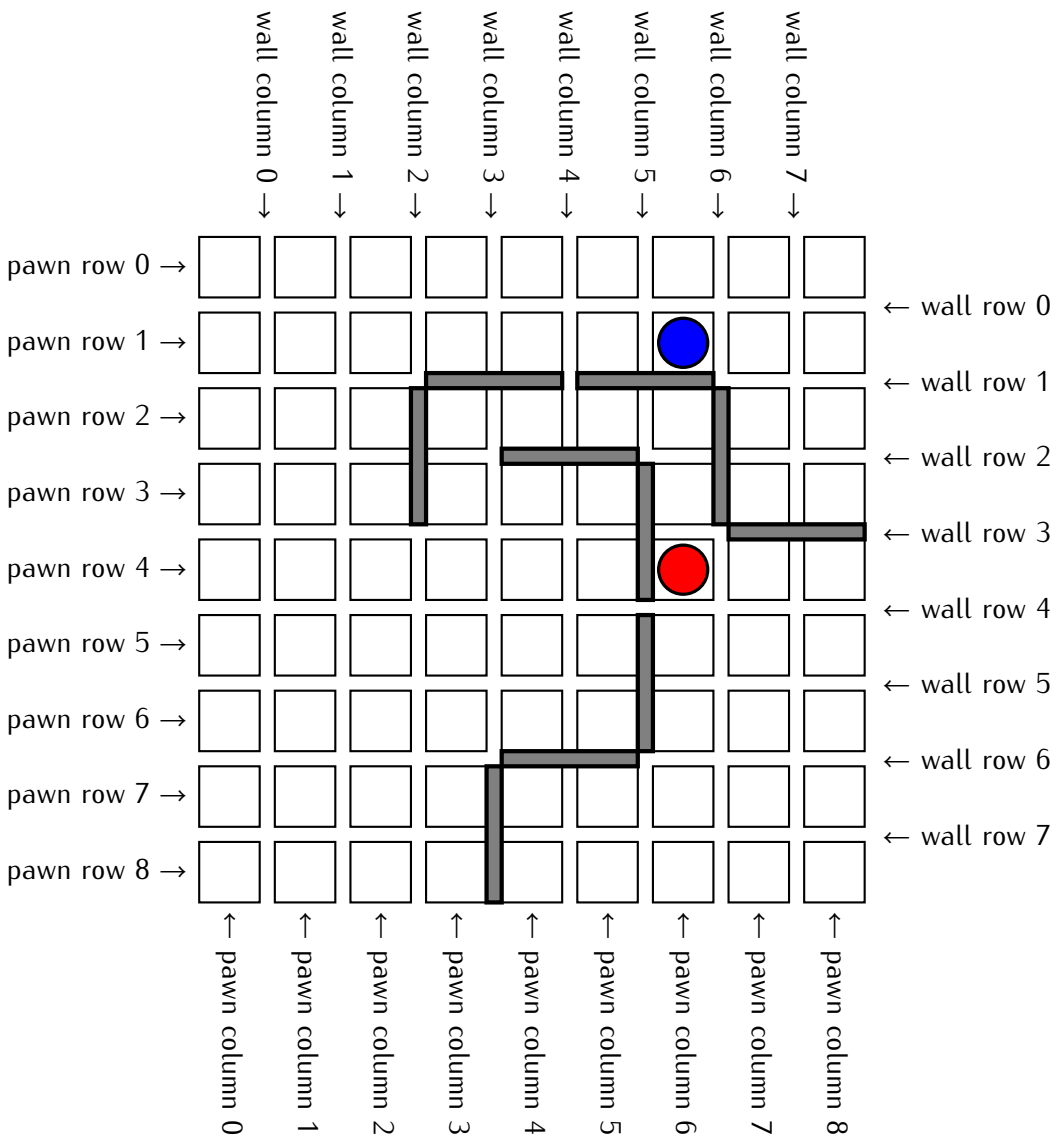


Figure 4: Example of the state of a board after 21 steps have been played

3 Playing with the framework

The way the framework works is the following. The file `game.py` contains the implementation of the rules of the game and the graphical interface. It has to be launched with the URI's of the two players, or the keyword 'human' for you to play directly on the computer. Each non-human player is a server which communicates with `game.py` for the moves.

You can play Quoridor against another human on the same machine using the following command line:

```
python3 game.py human human
```

As each AI agent is a small server to which `game.py` connects, it allows you to try your agent against the agents of another group without giving away your precious source code! To launch the random agent, you can enter in a terminal:

```
python3 random_agent.py -p 8000
```

To play against the random agent you just launched, you can enter in a different terminal:

```
python3 game.py human http://localhost:8000
```

To use the graphical user interface, you will need the Tk bindings for Python. Those are installed in the INGI infrastructure. On Debian-based systems, you just have to install the `python3-tk` package. To install the tkinter module on other OS's, refer to the tutorial provided at `http://www.tkdocs.com/tutorial/install.html`.
To see the different options available to launch a Quoridor game, you can use the `-h` option as follows:

```
python3 game.py -h
```

The same holds for the random agent.

Licensing

The provided classes are licensed under the General Public License[1] version 2.

We will ask you to also license your code under the GPL version 2.

[1] `http://www.gnu.org/licenses/old-licenses/gpl-2.0.html`