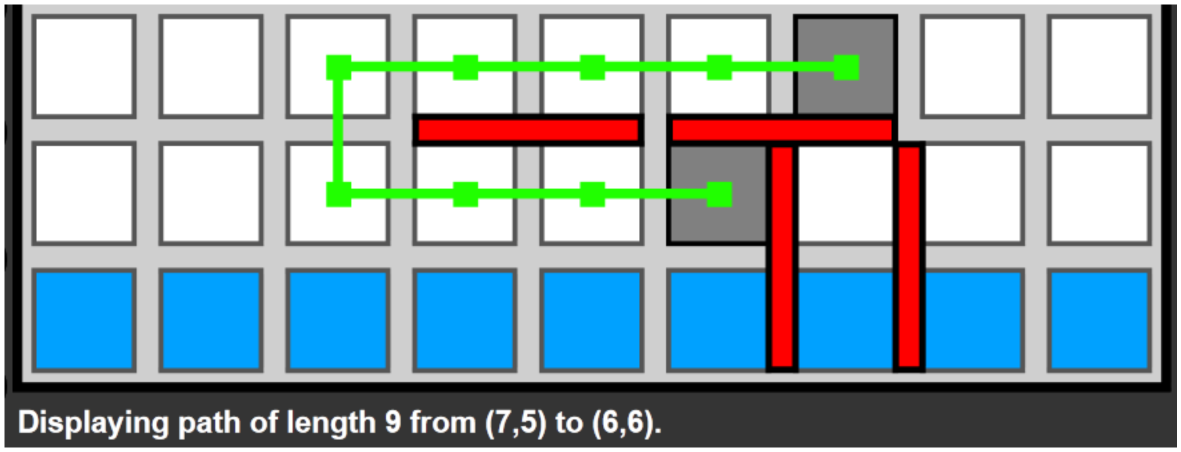
Data Structures for Problem Solving
Quoridor - Project Part 1

3/4/2013

1 Problem



Displaying path of length 9 from (7,5) to (6,6).

Figure 1: Shortest Path

Part 1 of the Quoridor project is about finding a path between two positions on the board, or determining that no such path exists. This will be useful later in the project, both for those who implement a "student player" as well as those who implement a "student engine."

This document assumes that you have completed the pre-lab assignment.

The `config.txt` file might contain some `PRE_MOVE`s that specify player moves that happen before the game begins. This feature is useful for debugging your code. For example, `PlayerMove(1,False,0,1,2,1)` means that player 1 is placing a wall (the second parameter set to `False` indicates a wall placement) from position $(0, 1)$ to position $(2, 1)$. For the wall coordinates, we guarantee that walls are specified from left (start) to right (end) for horizontal walls, or top (start) to bottom (end) for vertical walls.

For more details, as well as the coordinate descriptions, see the `Model/interface.py` printout supplied by your instructor.

Note: There are no players in Part 1 – in this case the engine expects the first parameter of each `PRE_MOVE` to be 1.

1.1 Problem-Solving Session Deliverables (20%)

Students will be organized into teams of 4 or 5 students for this assignment. Each team will work together to deliver the following items:

Assume for this session that we are dealing with a smaller board that is 3 rows by 4 columns, and the `config.txt` file contains the following `PRE_MOVE`s:

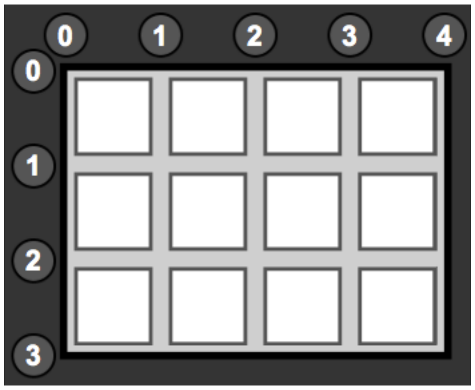


Figure 2: Coordinate System - 3 Rows by 4 Columns

```
PRE_MOVE PlayerMove(1,False,0,1,2,1)
PRE_MOVE PlayerMove(1,False,1,2,1,4)
PRE_MOVE PlayerMove(1,False,1,3,3,3)
```

(a) Draw this board and the walls on graph paper. The squares should be labeled with their corresponding (r, c) value.

(b) Design a data representation for the current board. You need to store the position of the walls, and be able to use the stored positions to determine if two neighboring board locations are separated by a wall.
Answer these questions:
- Do you represent the walls? If so how? For example, how do you store the wall `0 1 2 1`? Be specific: state whether you use a list, a 2D array, or something else (what?).
- Do you represent the squares on the board? If so, how?
- Using your representation, how do you find the neighbors of a particular square? For example, who are the neighbors of square $(1, 2)$?
- Show a graphical representation of the final contents of your data structures.
Note: Remember when it is time to implement this it will be done in the `init` function using whatever data structures you create.

(c) Outline the algorithm for the `last_move` function. Trace your function using your board representation and the object `PlayerMove(1,False,0,1,2,1)`.

(d) Outline the algorithm for the `get_neighbors` function that for a given (r, c) square returns the list of its reachable neighboring squares. Trace your function using your board representation and the square $(2, 2)$. What should the result be?

(e) (Optional) Develop a pseudocode for the `get_shortest_path` function that finds a shortest path between squares (r_1, c_1) and (r_2, c_2). If such a path exists, the function returns a list containing the coordinates of the squares that form the path; if not, it returns the empty list []. Trace your function using your board representation, the source square $(2, 0)$, and the destination square $(0, 2)$.

(f) (Optional) Make a list of 3 test cases for your `get_neighbors` and `get_shortest_path` functions using the same board. Make sure the tests test a variety of scenarios.

1.2 Implementation Deliverables (80%)

1.2.1 Cheat Checking

You are required to work on your implementation *individually*. Do not give your code to anyone else. The department will be doing an exhaustive cheat check of all students, across all sections. If you are caught cheating you will receive a 0 for this assignment, and potentially face harsher disciplinary action.

1.2.2 Installation

These instructions assume that you have already set up your machine for Eclipse and PyDev from the system setup instructions at:

`http://www.`[redacted]`/SystemSetupInstructions2013.html`

Additionally, you should have downloaded and set up the project per the instructions in the *Quoridor Client Student Download* post from the project website:

`https://`[redacted]`royale/docs/`

1.2.3 Student Player

If you are a student player, in Eclipse, you should navigate to the folder `StudentPlayers`, right click on the `RenameYourPlayer` folder, select `Copy`, right click on the `StudentPlayers` folder and select `Paste`. Your should name the folder the same as your RIT username, e.g. `abc1234`.

The main module in your new folder is called `__init__.py`.

In order to work on part 1, the config file, `config.txt`, should have only your username after `PLAYER_MODULES`, and `PART_ONE` should be set to `True`.

For this assignment, you will be implementing the following functions in `__init__.py`:

- `init`: initialize and return your structures
- `last_move`: get information about pre-moves
- `get_neighbors`: get/return all valid neighbors for a square
- `get_shortest_path`: get/return the shortest path between two squares

In addition, you should make use of the `PlayerData` class in `playerData.py` for storing any permanent data.

Also, you are free to take and use any of the code that is developed in lecture (e.g. `myNode.py`, `myStack.py`, `myQueue.py`, etc...). You must store these files in your username directory. In order to use these files from your username directory, you must do a special import statement, for example in your `__init__.py`:

```
from .myQueue import *
from .myStack import *
```

1.3 Grading

Your grade for this part of the project is *non-recoverable*. It is primarily based on the functionality score you receive for your implementations of the `get_neighbors` and `get_shortest_path` functions.

By default, in the `config.txt` file, the auto-grading parameter for part 1, `PART_ONE_AUTOGRADE`, is set to `True`. When you run in part 1 mode, the engine will call your `get_neighbors` function for every square on the board. It will report the result of the tests with a console message:

```
### valid neighbors out of ###. ### excessive neighbors. Neighbor Score: ###%
```

If you get a neighbor score of 100%, you have passed all neighbor tests successfully for that board configuration.

The shortest path tests works similarly. Be aware that it can take a noticeable amount of time to complete these tests (e.g. 5 seconds or more). The engine will generate all possible combinations of two squares on the board and call your `get_shortest_path` function for each pair of squares. It will report the result of all the tests with a console message:

```
#### out of #### paths are correct. Path Score: ###%
```

If you get a path score of 100%, you have passed all shortest path tests successfully for that board configuration.

You should test your code by setting up the `config.txt` file with different wall configurations, using the `PRE_MOVES`, to make absolutely sure your functionality is always correct. If you want to add more than 20 walls, change the config parameter dictionary value for 1 player in `NUM_WALLS`.

1.3.1 Grade Breakdown

- Getting Neighbors: 30%
- Getting Shortest Paths: 40%
- Code Style: 5%
- Submitted Correct File: 5%

1.4 Submission

Part of your grade for this project is making sure you submit your project correctly. It is extremely important that you follow these instructions *exactly*.

1. In a file explorer window, navigate to your workspace directory. This is the directory that Eclipse asks you for upon startup. By default it is usually in your `Documents` directory in a folder called `workspace`.
2. Once in your workspace folder, navigate to your quoridor project folder.
3. Navigate to the `StudentPlayers` folder.
4. Highlight your username folder, and zip it up. On Windows, you can select `Send To ...Desktop ...Compressed File`. On OSX, you can select `Compress`. This will create a zip file, which you should name/rename `username.zip`, where `username` is your username.

5. Go to MyCourses and upload/submit the zip file to the Project Part One dropbox.

Your submission is due by the deadline to the MyCourses dropbox. There is no late dropbox for project submissions. Make sure that you submit a `zip` file and not anything else in any other format (e.g. `rar`, `7z`, etc.). Do not submit your individual python files! All files must be compressed into the zip file, whose top level is a single folder with your username.