AI Programming Project:
Using Minimax with Alpha-Beta pruning to play Quarto

Purpose:

- Learn the basic elements of adversarial search in AI.
- Gain hands-on experience with one of AI's most popular adversarial search methods: Minimax with alpha-beta pruning.

1 The Assignment

You will implement Minimax with Alpha-Beta pruning from scratch in the language of your choice. It is a very simple algorithm that is well-described in most AI textbooks. The lecture notes for this class may also offer some assistance.

2 Quarto

Figure 1 shows a basic 4 x 4 Quarto board with 16 pieces. Each piece has 4 properties: size (large or small), shape (circle or square), color (red or blue), and hole (yes or no), where the hole is denoted by a yellow circle in the piece's center. No pieces are the sole possession of a particular player: all are shared.

Players alternate placing pieces on the board. The goal is to create a full row, column or (main) diagonal consisting of 4 pieces that share at least one of the 4 properties. The player who puts the final piece into such a row, column or diagonal is the winner. Figure 2 shows several different winning *lines*.

Quarto differs from most board games via an interesting twist: you never choose your own piece; the opponent does.

A quarto game begins with Player 1 choosing any piece (not already on the board). That piece is given to Player 2, who must place it on the board. Afterwards, Player 2 chooses the next piece for Player 1, who then places it on the board and selects the next piece for Player 2. This continues until either a) all pieces are on the board but no lines share a common property, or b) a 4-piece line with one or more common properties is formed.

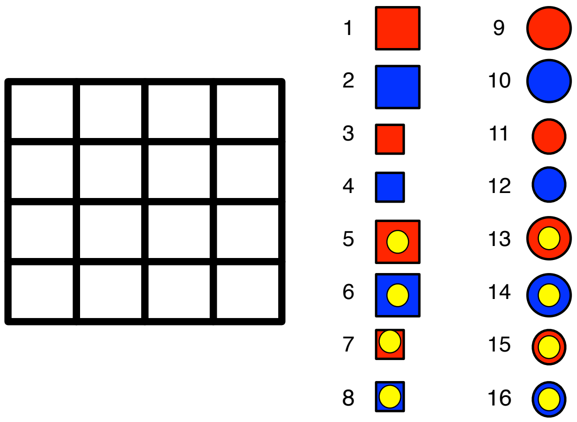

Figure 1: The standard 4x4 Quarto board along with 2-dimensional facsimiles of the standard 16 pieces. The numbering of pieces is arbitrary and plays no role in the game. All 16 pieces are shared by the two players.

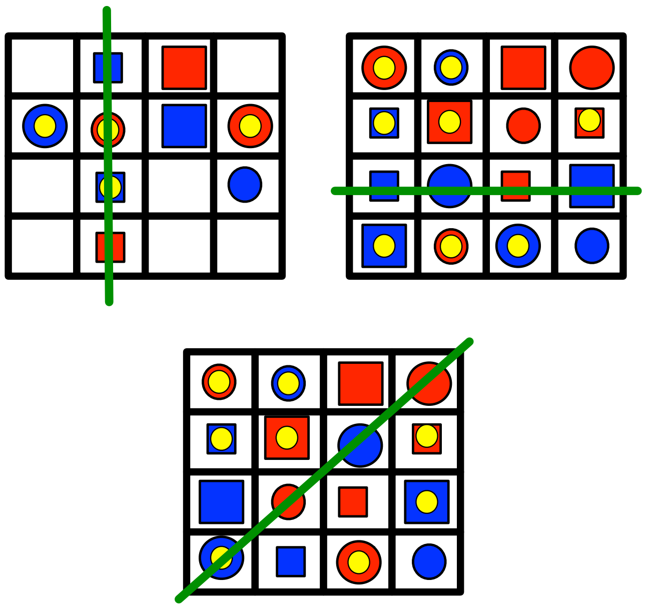

Figure 2: Three different winning configurations in Quarto. (Top Left) All pieces in the second column are small. (Top Right) All pieces in the 2nd row (numbered from the bottom) are smooth (i.e., have no hole). (Bottom) All pieces along the diagonal are circles.

A player's turn thus consists of two actions: 1) Placement of the received piece on the board, and 2) selection of a piece to give the opponent. Neither player possesses pieces once they have been placed on the board. Thus, either player can take advantage of formations already on the board.

Success in this game requires the ability to look ahead several moves and to find situations that essentially **force** the opponent to give you a piece that can complete a winning line. Figure 3 shows one such scenario. By giving piece 6 to player 1, player 2 sets himself up to receive a winning piece on the next turn.

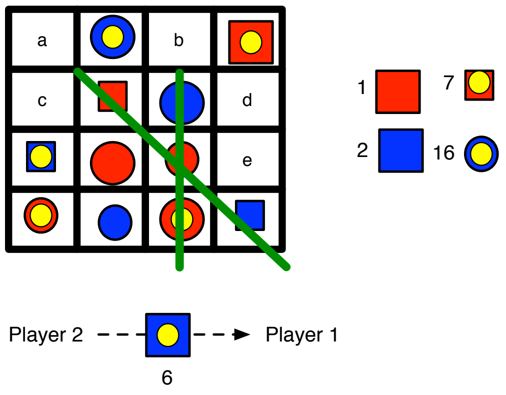

Figure 3: A forced win in Quarto. By giving Player 1 piece 6, Player 2 insures that wherever Player 1 places piece 6 and whatever piece is given to Player 2, the game will be won by Player 2.

To play an online version of Quarto, check this site: http://quarto.freehostia.com/en/

2.1 Levels of Play

For this project, three levels of automated play are defined: random, novice, and minimax-D.

A random agent always chooses randomly among both the locations to place its current piece and among the pieces to give the opponent, while a novice plays as follows:

- It always places its piece in a winning formation, if possible. Otherwise, it chooses randomly among the existing open cells.
- If given a choice among pieces to give the opponent, it always chooses one that cannot be used to immediately win the game, if such a piece is available.

Finally, the minimax-D agent uses Minimax with alpha-beta pruning to a maximum depth of D in order to choose moves.

3 Minimax for Quarto

After playing a few rounds of Quarto, you may realize that, even though the board is just 4 x 4, there are quite many possibilities that require consideration on each turn. This becomes particularly obvious near the end of the game, when 5-6 un-played pieces remain. You may notice that if you could just look 2 or 3 moves into the future, you could make strategic choices and guarantee a win - if only you could keep track of all those future possibilities in your head! Of course, the computer can keep track of all those possibilities, and with just a few ply (i.e. levels of) lookahead, can easily play *perfect* Quarto, at least toward the end of the game.

Quarto has a large search-space, but one that shrinks dramatically as the game proceeds. For example, on the first move, player 1 has 16 options, one for each piece to give to player 2. On the second move, player 2 has 16 x 15 = 240 options: 16 possible positions to place his piece and then 15 possible pieces to give to player 1. With this early-game branching factor of over 100, search to a depth of 4 or 5 ply becomes computationally demanding. However, later in the game, deeper search becomes feasible.

This large branching factor can be a problem for many computers, so it is okay to allow a minimax agent to make its first 2 - 4 moves using a random or novice strategy. That is, you may need to wait until 4-8 pieces have been placed on the board before using minimax to compute moves.

Because all pieces are shared in a Quarto game, the evaluation of intermediate (i.e., non-winning) states can be difficult. It is easy to assess whether a state (S) is *potentially near a winning state* based on the number of 3-piece lines with a common property (e.g., the 3 smooth pieces along the diagonal of Figure 3) that have an open spot (e.g. cell a in the same figure) along that line. But it is hard to estimate which player can win from S without doing further lookahead. And in Minimax search, only the bottom nodes of the tree are evaluated, so further lookahead is not normally an option.

In addition, the presence of 3-piece lines will not always indicate a near win, since:

- If player A cannot fill a line but is worried that player B might do so, A can simply fill the line with a non-winning piece.
- Player A chooses the piece for player B, so he controls B's ability to fill the line.

In most cases, there need to be many open 3-piece lines for either player to be able to guarantee a win.

All of this implies that Minimax may be of little value early in a Quarto game, since the bottom states in the Minimax tree will not be final states and thus not be amenable to accurate evaluation. However, as the game proceeds past the midway point, the lookahead of Minimax should prove quite valuable, allowing the computer to consistently beat a human player (with less than perfect mental lookahead).

4 The Interface

Your system must allow the user to enter the agent-type for two players. The possible agent types are:

1. Human - the computer will receive commands regarding board locations and quarto pieces from a real person.
2. Random - choose locations and pieces completely at random.
3. Novice - as described above.
4. Minimax D - where D is the maximum depth that minimax with alpha-beta pruning will search.

The selection of player types should be done through the system interface, NOT via changes to the source code, recompilation, etc. The user should be able to select any combination of agents, such as a novice versus a human or a minimax-3 versus a minimax-4. Your system should then perform the proper simulations, and if a human is involved, she should be querried in an understandable fashion. Anyone who understands quarto should be able to play a game on your system without knowing anything about your code.

You need not write fancy graphics for displaying a quarto board. Simplifications such as the following are fine.

Each quarto piece consists of 4 properties: 1) red or black, 2) big or small, 3) bracketed or unbracketed, 4) starred or unstarred. Hence, the 16 pieces are:

r, b, R, B, (r), (b), (R), (B), r*, b*, R*, B*, (r*), (b*), (R*), (B*),

Here, r denotes a little red block, while R represents a big red block, with similar semantics for large and small black blocks, B and b, respectively. Using these symbols, a typical board configuration can be displayed on a command line or in a simple text window as follows:

```
  r                    (R*)
        b*      B
(r*)   (B*)
        (r)            (b*)
```

These types of pictures take a little getting used to, but they are a fine substitute for fancy graphics in this assignment.

5 Quarto Tournaments

25 % of this assignment involves playing your minimax agent against at least 2 other agents (written by at least 2 other groups). To get these points, your group will need to organize a quarto tournament with these other groups. The details of the tournament are up to you, and they will not have a profound effect upon your grade. The main thing is that each of the agents has the opportunity to play MANY (i.e. hundreds or thousands) of games against EACH of the other agents. The results of these games must be well documented in terms of the number of wins, losses and ties for each agent.

A demo version of the tournament must also be made available to the course instructor such that, on demo day, a quick tournament involving just 1 or 2 games for each pair of agents can be completed in the course of one or a few minutes.

6 Deliverables

1. (**30 points**) A working demo of your system in which the course instructor can choose any two types of agents and then watch the game being played on the screen. If one of the chosen agents is a human, then the instructor should have no trouble understanding the interface and entering his or her moves.
2. (**15 points**) . A clear description (of 2-3 pages), including helpful illustrations, of your state-evaluation function. This is the core of your report. Write it well!

3. (**10 points**) A table showing the results of 100 runs of your system in which a novice plays against a random player.
4. (**10 points**) A table showing the results of 20 runs of your system in which a novice plays against a minimax-3 agent.
5. (**10 points**) A table showing the results of 20 runs of your system in which a minimax-3 plays against a minimax-4 agent.
6. (**15 points**) A demo of a tournament in which your minimax agent participates against the agents of at least 2 other groups.
7. (**5 points**) A table illustrating the performance of your minimax agent within the tournament.
8. (**5 points**) A brief description (approximately 1 page in length) of your experience programming for and participating in the tournament.

For this project, demos will be given with the other members of your tournament group (should you choose to do that part of the assignment). However, one report (covering points 2-5,7-8 above) must be delivered **per group** (of 1-2 people). You are free to work closely with members of other groups, but each group must code up its own agents and design its own state-evaluation function.