Programming Assignment 1: Minimax Search for Konane

CS 436 – Artificial Intelligence – Fall 2008
Due: October 20, 2008

Abstract

For this programming assignment, the goal is to implement the game of Konane and then create an AI player using the MiniMax search algorithm, both with and without $\alpha\beta$-pruning. The writeup will be in the form of short answers to the questions listed in Section 3.

1 The game of Konane

Konane, also known as Hawaiian Checkers, is a two player game played on a square grid. The grid can be of variable size; for this assignment, we will restrict the board size to even side lengths in the range from 4×4 to 8×8. The game begins with the board filled with pieces; a black piece is placed in the square in the upper left corner of the board, and the rest of the board is then filled by alternating between black pieces and white pieces (see Figure 1(a)).

Black always goes first. The game begins with the black player removing one black piece, either from one of the four corners, or from the four squares at the center of the board. For example, on the 6×6 board shown in Figure 1(a), the black player must take a piece from one of the locations $\{(0, 0), (2, 2), (3, 3), (5, 5)\}$. The white player then removes any single white piece adjacent to the empty space left by Black's move.

After this point, game play proceeds by alternating turns. On each turn, the active player selects one of his or her pieces, and moves that piece by jumping over one or more of the opponent's pieces. Each jump must move over a square containing an opponent's piece, and land in an unoccupied square. Jumps must be either vertical or horizontal (no diagonal moves are allowed). Multiple jumps may be made in one turn, provided that they are made consecutively by the same piece, that all the moves are made by the same piece in the same direction (eg. on the board in Figure 1(b), for the piece at (0, 2), two jumps to the right would be legal, but one jump to the right and one jump up would not), and that each move would be a legal move taken by itself (eg. each jump moves a piece exactly two squares and ends in an empty square). Pieces that have been jumped over are removed from the board.

Game play continues until a player is unable to make any legal move, at which time the game is over, and that player is considered the looser.

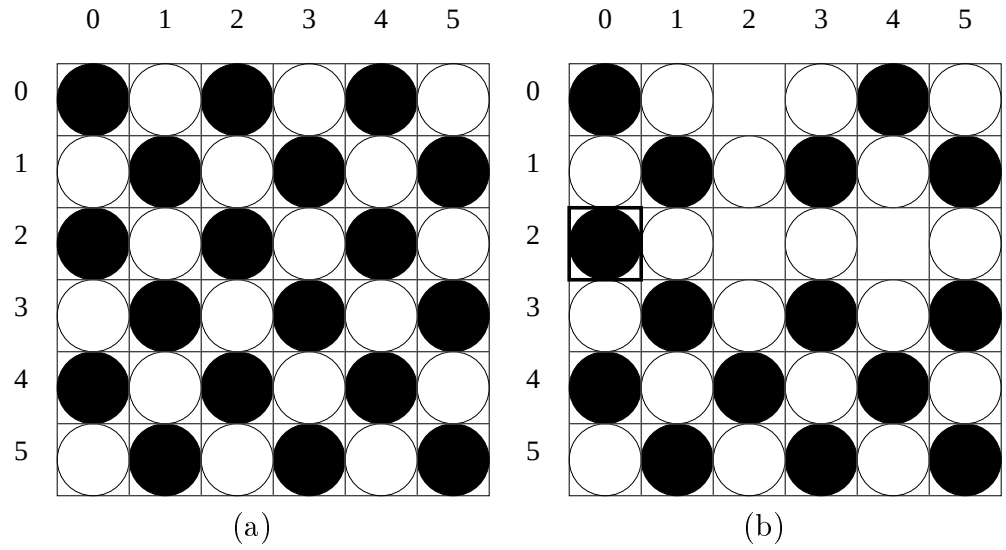

Figure 1: **(a)** A 6×6 Konane board before play has begun. **(b)** A board which could not be generated in actual play, as no white pieces have been removed; it serves to demonstrate legal jumping. The piece at (0, 2) can make either one jump to the right, ending at (2, 2), or it can make 2 jumps to the right, ending at (4, 2); these are the only legal moves for that particular piece.

2 Program Requirements

2.1 Submission Requirements

You are free to use any computer and any programming language for this project; however, we advise you to avoid languages and systems such as SQL, Matlab, etc. Your program should be submitted as an archive, either a zip file or a tarball, with the filename consisting of your first and last name and the assignment name, delimited by underscores, all lower case. For example, '`john_sheppard_pa1.tar.gz`'. You should email this archive to `john.sheppard@cs.montana.edu`.

Your archive should contain any and all files required to build and run your program in case the instructor wishes to run the program. It should also contain a README file, in which you list the files in your archive and a brief description of what each file contains. It should also contain any relevant notes on the functioning of your program, such as known bugs, parameters that result in excessive runtime or poor behavior, etc. You will still lose points if your program does not perform correctly, but if you document the failure modes, you can get partial credit. Your archive should also contain a `./data/` subdirectory which should contain output from sample runs, timing information, and other information as requested.

2.2 Feature Requirements

Your program must correctly implement the rules of Konane, allowing all legal and only legal moves to be made, and correctly terminating and listing a winner when the game is over. Your game must allow for each player to be either a human, a MiniMax agent without $\alpha\beta$-

pruning, or a Minimax agent with $\alpha\beta$-pruning. The user must be able to select one of these values for each player at runtime. For AI players, the user must be able to select maximum search depths; you should describe behavior for different depth values in your README, and suggest reasonable values to use for 4×4, 6×6, and 8×8 boards. Your AI players should be able to take the role of either Black or White. The user must also be able to select the board size.

Human players should be prompted for a move on their turn; if they provide an invalid move, they should be prompted again until they give a valid move. AI players should never attempt to make an invalid move. The current state of the board should be displayed after each move, along with which player's turn it is. For AI players, the maximum search depth reached, number of nodes evaluated, and the time taken to decide on a move should be displayed each time a move is selected. The display of the board can be simple ASCII text; for example:

```
-------
|b|w|b|
|w|b|w|
| | |b|
-------
```

You should implement the MiniMax search algorithm as described in class or in Chapter 6 of Russell & Norvig. You should implement two Konane playing agents, one using the simple MiniMax algorithm, and one using MiniMax with $\alpha\beta$-pruning. Both agents should use the same board evaluation function so that their performance can be compared directly to one another (see R&N p. 171 for a discussion of evaluation functions). Try to implement the best board evaluation function you can; you may want to play some games against a friend once you've implemented the basic game with human players. While you are not allowed to work with anyone on your program, or show anyone your code, you are allowed to play the game with others, regardless of whether they are taking the class.

3 Writeup Requirements

Your writeup for this assignment will be relatively short, consisting of a series of answers to the questions given below. Your answer to each question should be concise, no more than a few paragraphs. You are, however, required to answer in complete, grammatically correct English sentences except when explicitly instructed to use a different form (eg. "create a table..."). This is a writing assignment, and your writing is a part of what is being evaluated.

You should submit your writing assignment in class, and you should also email an electronic version to `john.sheppard@cs.montana.edu`. Your submission should be in the PDF file format; please do not submit MS Word files, flat text files, or scans of hand-written work. Please see the instructor if you have difficulty creating PDFs.

Students are encouraged to generate their writeups using LaTeX, as templates for later writeups will be given in that format. If you are unfamiliar with LaTeX, there are a great many resources available online for learning it. Note that use of LaTeX is not required for this or future assignments, only encouraged.

3.1 Questions

Question 1. Specify a hypothesis about expected performance from the algorithms and from the heuristic function you defined. Remember that this hypothesis needs to be *testable* since your experiments, ultimately, should be designed to test the hypothesis.

Question 2. Play several games with the MiniMax player vs. the $\alpha\beta$ player. Use board size 4×4, 6×6, and 8×8. For each size, allow each player to be black once. Use the same maximum search depth for both players. Make a table of the average number of nodes explored for each agent at each board size; this table should be your response to Question 2. A partial example (with nonsensical data) is given:

Max. Depth	4×4 Minimax	4×4 $\alpha\beta$	6×6 Minimax	...
1	23	48	234	...
2	75	21	22	...
...	...	...	...	...

Question 3. Create a set of tables, this time showing the average number of states explored for each algorithm at each turn (averaged over 2 games, so each algorithm can be Black once). There should be three tables, one table for each board size; for each board size, select a single depth bound based on your earlier results. A partial example (with nonsensical data) is given:

4×4 board, max depth 7:

Turn	Minimax	$\alpha\beta$
1	5	7
2	15	2
3	71	32
...	...	...

Question 4. Discuss the data presented in your tables; what do the data tell you about the algorithms? How well do they scale with board size? How significant is the difference between the runtimes of the algorithms near the beginning of the game? In the middle? Near the end?

Question 5. Describe your board evaluation heuristic and how it works (in complete English sentences, not pseudocode).

Question 6. Describe how you came up with your evaluation function. This should include, but not be limited to, what features of the game state you considered using, why you chose the features you did use, how you balanced speed of the evaluation algorithm with performance, and generally why you settled on the evaluation function you did. This response should be a bit longer than the others; it should be at least several paragraphs. Be concise, but use the space you need to answer the question.

Question 7. Discuss any conclusion you can draw from the data in your tables and show and why we should expect the behavior you observed. Be sure to discuss relative performance, not only with respect to nodes evaluated but with respect to win-loss ratio. Was your hypothesis verified or not? Why or why not?