The Dala Game

http://en.wikipedia.org/wiki/Dala_(game)

Searching Method

Alpha-beta Search

Our client uses standard minimax algorithm with alpha-beta pruning. However, some slight modifications have been made:

1. When a player has a capture, **the move to take away a piece is treated as a new node**, and therefore there may be two consecutive max or min nodes in the search tree. However, when computing the depth for cutoff test, the capturing node is not included.
2. The first level of the search is treated differently from the rest. **In the first level, greedy approaches are used to select from different succeeding nodes with the same value.** This includes the following:
    a. Moves that cause a capture are favored over non-capturing moves.
    b. Moves with higher local evaluated values are favored over those with lower values.
    c. In the Movement Phase, a move that decreases the total distances between the player's pieces is favored.

The reason why the capturing nodes are not included when computing depth is because that we want to make sure the evaluating function is applied to the states with the same progress in the game, or our evaluating function may have incomparable results due to different weighting in different stages.

The "total distance heuristic" is used in the Movement Phase to solve the problem that the next state that has a different evaluated value may be too deep to search in the final stage. For example, consider the state on the right; both parties must make many moves to achieve a capture. Such cases may lead to *infinite repeated moves that never lead to a win or lose*.

1					2
2					
1	2				1

The search algorithm generates all possible moves in a given state and continues to search each move until the cutoff condition is met. Minmax rule is used to maximize the estimated utility value under the assumption that both players move optimally according to our utility evaluation.

The Cutoff Test

Our cutoff test limits the search depth to about 4 in the Dropping Phase, and slightly relaxes the limit in the Movement Phase according to the number of pieces left. This is due to the nature that the branching factor is much smaller in the Movement Phase if the number of onboard pieces is small.

Our search won't stop at the capturing move because of the potentially dramatic change of the evaluating values. If the terminating condition is met (that any one player has fewer than three pieces left), the search is also cut off.

Finally, additional time limit is set so that our client won't run out of time. When 20 seconds have passed, the depth is limited to 2, and when 30 seconds have passed, all nodes are cut off.

The cutoff depth can be controlled by a parameter, which is separated when our client is the first player or second player.

The Evaluating Function

Our client uses five factors to evaluate a given situation:

1. The total number of captured pieces from the enemy.
2. The total number of positions to drop where a capture can be made.
3. The total lines on the board that a capture can be made. (Note that previous factor counts the number of positions, but this one counts the lines so that multiple positions on the same line are counted as one.)
4. The total number of positions to move in where a capture can be made.
5. The total number of positions to move out where a capture can be made.

Due to the differences between phases of this game, the second and the third factor are *ignored* in the Movement Phase and the importance of the third and the fourth factor are lowered in the Dropping Phase.

Our client uses different strategies according to his position as the first or the second player. It has been observed during my experiments that the Movement Phase is more important for the second player.

Let the values of the factors be V_0~V_5, our evaluation formula for a player is:

$V_p = V_1 \times V\_LOST + (V_2 \times V\_TWO + V_3 \times V\_LINE) \times S + (V_4 \times V\_MOVE + V_5 \times V\_MOVEO)/T$

S = 1 in Dropping Phase and 0 in Movement Phase, T = V_REDU when the player has more than 1 piece to drop. (We choose to start focusing on the Movement Phase when we only have one piece to drop.) Otherwise, T = 1.

The final evaluation value is the difference calculated by subtracting the V_e of the enemy from the V_p of the player. The value of termination nodes are calculated independently using the number of remaining pieces of players. This also includes the situation where a player has no moves to make.

Implementation Details

State Representation

The states in our program have four parts: (1) the board configuration, (2) the number of pieces that remain to be dropped for each player, (3) the number of pieces that have been captured for both players and (4) the depth in the search tree.
Since the second and the third information is unavailable from the input, our **action()** function must track each input and output to calculate the correct state. The calculated result is then fed to the real search function. The depth of the state is used solely in the cutoff test.

Evaluation Value Calculation

The calculation of positions to drop or move to make a capture is done by scanning 6x6 positions and checking if a drop or move is valid and whether a capture would result. The distance value used in the first level is the sum of all Manhattan distances between all pieces. Finally, the local evaluating value used in greedy approach in the first level is calculated by directly applying the evaluating function to the child states.

Searching for Optimal Parameters

The optimal values of **V_LOST**, **V_TWO**, **V_LINE**, **V_MOVEO**, **V_REDU** and cutoff depth for both the first and the second players are searched using a generating program. The program automatically generates new values and starts the java server to check the results of the generated client. Two stages are used to generate clients and then select best clients.

Our program is designed so that it can read the parameter file to determine its behavior. The generation script can simply overwrite the parameter file to test the result. All tests are done under Ubuntu Linux 11.10. The **gen.py** and **contest.py** scripts may not run properly under the Windows environment.

Generating Stage

In the generating stage, we use **gen.py** to generate parameters for both the first player and the second player. The new parameters are generated using local search with random restart. In each iteration, one parameter is selected to be mutated from the currently best player. The parameter would be changed slightly, with a small possibility to change greatly. After that, all parameters are subject to mutate with a very small possibility.

The newly generated client must beat the previously recorded best player to be recorded as the current best player. If no better client is produced in many runs, a randomly constructed client would be produced by randomly generating all parameters.

The cutoff depth is also searched because of the strange phenomena observed during my experiments that a client with shallower search depth may sometimes beat the one with deeper depth. This may be caused by inaccurate evaluating function.

Contest Stage

In the contest stage, all clients previously recorded in the generating stage participate in a contest with one another. However, the time it takes to arrange contests between all clients is too long to be practical. So a simpler approach is taken:

contest.py is used to schedule the contests. Every time we select 10 clients to compete with one another. The best clients for both the first and the second players are selected. After several rounds, the number of remaining clients can be reduced by a factor of 5.

After each round, the log produced is examined to find the best parameters in each contest. And the selected clients would then again join contests with one another. The contests continue until the total number of clients is small, we then select the best client manually. The best parameters for the first and the second players are combined to form the final client.

Result

The final selected parameters are:

Player	V_LOST	V_TWO	V_LINE	V_MOVE	V_MOVEO	V_REDU	*cutoff depth*
1_{st} player	876294	251283	751278	720102	846047	1	4
2_{nd} player	911133	75813	68489	159224	395136	7	3