Homework 7: Othello

DUE: on or before May 2nd. A tournament will be held in class on the 28th for programs turned in early.

Your goal for this assignment is to implement an Othello playing program that can beat you at Othello. Your program should perform heuristic minimax.

The testing program will supply the MAXDEPTH argument. Turn in a zip file consisting of a single folder using the naming convention below (important). The folder should contain the source and compiled versions of your code, include an executable file named "play" as described below, and a project report. The report must be in pdf (preferred) or Microsoft Word format (.doc or .docx). The report should be at least 3 pages long and include the following information:

- Names of the team and team members
- An English description of your program's data structures
- An English description of your program's static evaluation function
- Justification of your design decisions, including your choice of programming language
- Any special features of your program
- How you tested your program
- Citations to any resources you used in developing your program, such as articles about Othello strategy.

Modify your program so it uses the milliseconds per move parameter (TIMELIMIT1) to determine how much search to do before returning an answer. You will need to devise a cut-off strategy that relies on time. Your player will be tested with the following time limits:

1 second

4 seconds

16 seconds

60 seconds (1 minutes)

240 seconds (4 minutes)

The Linux operating system call clock( ) can be used to check the time using a clock that advances according to the amount of time the process has been executing (as opposed to the actual time, which can be greater). For example, in C++, the following code snippet determines the amount of time that has passed since the opponent's last move was read:

```
#include <time.h>
clock_t start, now;
...
    // read opponents move
    scanf("%i %i", &opponentX, &opponentY);
    start = clock();
    ...
    now = clock();
    // units for clock() are CLOCKS_PER_SEC, units for timelimit1 are 1/1000 of a
```

```
second
    time_passed = (now - start) / (CLOCKS_PER_SEC / 1000)
...
```
Note that:

- •CLOCKS_PER_SECOND is on the order on 1,000,000. When doing arithmetic on clock tick units, you have to be careful to avoid arithmetic overflow.
- •Simply checking whether you are out of time in your CutoffTest function and returning from that function if you are is not by itself adequate, because that would only return from one recursive invocation of MiniMax. When you are out of time, you must print your answer and be ready to read the opponent's next move.
- •A simple way to solve this problem is to run experiments to determine the amount of time that corresponds to a depth-n search, for various values of n. Then, just write your program so that it converts the time limit to a depth limit.
- •Another approach would be to implement an iterative-deepening version of minimax search, and when the time limit is reached, return the answer calculated for the highest completed iteration.

Grading:

- •50% correctness (10% of the grade will be the timelimit portion of the code so if you dont feel you can get it finished in time, don't worry about it.)
- •25% programming style (make sure to comment your code, make it easy to read, etc)
- •25% report

Bonus: up to 10 extra points for performance in the tournament.

Teams

You can either work by yourself or with a partner. Only one partner needs to submit the assignment. Please make sure to clearly specify who you worked with when you submit your code.

Programming Materials

Pseudocode for Heuristic MiniMax with Alpha-Beta Pruning

Specification

Player programs are a folder (directory) containing a main program named "play" and any necessary auxiliary files.The name of the folder is the team name, written using only lowercase letters and using a dash "-" in place of blanks. The team name is of the form

word-word-member-name-member-name

For example: screaming-banjos-tim-kopp-henry-kautz

The file play should be executable. It might invoke other files in the directory. For example, if your player is written in java and consists of the files foo.java and foo.class, then the file play would be the

script:

```
#!/bin/bash
cd "$(dirname "$0")"
java foo
```

When play is executed, it starts by reading a line from stdin. **It must read the entire line including the end-of-line character "\n".** The line is of the form:

```
game COLOR DEPTHLIMIT TIMELIMIT1 TIMELIMIT2
```

COLOR is either the letter B or the letter W, and indicates whether the program makes the first play: B (black) means go first. The remaining arguments are integers. DEPTHLIMIT is the cutoff depth, or 0 if none. TIMELIMIT2 is the amount of CPU time in milliseconds(1/1000 sec) allowed for each move, or 0 if no limit. TIMELIMIT2 is the total time budget for the game in milliseconds, or 0 if no overall limit. If TIMELIMIT1 is not 0, then DEPTHLIMIT is ignored. If TIMELIMIT2 is not 0, then DEPTHLIMIT and TIMELIMIT1 are ignored.

If COLOR is B, then the program must write a line (ending with the newline character) to stdout specifying its move.After printing its move, the program waits to read a line from stdin indicating the opponent's move. If COLOR is W, then the program waits to read a line from stdin specifying the opponents move, and then begins to work on its move.

Moves are specified by line consisting of two integers

```
X Y
```

indicating a position by its horizontal and vertical position. X and Y range from 0 to 7. If a player has no legal move, it should "pass" by printing the string "pass" rather than a pair of numbers. **Lines must be terminated by end of line characters "\n".**

The tournament program determines when the game is over **(all squares are filled or neither play can make a move)** and who the winner is. When a player program believes that the game is over and it cannot move, it should wait by reading from stdin. If a line is able to be read (it should not, unless the program has a bug), it should a line consisting of the word "pass" and loop back to read from stdin.

If a program takes longer than TIMELIMIT to print its move, or if it makes an illegal move, it immediately loses and the tournament program ends the game.

User Interface Program

The interface is invoked as follows:

```
tournament GUI BLACKPLAYER WHITEPLAYER DEPTHLIMIT TIMELIMIT1
TIMELIMIT2
```

GUI is one of the following strings:

gui	Display the game graphically (using X windows and the DISPLAY environmental variable).
text	Print a record of the game to stdout
none	Nothing is printed except the final winner of the game

Each of BLACKPLAYER and WHITEPLAYER is either the flag -human or the name of a player program directory (including the path if not in the working directory). If a player is human the player either uses the gui to enter her moves, or in the case of text display types her moves as pairs of integers to stdin after receiving a prompt. TIMELIMITs are not enforced for human players. For program players, the tournament program runs them and interacts with them via stdin and stdout as described above.

At the end of the game, the tournament program prints a single line to stdout of the form:

winner COLOR ADVANTAGE REASON

where COLOR is one of the letters B,W, or T (for tie), ADVANTAGE is an integer equal to (# winner squares - # loser squares), and REASON is one of the following strings:

legal	The game was played properly by both players
timeout	The losing program exceeded the time limit on a move
illegal	The losing program made an illegal move

Note that in the came of timeout or illegal, ADVANTAGE might be negative.