COMP3130 – Group Project in Computer Science
Warm-up Project – 4×4×4 TicTacToe Agent

Andrew Haigh – u4667010;
Timothy Cosgrove – u4843619;
Joshua Nelson – u4850020

March 24, 2012

1. Abstract

The purpose of this project was to implement an intelligent agent to play 3 dimensional 4 by 4 by 4 Tic Tac Toe. We chose to implement this in C, with a pipe interface to python. This allowed us to use python's 3D libraries for visualisation while utilising the speed of a compiled C program.

2. Solution Overview

The core of the agent is implemented with a combination of minimax and α-β pruning.
Board states are stored simply as a 3 dimensional array of chars; but this data is passed through the program as a struct including extra information such as a heuristic evaluation of the state. Once the minimax program reaches a fixed cut-off depth we select a state based on this heuristic. For testing purposes all code is compiled with the -g flag. This allows us to use the program `gprof` to analyse running time and total number of function calls (this is used to optimise and evaluate the effectiveness of α-β pruning).

Overview of Modules:
`visualisation.py`:

- The python module which takes user input and displays the game board using VPython
- This is the main program, it creates a sub-process (`worker.c`) which returns board states to display
- Contains several visualisation options: press 'g' to switch views and 's' to enable red-cyan stereoscopic 3D.

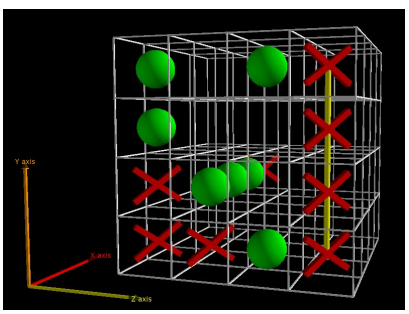

Fig 1: Example of game visualisation

`worker.c`:

- The link module which handles communication between python and C
- Calls code from `state_functions.c` to start minimax and pick the next move

`state_functions.c`:

- Contains all core functionality of the AI agent including minimax functions and state evaluation functions
- Includes a simple victory function which checks if a player has won with a line nearby to the most recent move. (comprehensive victory checks are made in `visualisation.py`)

3. State Space for 4×4×4

In the original scope of this project we had intended to examine the entire state space before making a move. However, the state space is in fact exceedingly large. Considering all possible states we have 3^{64}, but many of these (far more than half) are illegal or unreachable. A mathematical paper by Oren Patashnik[1] points out that after 18 moves (under perfect play by player 1) the game will be over or in a state where every move is forced. With this in mind we can consider a more reasonable upper bound on the states we must examine:
64 choose 18 = 3601688791018080.
If we assume we can examine 1 state per clock cycle (a completely and utterly unreasonable assumption) it would still take 250 hours to search this reduced state space on a 4GHz CPU. Minimax with α-β pruning and other clever tricks could significantly reduce this figure; however it remains obvious that searching to depth 18 for every move is unachievable. Thus we set a depth cut-off where we stop searching and perform a heuristic state evaluation.

Due to this inevitable cut-off it also became impossible to determine if either player has a winning strategy. It is however interesting to note that (in the paper mentioned above) Patashnik determined that the first player does in fact have a winning strategy.

4. Heuristics and Cut-offs

briefly explain heuristics and cut off decisions

5. Pure Minimax vs. α-β pruning

Some tests were run using the Unix tool `gprof` to determine the effectiveness of heuristics and α-β pruning. By counting the number of calls to our state-evaluation function we get the total number of examined leaf nodes. Furthermore `gprof` returns a summary of how much time (in seconds) is spent running each function and the whole program.
The tables below summarise `gprof` results when the agent is making it's first move; run at various cut-off depths:

Total Leaf Nodes Examined:

Depth	Minimax	α-β	α-β + Heuristics
3	16777216	283793	???
4	???	???	???
5	???	???	???

Total Time Taken (Seconds):

Depth	Minimax	α-β	α-β + Heuristics
3	11.7	0.23	???
4	???	???	???
5	???	???	???

The benefits of α-β pruning are obvious and indisputable, often cutting the number of nodes and total time by a factor of over 100!
The benefits of our heuristic function are however less obvious. We notice a that a great deal less leaf nodes are examined; but the improvement in running time is negligible. This is due to the fact that we must constantly evaluate states for their potential value; taking precious CPU time.

6. Summary and Reflection

A Summary. Also discuss how this will prepare us for reversi (language choice, teamwork, gitHub)

7. Bibliography

Pfft what am I; an Arts student? I don't know how to make a bibliography. Man I should get some sleep.