Search and Game Trees
Algorithms for Videogames
COMP09041

Paul Keir

February 3, 2025

Instructor: Paul Keir

Laboratory Objectives

The following tasks encourage you to explore the opportunities to apply AI to board games, as discussed in today's lecture. Note that there is no need to complete *all* subtasks before progressing to subsequent objectives. You are free to explore unfinished objectives as homework.

First Objective

With reference to today's slides, create a C++ function to evaluate the *nim-sum*. Call it from a simple `main` function. You can start with the initial Visual Studio project, nim-sum, created using CMake.

a. Start with the function provided which takes *three* `unsigned` values; corresponding to a game of nim with *three* heaps of stones. It currently returns 0. Change it to return the *nim-sum*.

b. Then, develop a new *nim-sum* function, to handle games involving an arbitrary number of heaps.

c. To handle an arbitrary (nary) number of heaps, did you use variadic templates, or a container type such as `std::vector`? Can both handle a user specifying the number of heaps at runtime?

Second Objective

Change the *startup* project to nim. Build and run the program. Note that the computer player currently does nothing - the human player can only lose.

a. Look at the `nim::human_move` member function. Both it and `nim::computer_move` are called within `nim::play`; though `nim::computer_move` is empty. To start with, add code to `nim::computer_move` to take one stone from the first non-empty pile.

b. Improve the computer player. Perhaps use the `nim_sum` function you created in the first objective ... along with the algorithm described in the lecture slides.

Third Objective

Obtain a copy of the chess program: **WinBoard**. Either download it from `http://hgm.nubati.net`, or build it from source by downloading the Visual Studio 2022 version from Aula (xboard-4.9.1.zip); and opening `xboard-4.9.1\winboard\winboard.sln`.

Then, experiment with the UCI Chess protocol specified in engine-interface.txt and applied in sunil-orig.cpp and sunil2-tidy.cpp.

a. Use CMake to generate the uci-chess-engines Visual Studio solution.

b. The sunil2-tidy project is a tidy version of sunil-orig. Ensure sunil2-tidy is the startup project, and build it.

c. Find the location of the exe you just built using Windows Explorer.

d. Use the instructions from the last slide of this week's lecture to configure WinBoard.exe to play against this program (created from sunil2-tidy.cpp).

e. The main action of the initial code is to move the pawn in the leftmost column (column 'a'), from its initial row (row 7), forward by two rows (to row 5). For the next AI move, another column is selected. How many valid moves will this "chess engine" make (for now)?

f. Modify either engine to maintain the state of the chess board, and play with a hint of intelligence.

g. Implement Minimax along with a suitable cut-off depth and evaluation function. Again, refer to this week's slides for guidance with Minimax. This is a substantial piece of work.