CS3105 Practical 2: Torus Checkers

130013466

April 8, 2016

Abstract

The aim of the practice was to implement the game logic a version of checkers called Torus checkers, in which there were no kings and the board wrapped around as in a torus. A Alpha Beta MiniMax search tree was then constructed to give moves given a state to play from.

Part I

Introduction

This was a challenging yet fun practical. Most of the requirements were met but not all. The basic requirement to put a time limit on the computation of the was not implemented but luckily the program ran very fast so it isn't really needed to get below five seconds in any case.

Part II

Design and Implementation

As the nature of the program was to give and input and receive an output with no side effects, except the IO, it seemed obvious that this problem would be well suited to a functional language. The lazy evaluation of Haskell also seemed like it would suit itself very well to the MiniMax algorithm since the whole search tree could be defined but only the required states would actually be evaluated with little effort from the programmer. The program is split into IO, AI, and Game logic. Thanks to these choices the program ran extremely fast at deep depths.

Part III

Testing

For the logic of the torus positions I used hunit tests provided in CheckersTests.hs. This was very useful while working with quite confusing code and has given me confidence that it works correctly. For all other mechanic I tested them manually multiple times but they were all far more manageable. The only part with dubious testing would be the AI as the basic opponent provided was too simple and running it against itself seemed pointless because potential bugs they both contained could go unnoticed quite easily. Though the efficacy of the AI is doubted the validity of the outputted moves seems concrete.

Part IV

Evaluation

Unfortunately implementing the game logic was quite time consuming so after the AI was finally written there was no time to create a more sophisticated evaluation function. Thankfully the program can easily run at a deep depth so it still play reasonably well due to how far it can look ahead. In some ways this allows it to basically run as a Monte-Carlo search tree but with a slightly odd evaluation.

Note

I would like to enter the competition. If you need me to write a function to print out the next state then feel free to email me to ask.