Quarto! Minimax player
IT3105

Nordmoen, Jørgen H.
Østensen, Trond

October 24, 2012

Abstract

This paper is an introduction to our Quarto! minimax player. In this we will try to explain how we created our implementation, what sort of decisions the player makes the reasoning behind it and our result both against other configurations of our player and against other students minimax implementations.

Contents

1 Introduction

The game of Quarto![1] is a quite simple game regarding its rules, but the hard part comes into play when us humans try to remember all the different attributes of the game. This is where a game playing algorithm comes into play. With an algorithm memory no longer becomes an issue as the algorithm can enumerate, if possible, and search through the whole instance space. In this assignment we were tasked with creating a minimax[2] player with Alpha-beta pruning[3]. The algorithm it self is not the most difficult one to implement, thats not to say that we didn't need to debug our code, but the challenge is coming up with a good set of heuristics to evaluate an intermediate state. We will first introduce our implementation in a birds eye view, we will then explain our chosen heuristics in more detail before we move on to our experience in the tournament. Lastly we will describe our results, both against our own implementation and our tournament results.

2 Implementation

Since we did the last project in Java creating a highly parallel algorithm for roll-out simulation we wanted a different challenge this time around. Because of this we decided that we wanted to create our implementation in Python, but because Python can be quite slow compared to others we wanted to implement the time critical Minimax algorithm in C and interface that with our Python code.

Since our implementation had to be supported in both Python and in C we decided quite early on that we needed to represent pieces as simple as possible. After some testing we found that representing each piece as an byte worked very well. Using a byte where each bit represented an attribute gave us a very easy task of sending information between Python and C and made our implementation very fast. There were several advantages representing our pieces as a single byte, not only did it help us interface Python and C, but the representation would not have to change if we needed to play with someone else using this

[1]Explanation of rules in video form: https://www.youtube.com/watch?feature=player_embedded&v=P6dy2eaYmos#!

[2]Explanation of minimax from Wikipedia: https://en.wikipedia.org/wiki/Minimax

[3] Further explanation on Wikipedia:https://en.wikipedia.org/wiki/Alpha-beta_pruning

same representation. This is because the method of comparing similarity would not have to change even though we gave different attributes to different bits in the implementation.

Not everything could interface as easily as our pieces though. Since Python is inherently object-oriented our representation of the Quarto!board

3 Tournament

4 Results

4.1 Local

4.2 Tournament

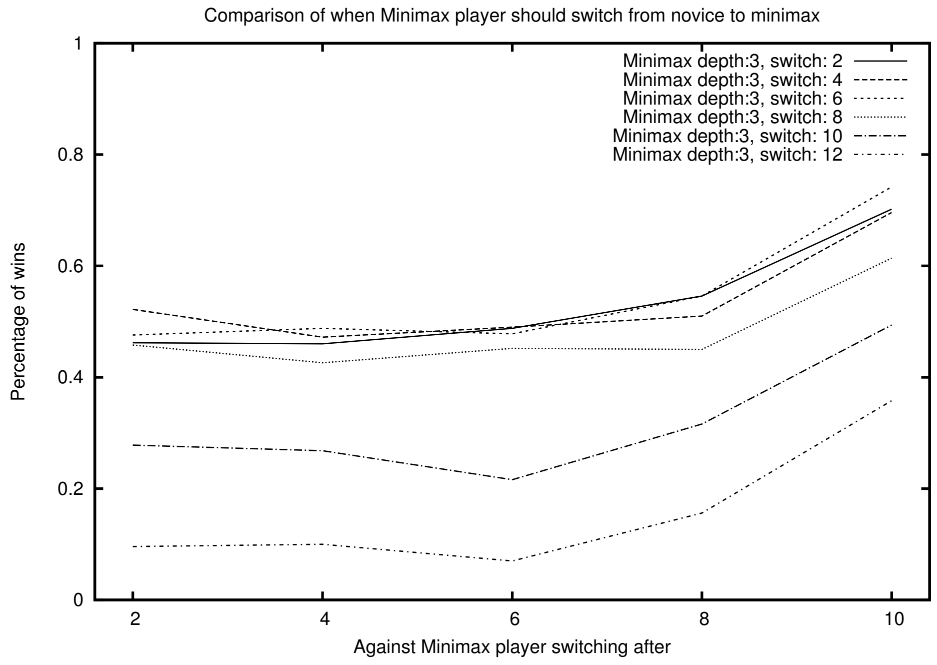

Figure 1: Graph describing our minimax switch results