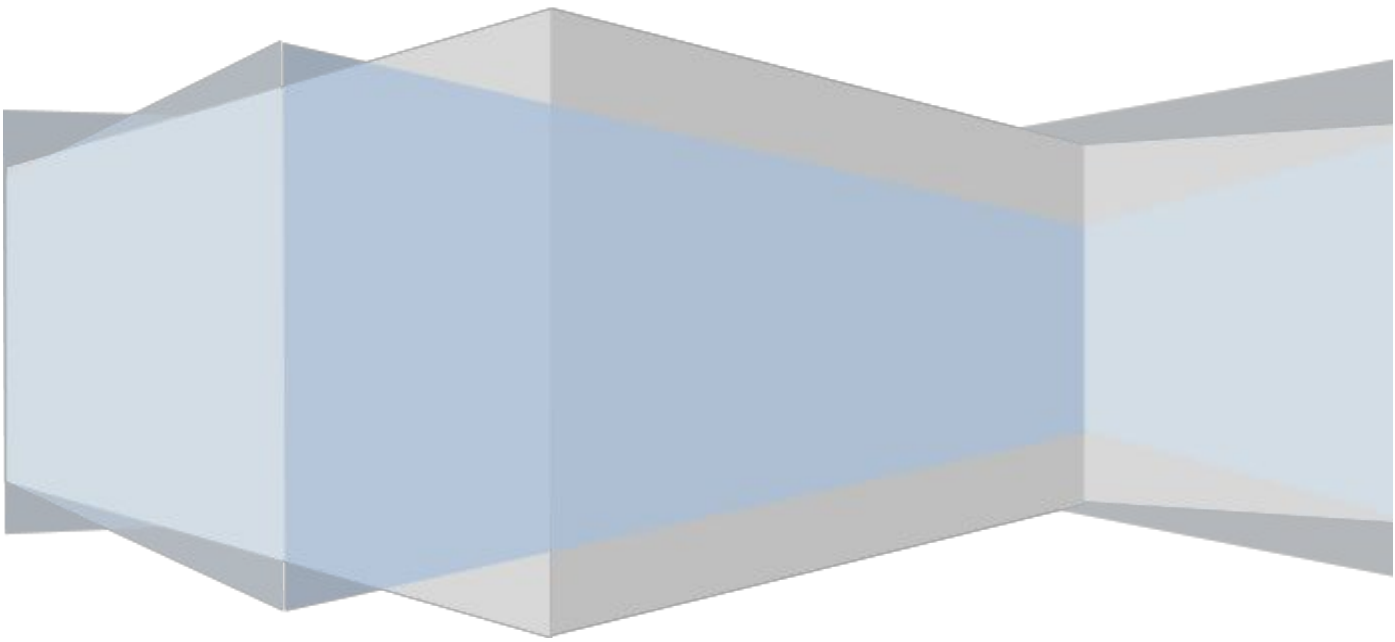
Technical Document

Monte Carlos Tree Search

Jason Butterfield



Introduction

This document is about the techniques applied to the project "Take2Checkers" Written in C++, in which it demonstrates the technique Monte Carlos Tree search to create a AI to challenge the player in the game of Checkers.

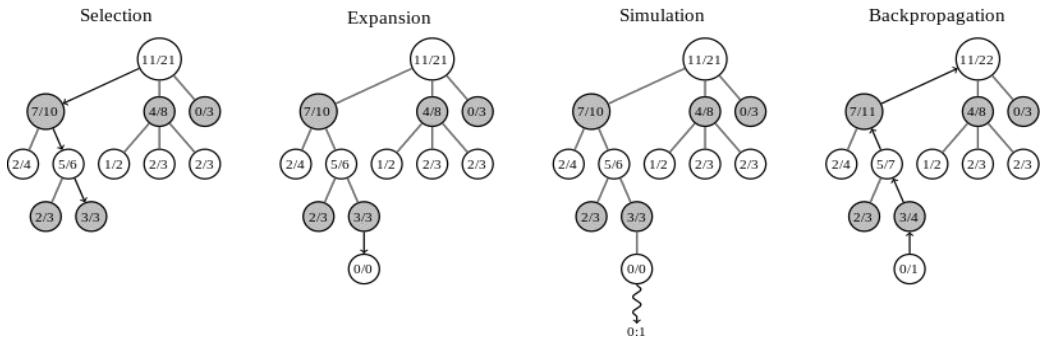
Monte Carlos tree search Technique

Monte Carlos Tree Search first off works by simulating possible play options of the current game for both players and returns a value in which is deemed "Best Action" and returns that action to be played by the AI,

While using this method it can be used in such a way to find all possible outcomes of the game, as each option is taken by this technique, from each option a weight value is returned to determine the "Value" of that action and to help decide what is considered a perfect move.

Also using this method you can make it limited by how many times it clones and replays the current board, to decide in which option is best suited to the current board or a quick move that may have not been the best option.

(Diagram of the Monte Carlos Tree Search)



(https://en.wikipedia.org/wiki/Monte_Carlo_tree_search#/media/File:MCTS_(English).svg)

How it has been Implemented

How the MCTS(“Monte Carlos Tree search”) has been implemented in the project “Take2 Checkers” is the same as the concept but changes to the algorithm with how many interactions the game clones and plays, by adjusting the amount of games played on the clone,which allows for more branches of the current game to be created, each branch that is created is a possible outcome of the game.

“CodeSnippet below from Take2Checkers”

```
//b.For each playout :
for (int a = 0; a < m_playouts; a++)
{//
    //Clone the current game
    m_clone = game.Clone();

    //ii.Perform the action
    m_clone->performAction(&action);
    //Untill time has been meet or there is a end state
    while (m_clone->isGameOver() == STATE::UNKNOWN && timer.Elapsed() > seconds(1.0))
```

Also added to the MCTS algorithm is a timer object, By adding this incorporates a limit on how long each simulated game will take, due to the nature of the MCTS algorithm it would find each possible ending with the cost of waiting five minutes just to make a move in which ruin’s game value as the player might be quick to make moves.

Not much has come to mind in adjusting it for multiple levels for now but it is possible.

Also added to the algorithm is a temporary action in which if the AI has no moves at all to make the game deems it as over and lets you embrace your victory.

What hasn’t been Implemented

What the project hasn’t implemented is an adjusted weight value based of the opponent's possible future move, in which if the AI was to make a move and lose a piece due to it, in which that would return a negative value unless it got a piece from that move, in which it would be a normal weight value, But if the move was to get a double jump it’s weight value would increase by a extreme amount.

In that regards anything that causes the AI to come out on top of say a Piece Trade then it would be a positive weight to do that move, While the game still looks for a winning gamestate it also looks at the value of each move in what it could of gain from that option.

The end of the document