

Technical Design Document

**Dominion AI Using Monte Carlo Tree Search (MCTS)**

Version 1.0

|  |  |  |
| --- | --- | --- |
| **Position Title** | **Name** | **Signature** |
| Programmer | *Jonathan Sarasua* |  |
| Advisor | *Matt Butler* |  |

Table of Contents

[Table of Figures 5](#_Toc64922132)

[Introduction 6](#_Toc64922133)

[Overview 7](#_Toc64922134)

[Scope 7](#_Toc64922135)

[End Product 7](#_Toc64922136)

[Deliverables 8](#_Toc64922137)

[Table of deliverables 8](#_Toc64922138)

[Technology Sources 9](#_Toc64922139)

[Background and Previous Work 10](#_Toc64922140)

[MCTS 10](#_Toc64922141)

[Dominion 10](#_Toc64922143)

[Theory 12](#_Toc64922144)

[Monte Carlo Tree Search 12](#_Toc64922145)

[Selection 12](#_Toc64922147)

[Expansion 13](#_Toc64922149)

[Simulation 13](#_Toc64922150)

[Backpropagation 13](#_Toc64922151)

[Choosing the best move 13](#_Toc64922152)

[Implementation 14](#_Toc64922153)

[Chance Nodes 14](#_Toc64922154)

[Selection 14](#_Toc64922155)

[Expansion 14](#_Toc64922156)

[Simulation 14](#_Toc64922157)

[Backpropagation 14](#_Toc64922158)

[Architecture 14](#_Toc64922159)

[Results 15](#_Toc64922160)

[Selection 15](#_Toc64922161)

[Expansion 15](#_Toc64922162)

[Simulation 15](#_Toc64922163)

[Bibliography 16](#_Toc64922164)

# Introduction

Dominion is a deckbuilding game where every turn you try to play and buy cards to make your deck better with the eventual goal of buying more victory point cards than your opponents. The artifact is both a playable game of Dominion and an AI using MCTS as its algorithm. The AI is implemented with chance nodes to handle the hidden information and uses Upper Confidence Bounds Applied to Trees (UCT) as the selection method. The AI is tweakable to show how changes to the AI change the result. Simple state machine strategies for Dominion like “Big Money” and “Single Witch” are used to test the strength of the AI.

# Overview

Deckbuilding games like Dominion have built in complexity with handling hidden information like not knowing what cards will be drawn or what cards the opponents have. Every card added to the player’s deck changes the way the deck plays. MCTS is an exciting AI strategy because of not relying on domain knowledge. The focus can then be on making minor changes instead of trying to solve the entire game.

## Scope

Building an AI first requires building the game of Dominion. All of Dominion is a massive game, so it was simplified into only using a subset of cards and also the cards would not be changed out each game like in regular Dominion.

## End Product

Monte Carlo Tree Search AI

* Tree
  + Data structure to store game state, pointer parent, child nodes, and metadata
* Run Iteration Step
  + Select
  + Expand
  + Simulate/Rollout
  + Backpropagate
* Run AI in a separate thread
* Run simulation step using a job system

Dominion

* Cards
  + Treasure
    - Copper
    - Silver
    - Gold
  + Victory
    - Estate
    - Duchy
    - Province
  + Curse
    - Curse
  + Action
    - Village
    - Smithy
    - Laboratory
    - Festival
    - Market
    - Witch
    - Council Room
* Player Actions
  + Play Card
  + Buy Card
  + End Phase
* UI
  + Mouse controlled UI System
  + Controls to play moves, switch between players
  + Controls to modify the AIs playing for player 1 and 2

# Deliverables

## Table of deliverables

Table of all of the features in a similar format to what Danny did

# Technology Sources

Technology sources used:

* Visual Studio
* TinyXml2
* STBI

# Background and Previous Work

### MCTS

MCTS has been used with many different games like Alpha Zero with chess. [1] It is commonly mixed with other AI strategies with thousands of papers covering a wide range of use cases. MCTS with Dominion though has been researched very little. Only one paper was found on the topic by Jansen and Tollisen. [1] They changed the simulation step to use a greedy heuristic to solve issues random has with the game of Dominion. They also tested using UCB versus UCT for selection and a multitude of other tweaks to MCTS.

### 

Figure 3: Dominion setup [2]

### Dominion

**Game setup for a two player game:**

* Card piles are set up for each card being used as shown in the above image
* Treasure cards use their entire pile size
* Each victory card pile has a size of eight
* Each action card pile has a size of ten

**Goal:**

* Have the most victory points from victory cards by the end of the game

**Card Types:**

A picture containing text

Description automatically generated

Figure 5: Curse and Victory cards [4]

A picture containing text

Description automatically generated

Figure 4: Treasure cards [4]

A picture containing calendar

Description automatically generated

Figure 6: Examples of Action cards [4]

**Game Ends When:**

* Any three piles are empty or the Province card pile is empty

**Start of game:**

* Each player starts with a deck of seven coppers and three estates
* The decks are shuffled and each player then draws five cards

**Each Turn:**

Player’s will take turns doing the following in order

1. Action Phase
   1. Play one action card
2. Buy Phase
   1. Buy one card from one of the card piles
      1. The card bought goes to the player’s discard pile
3. Cleanup Phase
   1. All cards in the player’s hand and play area are discarded to their discard pile
   2. A new hand of five cards are drawn
   3. All resources gained this turn are reset back to their starting amounts
      1. Money is set to zero
      2. Number of actions and buys is set to one

**Strategy:**

* Players want to buy action and treasure cards to build up enough money to buy victory cards
* The province pile running out is the normal way most games will end
* Trying to reach eight money consistently, so the player can repeatedly buy provinces each turn is the most common strategy

**Simple AI Strategies**

These are common baseline strategies players use to test if a strategy they are trying is good.

* Big Money [4]
  + If you have eight money, buy a province
  + Else if you have six money buy a gold
  + Else if you have three money buy a silver
  + Else do nothing
* Single Witch
  + If you have five money and don’t have a witch, buy a witch
  + Else follow Big Money
* Double Witch
  + If you have money and don’t have two witches, buy a witch
  + Else follow Big Money

# Theory

## Monte Carlo Tree Search

MCTS is an AI strategy that tries to fuse the benefits of a tree and other AIs In pure MCTS the other AI is random moves, but it could be a heuristic, genetic algorithm, or even a neural network. A tree in games is a data structure where each node contains a game state and a list of child nodes with an input move to reach that node. There is a starting root node that becomes a massive branching tree from all of its child nodes and their child nodes and so forth.

If a complete tree of all game states existed for a game, you could solve for which move is the best. Unfortunately most games are too large to completely map in a tree, and a tree search by itself can’t give a result without being complete. MCTS is a method of trying to build a tree while always able to return what the tree believes is the best move. It will explore the most promising moves in a game first while still exploring other moves. Below are the four steps done in a loop as many times as possible. The loops done means the AI is more confident in the move it is choosing.

### 

Figure 1: MCTS Steps [2]

### Selection

The first step in MCTS is selection. Here the AI moves down the tree looking for the best node to expand. Starting with the root node, two questions are asked:

Do all possible moves from the current node have a corresponding child node?

If yes, we have found our node we need to expand, so we move the expansion step.

If no, what child node is the best to continue down the tree?

The first question is yes only if the number of child nodes is equal to the number of possible moves. If not all moves have been expanded, then a random move of the remaining choices is chosen for the expansion step. [1]

To choose the best child node to continue the selection process, we use the Upper Confidence Bounds Applied To Trees (UCT) formula to calculate a value for each child node. Whichever node has the highest value will be the new current node and we repeat the questions listed above.

##### Upper Confidence Bounds Applied To Trees (UCT)

### 

Figure 2: UCT formula [5]

The UCT formula is the idea that exploitation should be balanced with exploration. [6] Exploitation is looking deep into the tree focusing on the best move. Best move is normally considered highest win rate. Exploration is looking wide into the tree. Looking at the UCT formula, wi/ni is the number of wins at the current node divided by the number of simulations. This is the winrate at the current node and is a value between zero and one. The second half tries to nudge selection towards the least commonly visited children. It is a value that should also have similar range of zero to one with C being the exploration constant to change how much we should balance exploration and exploitation. C should theoretically be the Square root of two. [1] Ni is the number of simulations the current node’s parent has done and ni is the number of simulations the current node has done. Note that the parent’s number of simulations is the sum of its children plus one where the one is the first visit to the parent.

### Expansion

Once the selection step has chosen the best node to expand, the expansion will get a random move that can be made that is not already a child. It will then play that move and create a new node with the new game state stored inside. This new node will be added as a child to the currently selected node. The new node is passed along to the simulation step.

### Simulation

Given the node from the expansion step, a game of random moves for both players is played. The simulation step will the result of the game and will then be passed to the backpropagation step. Instead of random moves, any strategy of playing the game could also be used. Using other AIs could speed up how quickly MCTS finds a strong move.

### Backpropagation

Backpropagation takes the result from the simulation and saves it in the current node. Each node has the number of wins and simulations saved. The result is then sent up the tree where each parent also saves the result. Note that the number of wins is relative to the player that played the move to reach the current game state, so as the result gets sent up the tree it will flip between a win or loss depending on the current node.

### Choosing the best move

One of the benefits of MCTS is the ability to ask the tree at any point what it believes the best move is. The two most common methods is getting the move with the highest win rate or the move that has been simulated the most. Using win rate means the result may change more often especially at lower simulation counts since it has not explore many different paths. Using the most simulated means you are putting your faith in the selection algorithm on which it believes it has as the most confident and thus the most winning move.

# Implementation

Here I will talk about my implementation of MCTS and include where I differed from pure MCTS. Using heuristics as the simulation method, not allowing all expansion paths, etc.

### Chance Nodes

### Selection

### Expansion

### Simulation

### Backpropagation

# Architecture

Here I will talk about how the code is laid out. MCTS is separate from the Dominion game and must ask the game for what is playable and what isn’t. Also all moves still go through the game. The basic methods for MCTS match up to the four phases

#### *Engine*

#### *Game*

# Results

Here I will talk about lessons learned and what worked the best. I will give data on how much better with tables.

### Selection

### Expansion

### Simulation

# Bibliography

|  |  |
| --- | --- |
| [1] | J. V. Jansen and R. Tollisen, "An AI for Dominion Based on Monte-Carlo Methods," 2 June 2014. [Online]. Available: https://pdfs.semanticscholar.org/28b6/ada13e948cfaee4af5138ee667d404eb01ac.pdf. [Accessed 9 July 2020]. |
| [2] | "Monte Carlo Tree Search: Implementation for Tic-Tac-Toe," Stack Overflow, 22 April 2014. [Online]. Available: https://stackoverflow.com/questions/23803186/monte-carlo-tree-search-implementation-for-tic-tac-toe. [Accessed 9 February 2021]. |
| [3] | "Dominion\_(card\_game)," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Dominion\_(card\_game). [Accessed 9 February 2021]. |
| [4] | "Dominion Beginners Guide | Tutorial," UltraBoardGames, [Online]. Available: https://www.ultraboardgames.com/dominion/beginners-guide.php. [Accessed 9 February 2021]. |
| [5] | "Monte Carlo Tree Search," Wikpedia, [Online]. Available: https://en.wikipedia.org/wiki/Monte\_Carlo\_tree\_search. [Accessed 9 February 2021]. |
| [6] | L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML*, 2006. |
| [7] | "How Does Monte Carlo Search Tree work?," Stack Overflow, June 2017. [Online]. Available: https://stackoverflow.com/questions/44230911/how-does-monte-carlo-search-tree-work. [Accessed July 2020]. |
| [8] | "Big Money," 20 October 2018. [Online]. Available: http://wiki.dominionstrategy.com/index.php/Big\_money. [Accessed 9 July 2020]. |