Thesis Journal

# Week 4

Ais for Dominion Using Monte-Carlo Tree Search

This comes from a google book link. It is very close to what I want to create, so it would be a great source to understand the challenges I’ll face.

The paper starts by saying Dominion is difficult to make an AI because of hidden information and stochastic elements. The paper uses Upper Confidence Bounds (UCB) and Upper Confidence Bounds applied to Trees (UCT). The author explains the basic rules of Dominion and also states a big challenge is that each game will have different starting cards which cause different behaviors. Other Dominion AIs use state machines. UCT and UCB are used as part of the selection formula to determine which tree path to explore. The difference being UCT has a exploration constant C that can allow for more tuning. They created finite state machines for comparison being AI-BigMoney, AI-SingleWitch, and AI-Random. They found SingleWitch is by far the best of these finite state machines, so they compared it agains UCB and UCT. UCB started beating Single Witch at 10,000 simulations whereas UCT needed 100,000 with a tuned C of 5. When played against each other, UCB outperformed UCT.

Current Approaches in Applied Artificial Intelligence by Robin Tollisen, Jon Vegard Jansen, Morten Godwin, and Sondre Glimsdal

<https://books.google.com/books?hl=en&lr=&id=bLjuCAAAQBAJ&oi=fnd&pg=PA43&dq=AIs+for+Dominion+Using+Monte-Carlo+Tree+Search&ots=gL6PXgf-je&sig=jP7rExJref-Y3uc-jyh8MOP4YIY#v=onepage&q=AIs%20for%20Dominion%20Using%20Monte-Carlo%20Tree%20Search&f=false>

EDIT: Found later that a more complete version is below. I’ll be using this as a research source going forward although I haven’t had time to go through it fully:

<https://pdfs.semanticscholar.org/28b6/ada13e948cfaee4af5138ee667d404eb01ac.pdf>

Playing Prismata

I decided to play Prismata because of the GDC video on using Hierarchical Portfolio Search for it. Its also a Card game, so it would be a good idea to understand the game first to see if I should use it instead of MCTS. Playing the game, what jumps out is this is a deterministic game with perfect information. It is a turn based game where each player takes a turn choosing what to buy to make their side stronger (resources, combat, or defense). When you end your turn you will have X number of Defense and Y number of Combat. Your combat goes towards destroying your opponents defenses and any extra combat can go towards destroying the fragile combat or resource units. When you begin your turn you first have to assign your opponents combat to your defenders. It is critical that you keep up your defense, so you can choose where the damage goes. If you have no defense, then your opponent will be able to choose where the damage goes.

After playing this game for a little while I realized this is quite a different game than the deckbuilders I want to build an AI for. There are significantly more choices being made per turn, and you have perfect information. While it may be possible to use HPS, this game is not a good measure of if it will work for what I want to do.

Playing Your Cards Right: The Hierarchical Portfolio Search AI of Prismata

This is a GDC talk explaining the game Prismata and how they used a Hierarchical Portfolio Search. They wanted the AI to be able to teach new and experienced layers, add replay value, and allow for modular design. If a new card changes because of balancing, the AI should accommodate for it. They didn’t use MCTS or other heuristics because of the large number of actions. There are 4 different phases where you could have “partial players” where these PPs could play just one phase. A player move then would be choosing a PP for each phase. These PPs would have to generate tactical moves themselves. For instance the buy phase PP could buy Attack or they could buy Defense, etc.

David explains then for an RTS each unit could have a Portfolio. For instance Attack Closest, Kite Enemy, etc. You can use HPS WITH other algorithms to generate the portfolio. You will generate each combination as a “move”. The algorithms Prismata uses MCTS, Alpha Beta, and Random Selection. The move choice then is random. The algorithm is awesome for adaptation. You can always change what algorithms to use. It allows the designer to tweak portfolios. The results show that MCTS was the best algorithm under the hood. This is important because it means that HPS for my thesis would be an after thought. I would still need to prove MCTS works best for deckbuilders before I could even do HPS to allow for development. The great use of HPS it could be used for other things besides game behavior like bug testing, balance testing, and stress testing.

The idea of HPS

<https://www.youtube.com/watch?v=sQSL9j7W7uA>

Big Money

Big Money is a strategy used to play Dominion. Its focus is on buying Treasure cards with very few action cards. The pure version of it is ONLY buying basic cards. Because of the simplicity of this strategy its useful as a baseline AI. The pure version follows these rules: Buy Provinces at 8 money, buy Gold with 6-7, buy silver with 3-5, and buy nothing otherwise. There are simple optimizations that can be made like buying Duchies at 5 money if there are 5 provinces left. The reason this strategy is good is beginners like to buy the “fancy Actions” even if they don’t fully understand how the game works. Big Money is a way of confirming if your strategy is good enough. Another benefit of Big Money is it works in every set of Dominion and requires very little thinking. It is also a good way of iterating on strategies. If you do Big Money plus one action, you’ll normally beat Big Money. What about 2 or 3? The most common way of making Big Money better is “Terminal draw”. This means adding action cards that add lots of draw but don’t give more actions themselves. The big example is Smithy which gives +3 cards only. This allows for drawing more treasure cards which you should have really good ones of.

Big Money relies on two concepts: Money Density and Opportunity cost. Money density is the average value of money production in your deck. As a reference, you need money density of 1.6 to buy provinces with 5 card hands. Other cards give varying density changes and there’s more complexity which eventually would turn the game into a complicated state machine for all the different cards and for a complicated evaluation function. Opportunity cost means what are you giving up to buy your card. Smithys give lots of draw but each extra one is worth less and less, so it may be better to buy other cards. This is where the complexity of the game comes in so balancing Money density and Opportunity cost is the big challenge.

For my thesis Big Money can be super useful in testing my AI. I can write it first and then use it to tune my AI to what works best. I can also use variations of Big Money to get more variation for playtesting. I could dive in a little bit on a state machine to come up with better strategies.

<http://wiki.dominionstrategy.com/index.php/Big_Money>

# Week 3

For this weeks journal I tried to focus on AI in boardgames. Squirrel explained a little about Minimax and Monte Carlo Tree Search, so I wanted to take a deeper dive on these. I realized even though I have an article on Monte Carlo I hadn’t really understood it when I read it.

Game Ais with Minimax and Monte Carlo Tree Search

Philipp Muens starts by saying there are two modes of play: Aggressive where you make a move to win the game/set up a winning situation or Defensive where you prevent the opponent from winning the next round/set up to prevent future winning situations. This is perfect for explaining Tic-Tac-Toe to choose your next move. A game tree used for algorithms would be all possible moves the player can make and all possible moves then the opponent will make. He then takes a dive into Minimax. If you take chess and add up all of the white piece values while subtracting the black piece values, you’ve created an evaluation function. Of course you’d need a more complicated evaluation algorithm, but this would allow you to determine which is your best move. You can’t look one level deep though because chess is back and forth, so you must choose your best move while also assuming the opponent will choose their best move. White is maxing and black is minimizing. You want to look as deep as possible, but the amount of moves makes this difficult. To make it more manageable Alpha-Beta Pruning is used to remove the worst moves, so you don’t have to traverse as far down the tree. Deep Blue which beat Garry Kasparove heavily used parallized Alpha-Beta search algorithms.

The big problem with Minimax is its evaluation function. Montecarlo tries to solve this with randomly choosing moves and finding out which worked best. Monte Carlo uses the law of large numbers to guess that the one move that wins the most is probably the best. It uses the “Upper Confidence Bound 1 formula which is xi + C sqrt( ln( N ) / ni ) where xi is the average value of the game state, C is a constant “temperature” (he sets to 1.5), N isparent node visits and ni is current node visits. We start with a tree of only one move, then play a game from each of the possible moves and plug in the results of the moves. Unexplored moves have values approaching infinity, so you always explore those first. Once you’ve found a node that wins more, you extend its tree and continue again. To handle the high computation of this method, C can be changed. A high C means more exploring unvisited nodes, but a low C will hit visited nodes more to gather more info.

<https://towardsdatascience.com/game-ais-with-minimax-and-monte-carlo-tree-search-af2a177361b0>

Monte Carlo Tree Search in a Modern Board Game Framework

The paper describes a framework for playing complex board games with Settlers of Catan as a proof-of-concept. MCTS is used because it doesn’t require knowledge of how the game is played. Two changes are used, a simplification of Chance Node model from Expectimax and move groups to allow for use on a non-deterministic game. An avenuse of research is General Game Playing (GGP) where one algorithm can play multiple different types of games. A problem with MCTS is high branching can hide possible replies. He explains the graph state and cycle using “Placeables” as all pieces that can be placed. There are rule changes implemented to remove some of the branching and hidden moves. Monte Carlo uses four phases: Selection, Playouts, Expansion, and Backpropagation. Selection selects a node to research, Playouts plays a game from a specific node, Expansion is the new node that is added to the tree, and Backpropagation where information from the games traverses back up the tree. Selection uses UCT as the algorithm (research further). Playout uses pure random play since the focus of the paper is on no domain knowledge. For Backpropagation, Average was chosen from between Max, Average, Informed Average, and Mix since they say it’s the best. A chance node function replaces the backpropagation function when it is their turn. The function is the sum of the chance of the child nodes \* the value of the child nodes. An alternative is Grouped Chance Model where the chance node is grouped with the child nodes. This allows for faster converging of the expected average, but it can hide moves. For the experiment, there were no cutoff for playouts and 2 seconds per move. Roughly 1300 playours per second were calculated. The results showed MCTS outperformed the Group Chance Model changes. Looking into RAVE as a possible online learning model is a good place to look for improvements.

<https://project.dke.maastrichtuniversity.nl/games/files/bsc/Roelofs_Bsc-paper.pdf>

Expectiminimax

Expectiminimax is a variation of minimax where chance is part of the game. Instead of just the AI maxing and then minimizing for the player turn, a “chance” node is added which represents the random effect. Using a theoretical game of a die is thrown followed by a decision by the ai and opponent, the order would be “chance”, “max”, “min”. A proposed algorithm is shown which is a recursive tree traversal which has cases for the terminal node and the above 3. This information is useful to explain the other papers being read.

<https://en.wikipedia.org/wiki/Expectiminimax>

# Week 2

The Wavefunction Collapse Algorithm explained very clearly

Robert Heaton explains what Wave Function Collapse is in a general sense and walks through some examples. He first starts out by saying he is explaining the “Even Simpler Tiled Model” which is a slow but readable implementation. The first example is of planning a wedding dinner where you need to seat every person, but each person has a restriction on who they sit with. The algorithm first starts by saying all options are possible and then beings by “collapsing the wave function for a single seat” which means you pick a random name off the list of possible options. With this one decision made, the possible options for future seats are restricted, so you have to update all possible options for the other seats. Then you repeat this process again and again until every seat is taken or you reach a contradiction where a seat can’t be filled. If a contradiction occurs you can either rollback a few seats or you can start completely over (which is what Robert does).

The next example is on a basic tiled map of land, coast, and sea. Instead of using a bunch of rules (completely doable), he uses an image that gives us a basic idea of the rules. The rules look like “(SEA, COAST, LEFT) which means a SEA tile can be placed to the LEFT of a COAST tile. This rule would also need an accompanying (COAST, SEA, RIGHT) for the other tiles perspective. All of these rules can be generated by an image that checks the possibilities. The other part is calculating the frequency of each piece to change from pure randomly selecting of tiles to weighting the choices. On top of this, first start with the “low entropy” choices where the fewest options are available. We then repeat like the wedding.

He ends it by saying the “Overlapping Model” is the next step up. It is analagou to order-1 Markov chain

<https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/>

Playing Carcassonne on Steam

I chose Carcassonne because it had good reviews on Steam. I wanted to learn about the good and bad elements of their UI/UX. The digital Carcassone starts you out in a tutorial. It teaches you the basic controls and then immediately throws you into a game. The main button where you confirm placing your pieces is massive in the bottom left. Since the controls are very simple it was fine to have a large button although it did feel like a mobile port. Everything felt very pure tappable. WASD did not move around the map. Which didn’t feel very good, but you could click and drag to move the screen around. Points also weren’t shown, so it was very hard to see how well I was doing. They show confirmed points, but things like the monastery should still show their points they are currently giving (They give points for each adjacent tile) even if not all of the tiles have been placed. Another big problem is the lack of tooltips. There are numbers on the screen, but it isn’t obvious what they are. It would be a great help if there was even simple tooltips.

<https://store.steampowered.com/app/598810/Carcassonne__Tiles__Tactics/>

Playing Ticket To Ride On Steam

I chose Ticket To Ride because it also had good reviews on Steam. I started with the tutorial where it showed you are trying to score points by connecting cities with train routes. There are long term goals like new York to Miami, but it is made up of shorter routes like Miami to Atlanta. They chose a really cool way of playing cards for routes where you drag a card to the route and the top left corner has a bullseye, so you can no exactly which route is being chosen. Also the current route you are hovering over is highlighted with green or red to show if you can/cannot build it. When choosing your long term route cards, the map highlights the two cities, so you can easily determine if you want to choose it or not. It also darkens the rest of the map to make it even more obvious. This is a nice touch. Scoring is not obvious. They don’t explain that the longer routes give more points, and its hard to determine how well you are doing during the game. At the end it totals things nicely by going through each category one by one, but it would help to have in game scoring. There are also numbers that you have to guess at what they mean. Is that the number of trains you have left? No idea. I guess I’ll wait until I place a train to find out. The Ais turns went extremely quickly and allowed for a 15 minute game where normal games can last 2 hours. This was extremely helpful if I want to practice strategies. Overall, it was a good boardgaming experience.

https://store.steampowered.com/app/108200/Ticket\_to\_Ride/

## Week 1

AI-based Playtesting of Contemporary Boardgames

The paper analyzes the boardgame, Ticket to Ride using multiple different Ais to test balance of the game. They would each find the most desirable cities for each playstyle and using that knowledge could see what cities were best overall for all playstyles. This information could be used to balance the game. The paper also states that this information could be taken a step further and be used to generate maps itself that use strategies the developers want the players to use. This can cut out undesirable strategies from defeating the game.

The AI was also used to find issues with the rules. Another purpose behind the paper is to show how useful AI could be for playtesting boardgames to find flaws. As the rules in a boardgame grows, the game gets more complicated, and it becomes more and more important to have playtesters to balance the game. Hours spent finding problems could be reduced if an AI is allowed to jump into the playtesting.

<http://game.engineering.nyu.edu/wp-content/uploads/2017/06/ticket-ride-fdg2017-camera-ready.pdf>

The Truth About Digital Board Games

StoneMaier Games put out an article about their experience with Digital Boardgames. In it he first describes a full digital version to mean one where the game knows the rules, so it doesn’t mean versions like Tabletop Simulator which are physics based and assume the player knows the rules. They explain that digital versions actually increase the sales of the tabletop version most likely because it allows people to try the game out. They specifically mention that digital versions aren’t looked as a money making option for them. They want to be profitable, but they won’t make a lot of money. StoneMaeier then explains the process of picking a company to work with to make the games. They explain that boardgame developers then have to answer a bunch of questions once they’ve picked someone to make the game like how hands-on do they want to be in the process and what platforms will the game be on. Another big one is what versions of AI, local, and online multiplayer will they want. They said single player versus AI is most important followed by online multiplayer.

<https://stonemaiergames.com/the-truth-about-digital-board-games/>

An MCTS Agent For Ticket To Ride

This thesis written by Carina Huchler focuses on using Monte-Carlo Tree Search for Ticket To Ride. Carina explains the difficulties of writing an AI for Ticket to Ride since the game has a lot of unknown information. You don’t know what cards your opponents have and you don’t know what cards you will draw. She explains different search algorithms and the MCTS (Monte-Carlo Tree Search) is a best-first search algorithm. You build a tree and then choose the most promising one. MCTS has seen success in Catan and Bridge. The paper explains the rules of Ticket To ride, MCTS, and then how MCTS could be used for Ticket to Ride. She then explains how different types of MCTS agents compared against different players like Single Observer and Cheating Player. In conclusion MCTS is successful in Ticket to Ride with Flat Monte-Carlo working best. The thesis only focused on two players, so further research would be necessary to test if MCTS would work with more.

<https://project.dke.maastrichtuniversity.nl/games/files/msc/Huchler_thesis.pdf>