# Improving Dominion AI - Final Report

Joe Flynn, Andrew Taylor, Phil Cheng

***Abstract*—This research explores ways of improving existing Dominion AI agents through the use of a forward search in the player's actions phase as well as a genetic algorithm used to create ideal decks which the AI agent can use to plan their buys. The results from these areas show that there still is much room for improvement to these approaches, and that creating a general purpose AI agent for all expansions of the Dominion game is a very difficult task due to the large size of the search space and randomness inherent in the game.**

## I. Introduction

In this project, we investigate and present a planning and searching agent to play the game Dominion, which is a deck-building card game created by Donald X. Vaccarino and published by Rio Grande Games in 2008. Multiple expansions since the original have added to the game so that there are now over 350 different card types in the game, each with different abilities and types of effects. Although a much newer game than the classical games studied in the realm of artificial intelligence research like chess, go, poker, etc., Dominion has developed a large fan following and almost since its inception has drawn the attention for artificial intelligence solutions to play the game. However, in all the previous research we examined, the attempts to create such an artificial intelligence have been with limited success. Most importantly, no Dominion AI exists which can consistently beat skilled human players yet, which motivated us to look at improving upon what existed.


Fig. 1. Example Dominion Hand

## II. Background

At a high-level, the rules of Dominion include two to six players each taking turns drawing and playing cards, and collecting "victory points" in various ways. (See an example Dominion playing card hard in Figure 1.) Each game has its own unique "kingdom", which is the set of 10 randomly selected card types (from the entire Dominion card domain) which form the set of cards the players can use during the game. Each player's turn consists of three phases: Action, Buy, and Clean-up. During the Action phase, each player chooses and plays a card from their hand which can in turn produce a wide range of effects on the game including: providing the player with buying power (i.e. coins, potions, buys, etc.), giving the player additional action steps, attacking an opponent's hand or deck, or otherwise affecting the player's own hand, deck, or discard pile. During the Buy phase, each player uses their buying power to acquire new cards to strengthen their deck. During the Clean-up Phase, each player discards his/her cards and draws a new hand from his/her deck. "Victory points" are acquired depending on the specific cards played and bought. At the end of a game, the player with the most victory points wins.

There are two novel aspects of the game that led us to investigate this area and that drive many considerations in this project. First, unlike other popular card games such as Magic The Gathering where decks are unique and crafted beforehand, each player in Dominion begins a game with identical decks of unwanted cards and must build up the deck throughout the game. This forces players to carefully plan a deck to build towards, as well as compete for limited cards. Second, the cards available to build from are different every game, such that the dominant strategies are vastly different from one game to the next. This presents an interesting challenge for learning and planning systems.

In our approach to the game, we blended aspects of the previous research projects that we reviewed and then extended each aspect further. For example, like [1], we use a search tree for the Action phase. However, we include random reshuffles of the player's original deck implementing an approach similar to a Monte Carlo Search Tree (MCST) in reverse. Like [2], we recognized the importance of having a goal for which cards to buy during the Buy phase and devised a careful planner for determining this Buy plan. Like [3], we recognized the importance of having an effective game state representation to search over, and we attempted to apply reinforcement learning (RL) and hill-climbing to determine our final parameters for our game state evaluator. Like [4], our Action phase search uses a heuristic search algorithm, but the parameters over which it searches include simple heuristics as well as a genetically-evolved buying plan.

We also noticed an interesting aspect of previous research was that they typically trained on a small subset of the game, and could not be easily extended to the full set including all 12 expansion sets. To accomplish this, we realized a need for a planning phase to the game which executed before each game starts. This is important because each Dominion game randomly selects only 10 out the 350 possible card types to

be the ”kingdom” set used for the game, and thus every game is different. A human player would take some time before a game starts planning out which cards they would want to acquire based on their individual card abilities and synergistic effects, and we wanted to experiment with this as a potential game-changing aspect to Dominion AI.

Lastly, we added to the research in Dominion AI by designing and testing a game player that would work with the full set of Dominion cards including all 12 expansion sets. A previous research on evolving cards sets for Dominion by [5] pointed us to a java-based ”VDom” game engine [6] that has been created and evolved by an open-source community over the years, to include the full set of cards, but is limited in that it plays the cards only heuristically. We used the VDom engine as the baseline in our project, adding the Artificial Intelligence aspects of planning and searching described above to extend the capabilities of this existing system.

## III. Approach

The beginning of the approach involved modifying the VDom engine in order to be able to play games and gather statistics about these games. We started with testing out the simple heuristic players already defined in the VDom implementation. This gave us a framework with which to test.

After this was accomplished, we created our first VDom player, ”Joe”, who plays according to the well known ”Big Money” strategy. This strategy is very simple method of playing Dominion which relies on never buying action cards, and instead only buys treasure and victory cards. The play is very predictable, but it is considered a benchmark in the fact that it easily divides novice players from proficient players. While there is still randomness and luck involved in Dominion deck shuffling, in general a novice player who is just learning the game would typically struggle against a Big Money strategy, and an expert player should beat a Big Money strategy easily most of the time. For similar reasons, [4] and other research teams had previously used the Big Money player as their benchmark for performance. With the Big Money player implemented, we were able to create a benchmark with which we could evaluate the more complex player strategies that we worked on next.

The next player we created was a two-terminal buyer, ”Andrew”, which was an extension of the Big Money strategy, that also buys two additional action cards. The action cards are considered ”terminal” because they do not allow the player to play any additional cards afterwards in the same turn. These types of cards usually have a unique positive effect for the player. In general, adding a limited number of carefully terminals greatly strengthens the player’s deck. This was used as our second benchmark for our searching and planning player.

The next player we created was a step towards our final searching and planning player. This stepping-stone player was a simple heuristic player, ”Phil”, designed to optimize his Action phase and Buy phase card selections to meet simple heuristics which we considered important to increase the overall utility of the agent. In this sense, the player is a simple reflex agent. For the Action phase, this simply meant playing the cards that created additional actions and buys first, thus extending the player’s turn. For the Buy phase, this meant buying victory cards similar to Big Money, but also buying other high-cost action cards when the player could afford them.

After completing these heuristics, we created a clone function for the game state, which we would need for the forward planning and searching player. However, as a first order measurement, we decided to simply clone Phil’s game and look only one play ahead and evaluate the effectiveness on the player’s utility function. For the Action phase, cards that increased the player’s buying power or victory tokens, or harmed the opponent, were all considered to improve the player’s utility. For the Buy phase, cards that improved Phil’s overall game utility were cards that brought Phil closer to completing the game, or brought the player’s deck closer to an ideal number of actions.

At this point, we were ready to build a player agent who could pre-plan a game through several turns and also search multiple actions ahead to find which immediate action and buy would bring the player closer to optimal utility. This was our ”Jarvis” player, and the phases of the ”Jarvis” player are described next.

### A. Approach To Search

Due to the highly varied effects that Dominion cards can have, it is not possible to predict the outcome of playing the cards in hand without searching through each possible resultant state. Very few assumptions can be made about the best way to play a single card since there might be other cards that react to one of its effects, or synergistic resources that are not available in every game. For this reason, we emphasized searching through possible turn outcomes as completely as possible. In addition to the order in which to play cards, there are also various options presented to the player that must be searched through. These might be in the form of choosing from a set of effects, cards to interact with, or whether or not to react to a game event. To accomplish this search, we designed a tree that can hold different node types, including state nodes, card nodes, and option nodes. Card nodes are always children of state nodes, and may be action cards, treasure cards, or cards to buy. Option nodes may be children of any type of node and are added any time a decision is requested of the player. The states are compared by an evaluation of turn economy utility, which is driven primarily by coins and buys.

In a typical turn action phase, cards that are played remain in play until the Clean-up phase so that even a deck that is built to reliably draw and play every card will naturally terminate once all cards are in play. Under most circumstances, expanding an exhaustive tree of possible results is reasonable. However, there are a few studies that demonstrate kingdoms in which infinite play is possible. Even without infinite play being possible, there are times where the combination of options for how to play a series of cards may be too great to search entirely. To handle these cases gracefully, we implemented a width and depth constraint. When options are passed into the tree, they are ordered by an estimated likelihood of being

chosen. This way, if we need to restrict the expansion of a state, we can still easily try the best candidate options first. In addition, an iterative widening search [7] can be applied from here to increase the completeness of the search as resources allow.

Initially, we frequently ran out of memory due the presence of cards with higher branching factors. This was mitigated significantly by introducing a transposition table, to check if an equivalent game state had already been searched. Currently, the table is required to hold a large amount of information to compare states. This does not seem to impact the performance of the search since it is almost entirely driven by the time it takes to clone the game state. It also does not impact memory considerations since there are more game states in memory at any given time, and the size of an entry is a fraction of the overall state. If game state cloning were improved, or if we erased states from the tree during search, such as in a depth first search, then it is possible that this structure would become a major performance and memory driver. In this scenario we would likely need to come up with a way to hash states into the smallest possible bit signatures with a tolerable level of collision. Due to the nature of the game, there are no ways to unplay moves (as in retreating a knight in chess to a previous position), so the transpositions only happen within a turn when the same cards are played in varying orders and achieve the same effect. Thus, maintaining a transposition table between turns does not seem necessary, and we rebuild the table with each new search to determine what card to play next. However, there is likely much potential in leveraging this structure in accelerating future searches, allowing for a deeper searches on average.

### *B. Approach To Planning*

The main concept behind our work in planning is that we wanted it to be driven by our strong search. Other approaches we've seen in this area require highly simplified play in order to reach a number of iterations needed to converge on a dominant strategy. As much as possible, we wanted to avoid sacrificing playing strength so that plans based on sophisticated play are able to be found.

To accomplish this, a genetic algorithm was created to find an ideal deck which could be used by our player to plan out their buys. This algorithm takes place in a series of three phases where pools of decks are generated, evaluated, and culled to find the most ideal deck.

At the start of each game, the set of kingdom cards is retrieved with which all pairs of kingdom cards are found. Each of these pairs is used to generate the initial pool of decks the planner considers. These decks are composed of a set percentage of the kingdom card combos as well as a percentage of other cards which include the initial starting cards (7 Copper, 3 Estate), Silver, and Gold. For these decks, 80% of deck is composed of kingdom cards and 20% is composed of other cards. These percentages were chosen in order to quickly discover strategic synergies between the two cards by making them drawn together frequently. Among the 80% of the deck composed of Kingdom cards, there is an equal split among the pair of cards to simplify the initial deck generation. If one of the cards in the pair was favored, this might rule out better options of favoring the other cards in the pair. While it would be possible to generate both of these options, in the sake of time performance this ratio was kept simple.

In the other 20% of the deck, the 10 initial cards were added in addition to 1 Gold for every 2 Silver. This Gold to Silver ratio is based on an assumption of what would be the ideal ratio of Treasure in most decks. This ratio is also affected and fine tuned by mutations in the third generation. A deck size of 30 cards was chosen to mimic the general size of a mid game deck. With each of these initials decks, a series of 100 turns is played to evaluate the average turn economy over the set of turns. This number of turns was reached after experimentation in finding an amount of turns that was both manageable in time performance and large enough to stabilize the averages. The series of turns is played over 20 "games" consisting of 5 turns each. In each these "games", the player's deck is set to one of the initially generated decks, and they play according to their action search tree. This "game" contains no buys phase in order to just evaluate the decks themselves and is also played against a dummy player that plays no cards. After the conclusion of each Action phase, the evaluator is run to determine the turn economy resulting from the player's actions with the deck. These calculations are summed among all turns and are used to find the expected turn economy of the plan. Turn economy is a measure of how strong the buy phase will be, as well as how effective the action phase is, in terms of attacking the opponent or improving future turns.

After all the initial decks have been evaluated, the top 20% are retained to become the pool of survivors which are crossed over and mutated in the next two generations. With these survivors, all pairs of the surviving decks are generated which are then used to create a set of crossover decks. Among each pair, two child decks are created with the same Kingdom and other card percentages as the previous generation. The first of these decks has a 4:1 ratio split between each deck's kingdom card pairs with the second deck having a 1:4 ratio split. The decision for the unequal splits is that frequently a good plan will include a deck-drawing concept that requires many cards, and action payload that only needs one or two copies of each. After all of these child decks have been created, decks are randomly selected until 2 times the amount of survivors have been chosen. These decks are then added to the current pool with the survivors and are evaluated in the same manner as the parent decks.

After this evaluation, the deck with the highest average turn economy is selected for the final round of mutations. With this deck, 20 new decks are created each by applying 10 mutations to the deck for each new deck created. Each of these mutations picks two different cards – a "taking" card and a "receiving" card. The "taking" card has between a random 15% to 40% percent of its cards taken from the deck which are then replaced by the same amount of "receiving" cards. This percentage range was arbitrarily tweaked until it seemed to be moving a significant, but not overly significant amount of cards. The purpose of these mutations are to change the

ratio of cards among the deck to fine tune the cards ratios for the best plan. After these mutated decks are created, they are played in the same manner as previous generations to find their average turn economy. This does not necessarily result in a decks with higher average turn economy, but gains larger than 100% in average turn economy have been observed. The highest scoring deck among the final survivor from the first two generations and the new mutated decks is then chosen as the ideal deck that the player will use as a plan during their action Buy phase.

Our Jarvis player utilizes a deck plan by evaluating how close his deck is to the ideal planned deck. To make buys, Jarvis tries adding each possible combination of buys to the deck and returns the option that brings him the closest to the ideal. In this evaluation, each deck percentage is compared against the actual and the differences are summed. When there are more than the planned number of cards in the deck, we decided to still evaluate this positively up to a certain threshold but to reduce the evaluation of these extra cards by multiplying a dampening factor. In general in Dominion, the power difference between 4-cost cards and 5-cost cards is greater than other 1-cost differences, so a bonus multiplier is also applied to cards costing at least 5 so that these cards are typically bought when available unless there is a very good reason not to. Jarvis transitions into buying victory cards by evaluating game progress and scaling the score of buying victory up as the game gets closer to the end.

### C. Machine Learning

We originally proposed to add a machine learning aspect to our player if the time allowed, as we had read about several previous research projects into Dominion AI implementing various versions of learning in the past. Given our utility equations for both Action phase ”turn economy” and Buy phase ”game economy” were weighted sums of parameters of the current game state, we could manually tweak these parameters to arrive at a more optimal solution.

$$U = \sum_{n=1}^{N} Gain_n * GainFactor_n \tag{1}$$

The alternative to manually tweaking the Gain factors would be to let the machine do the work for us. We could play two versions of our Dominion AI against each other, slightly adapting the Gain Factors, and rewarding the agent who wins more games, thus over time gradually adapting the gain factors towards the ideal ratios. In the example of Action phase turn economy, the gains are parameters such as number of usable coins, number of potion cards, number of cards that a player is able to gain in a single turn, etc. that the agent would be trying to maximize. In the example of Buy phase game economy, the gain factors are deltas from an ideal ratios of treasure cards and action cards that the agent would be attempting to minimize. In both cases, machine reinforcement learning (RL) could be used to perform stochastic gradient descent towards these ideal gain factors.

## IV. Evaluations

To evaluate the performance of our Dominion AI, we wanted to use the Big Money and two-terminal buying agents (”Joe” and ”Andrew”) to evaluate the performance by win percentage against each of these players. For each player, we played several hundred games against these benchmarks to determine the approximate win percentages for each approach. See results in Figure 2.)

| **Player** | **% Wins Against:** | |
|---|---|---|
| | **“Joe”** (Big Money) | **“Andrew”** (2-Terminal Buy) |
| **“Joe”** (Big Money) | **X** | **22%** |
| **“Andrew”** (2-Terminal Buy) | **78%** | **X** |
| **“Phil”** | | |
| Heuristic Action Heuristic Buy | **43%** | **15%** |
| Heuristic Action Evaluative Buy | **48%** | **20%** |
| Evaluative Action Heuristic Buy | **5%** | **1%** |
| Evaluative Action Evaluative Buy | **17%** | **4%** |
| **“Jarvis”** | | |
| Heuristic Action Heuristic Buy | **45%** | **13%** |
| Heuristic Action Planning Buy | **35%** | **11%** |
| Searching Action Heuristic Buy | **37%** | **15%** |
| ***Searching Action Planning Buy*** | **28%** | **11%** |

Fig. 2. Performance Results

We began by evaluating the performance against the simple heuristic reflex agent (”Phil”) first. In this case, we noticed a significantly higher performance against the benchmarks when the Phil player used a simple heuristic for selecting his action card. The heuristic in this case always played cards that produced additional action phases first, followed by cards that allow drawing more cards, followed by cards that allowed more buying power, and finally he played the highest cost terminal card. Compared to the agent who only looked ahead a single card play, it is easy to see why the simple heuristic performed so much better. The single-card look ahead never gets over the horizon effect. We realized we would need to add a search algorithm that at least searches to the end of the current action phase or turn (which is exactly what we added for the next agent). It might have been possible to extract more performance out of the single-card look ahead by tweaking our utility function more, but at the time, we deemed this to be less valuable than working on the tree search for the more advanced player.

We also evaluated Phil's Buy Phase. In this case, using an evaluative buy, which compares the current utility with the utility gained by looking ahead one card buy, performed slightly better than than a simple heuristic buy. This was mostly because the heuristic buy was too simple, and bought cards based on their card cost only (since better cards typically cost more) but without considering the impact to utility at all. This further bolstered our need for our next agent to include a planning phase against which the buy utility could really be compared. That way the player would be able to buy specific cards that helped his overall plan, and not clog his deck with other cards that did not support better utility.

### A. Evaluation of the Search Tree

During each action phase, our agent searches the full turn for best utility to turn economy, which roughly translates to best buying power, so that the Buy phase will be successful. An Example of a successful search is shown in Figure 3. This is how a typical search would happen. In this case, the search tree tells the player to play Villa first instead of Woodcutter because although Woodcutter has a higher immediate evaluation (-1.0), playing Villa first leads to a better overall evaluation (5.25) further down the tree.

Next, we compared this strategy against a simple heuristic action, like in the Phil agent. Unfortunately, our search actually produced lower win percentages against the benchmarks than did the simple heuristic (losing on average about 20% more games). We think that the main reasons for the poor performance here were (a) a shallow depth of the search space, and (b) not enough tuning performed on the utility function itself.

The variety of cards in Dominion creates a very large tree of possible card interactions. The VDom implementation required that the player implement methods specific to each possible card interaction. We did not have enough time to override all of the heuristic functions in VDom's BasePlayer's implementations to feed the interactions into our tree, so Jarvis still has many blind spots where it can only try to play cards in one preset way. One of the weaknesses was in our cloning of the VDom engine's states, which cloned entire games in order to play a card and see its results without affecting the original game. Either having a more streamlined game implementation or more powerful processing hardware could have allowed us to search more widely, or implement an iterative widening search, such as [7]. Searching more deeply (i.e. into the player's next turn) could also overcome the inherent single-turn weakness we had when it comes to "duration" cards, which are cards with effects taking action in more than a single turn.

Given a more powerful search, we would also like to have to improved the utility function for the search. Our original intention was to use reinforcement learning to drive the utility function parameters to more ideal values; however, when experimenting with changing these parameters by hand, we did not see a significant change in performance. It was our opinion therefore that either the search depth or width needed to be extended first before the change in utility function would have more of an effect.

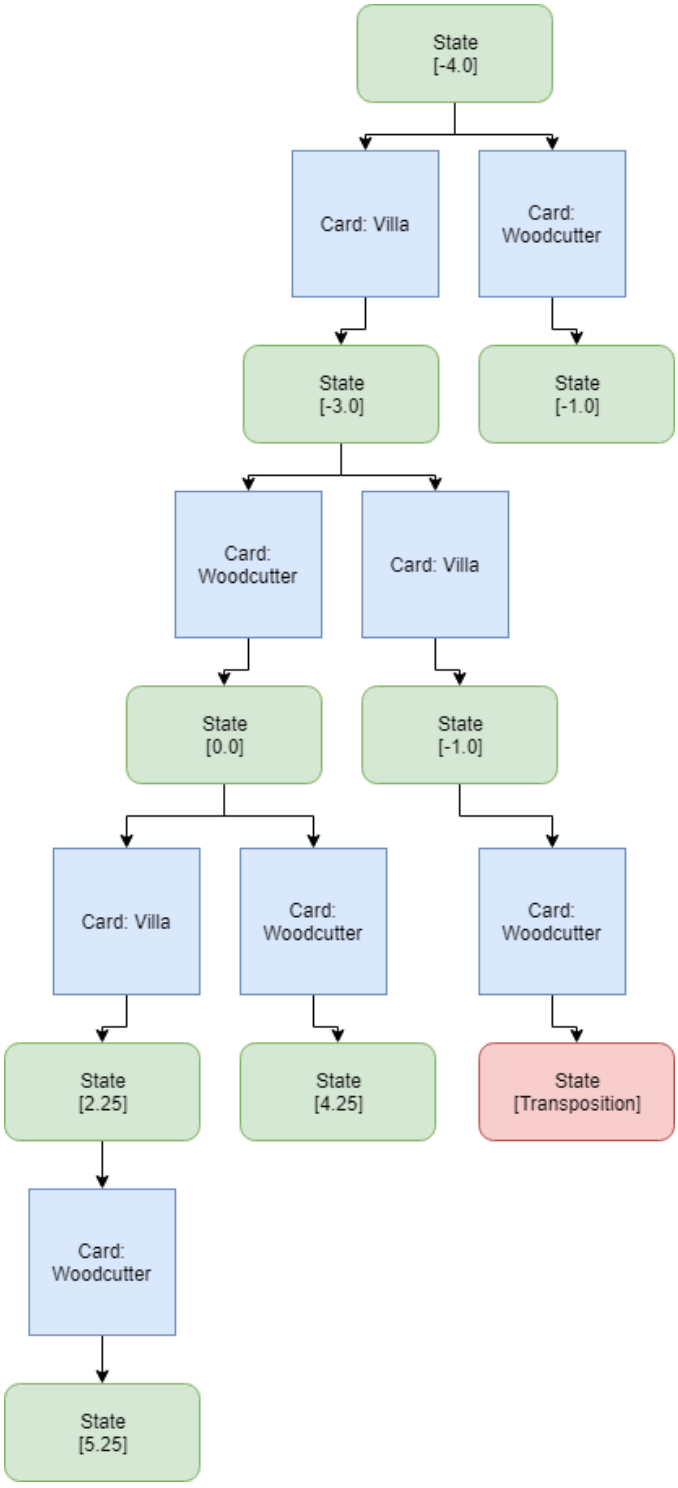


Fig. 3. Example of an Action Phase Search

### B. Evaluation of Planning

As seen by winning percentages of the planning player, our planning did not have the best results. From examining some of the decks it was creating, it seems like our turn economy heuristic is not always picking decks which we would consider to be good plans. While the kingdom card combinations chosen often seem to be what we would consider

good combinations, the ratio among the cards in the deck often is not in good proportions. The final mutation that changes these ratios should probably have been adjusted to not simply be based on randomly switching a random amount of cards. For many of the decks we have observed, it seems like the ratio of kingdom to treasure cards were not optimal. If the mutation instead tried to change this ratio, we think that the results would have been better.

It is also possible that we are not spending enough time during planning. It is normal for Human players take several minutes to decide on a plan for the game. We limited our planning to take less than a minute, so that we could run a large number of games to get good statistics on how our player was playing. If we did away with this limit, we could consider all the crossover decks made in the second phase of the planning as well as try many more rounds of mutations in the final round. This could potentially allow for our planner to find a much better ideal deck. It also would allow for the planner to find more stable average turn economies which would improve the calculation as to how ideal a deck is. In addition to make the phase longer, it could have sped up this planning through the use of threading; however, this would have been hard to implement with the existing vDom implementation, so it was not explored.

We also had to limit the search our planning player used as to not crash the program by exceeding the amount of available memory. If we had more powerful computers, or if we were able to better optimize the planning and search, we could have been able to search further in our planning games. This might have allowed for the player to find decks that it could play with better thus giving the player an overall better buying strategy.

## V. Related Work

The previous research into Dominion AI that we read about each attacked different areas of the game, summarized as such:

[1] created an approach using NeuroEvolution of Augmenting Topologies (NEAT) competitive co-evolution for determining which cards to acquire (buy) and a Monte-Carlo Tree Search (MCTS) based approach for determining which cards to play.

[2] created a very simplified game approach based on priority queues for cards to buy, which were evolved using genetic algorithms. The action-phase and end-game detection for the game however were simple and heuristically derived.

[3] presented a comparison of various neural network and machine learning techniques used successfully in other games, including Temporal Difference (TD) reinforcement learning and general hill-climbing techniques. This study uses a state representation including counts of each card in the supply, player's hand, discard pile, number of actions and buys the player can make, and counts of unknown cards, which is very similar to the representation we ended up using.

[4] implemented a heuristic search algorithm for the Action phase of the game, and uses "Average-Reward Evaluation" machine-learning to improve the Buy phase strategy.

## VI. Conclusion

From our research, the biggest takeaway should be that the nature of the Dominion makes it a very difficult to create an artificial intelligence that can perform well. The size of the search space and the randomness makes it a very memory intensive process to use forward search approach in playing. The randomness also makes it hard for our planner to find stable average turn economies.

To expand upon our research into planning, a variety of new mutations and ways of generating parent and crossover child decks could be explored. As mentioned earlier, trying mutations of the Kingdom card to other card ratio, would probably result in better decks than the more random mutations we are currently using especially if it was combined with the random mutations we are already doing. A longer planning phase (or a better optimized one) also would be a good area to explore, as it would allow for a much larger pool of decks to be considered as well as finding better, more stable average turn economies. Research further into a better action heuristic (average turn economy) might also be a good area of research since it is very likely that our initial assumptions on these utility functions were not optimal.

An idea that we had, but did not have time to implement, was to characterize cards in terms of measurements of common strategic effects. This would not be able to fully capture the impact of a card, so it could not replace search, but it might be able to give a learning system enough features to make accurate evaluations. Most high level strategic considerations in a kingdom revolve around the presence or absence of these common strategic features, so if a kingdom were characterized in terms of an aggregation of its these features, past experience from similarly characterized kingdoms might be applicable. This would hopefully avoid the issues we would expect from a learning system otherwise due to kingdom variance, that learned experience from one kingdom would only be applicable to a small percentage of games and would likely be incorrectly applied.

## References


[1] R. B. Fynbo and C. S. Nellemann, "Developing an agent for dominion using modern ai-approaches," *Master's Thesis, IT- University of Copenhagen*, 2010.

[2] M. Fisher, "Provincial: A kingdom-adaptive ai for dominion," *https://graphics.stanford.edu/ mdfisher/DominionAI.html*, 2012.

[3] R. K. Winder, "Methods for approximating value functions for the dominion card game," *Evolutionary Intelligence*, 2014.

[4] M. White and N. Walsh, "Automatic dominion-playing agent," *CS221 Project Final Report, Stanford University*, 2014.

[5] T. Mahlmann, J. Togelius, and G. N. Yannakakis, "Evolving card sets towards balancing dominion," *WCCI 2012 IEEE World Congress on Computer Intelligence*, 2012.

[6] Mehtank, "Androminon: Unofficial dominion for android," *https://github.com/mehtank/androminion*, 2017.

[7] T. Casanave, "Iterative widening," *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.