Universiteit Leiden
Opleiding Informatica

An Analysis of Dominion

Name:	Roelof van der Heijden
Date:	29/08/2014
Supervisors:	Dr. W.A. Kosters (LIACS), Dr. F.M. Spieksma (MI)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this paper we introduce a new kind of game, called a deck building game, of which Dominion is the most prominent example. We focus on the question to what extent traditional game analysis techniques can be used to analyze deck building games? To do this, we look at several simple strategies, like `Random` and `Greedy`, and some traditional techniques, namely Monte Carlo Tree Search and Dynamic Programming. We compare the strategies for mid (31 turns) to long games (100 turns). We conclude that our implementation of DP seems to be suitable only for games of medium length or shorter because of its space complexity, whereas our implementation of MCTS seems to fall behind other strategies with similar performance in regards of time complexity.

Contents

1 Introduction

In this paper we will introduce a new kind of game, called a deck building game. Several of these games have recently been designed and produced for the international board game market and gained a lot of popularity. Dominion, designed by Donald X. Vaccarino and produced by Rio Grande Games [5], is the most prominent example of deck building games [4]. As deck building games are a relatively new kind of game, there has not been very much research on the subject. The goal of this paper is to formulate an answer to the following question: to what extent can traditional game analysis techniques be used to analyze deck building games? To do this, we will look at several simple strategies, like `Random` and `Greedy`, and some traditional techniques, namely Monte Carlo Tree Search and Dynamic Programming. We will compare these strategies with other strategies specific to deck building games.

First we introduce the game components and definitions of deck building games. We define states, actions and strategies and define the expected score of a state when a certain strategy is followed. Next we take a look at some combinatorics of deck building games, to get an idea of the scope of the calculations that need to be performed. We also define state equivalence and prove that the expected scores of equivalent states are the same. After that we take a step back and talk about reachability and efficiency. These definitions will allow us to remove some cards from a game without affecting the strategies. We also prove that the best action for any strategy in any state has to be one of only three possible actions. We make extensive use of this theorem in the next section, Section 5, where we will list the tested strategies and techniques. These include, among others, implementations of Dynamic Programming and Monte Carlo Tree search. The experiments we ran and their results can be found in Section 6. These results are summarised in Section 7 and we propose some interesting questions for further research.

This paper has been written as part of the bachelor programs Computer Science and Mathematics at Leiden University. Walter Kosters (LIACS) and Floske Spieksma (MI) have been the supervisors for this project.

2 Game components and definitions

In this section we will define a deck building game and its components.

Each *deck building game* has several things in common. First of all they are card games, played over several turns. The player will have to try to gain as many points as possible before the game is over. He can do this by adding some specific cards which are worth points to his deck of cards.

Each turn a player can add at most 1 card to his deck. However, to add a certain card x to his deck, he needs to have a certain amount of money $c(x)$ available this turn. How much money he has available is determined by the cards in his hand. Each turn, he draws 5 cards randomly from his deck forming his hand. Just as some cards x give the player $p(x)$ points, other cards y are worth a certain amount of money, $v(y)$, called the value of a card. He adds the value of the cards in his hand to determine his total available amount of money m for this turn. After that, he can add a card costing at most m to his deck.

This is formalized in the following definition.

Definition 2.1 (Deck building game)**.** A *deck building game* G is a 7-tuple:

$$G = (TC,\, VC, c, v, p,\, GE, Q_0)$$

where

TC: the set of treasure cards, non-empty;

VC: the set of victory cards, non-empty and disjoint from TC;

c: a function $c\colon TC \cup VC \to \mathbb{N}$ associates a card with its cost;

v: a function $v\colon TC \cup VC \to \mathbb{N}$ associates a card with its value;

p: a function $p\colon TC \cup VC \to \mathbb{N}$ associates a card with its points;

GE: a function GE that determines when the game will end;

Q_0: a multiset with elements from $TC \cup VC$ and cardinality $|Q_0| \geq 5$ with which the player starts the game.

Definition 2.2 (Deck)**.** Let $G = (TC,\, VC, c, v, p,\, GE, Q_0)$ be a deck building game. A *deck* X is a multiset of cards from $TC \cup VC$ with cardinality $|X| \geq 5$.

In particular, Q_0 is a deck. It is sometimes called the game's *starting deck*.

For his turn a player draws 5 cards from his deck to form his hand.

Definition 2.3 (Hand)**.** A *hand* H is a multiset of cards from $TC \cup VC$ with cardinality 5.

Notation 2.4 (Collection of hands)**.** We use $\mathcal{H}_D$ to denote the collection of hands H of a deck D.

Notation 2.5 (Multiplicity)**.** Let X be a multiset of cards from $TC \cup VC$ and $x \in X$ a card. We use $q_x \in \mathbb{N}$ to denote the *multipicity* of x in X.

2.1 Game parameters

All these parameters influence the game in their own way. This section describes in more detail the individual parameters and how they influence the game.

2.1.1 Treasure cards

Treasure cards provide money for a player when drawn from the player's deck into his hand. Let TC be the non-empty set of treasure cards. As with all cards, to add these cards to the player's deck, a certain amount of money is needed. If $Tr \in TC$ is a treasure card, then $c(Tr)$ is its cost, while $v(Tr)$ is the value of the card when in the player's hand. The points $p(Tr)$ of a treasure card $Tr \in TC$ are defined to be 0.

2.1.2 Victory cards

Cards that provide points are called victory cards. Let VC be the non-empty set of victory cards. If $Vi \in VC$ is a victory card, then $c(Vi)$ is its cost, while $p(Vi)$ is the number of points

it is worth. The value $v(Vi)$ of a victory card $Vi \in VC$ is defined to be 0. The sets TC and VC are required to be disjoint.

2.1.3 Game end function

The game end function GE is a probability distribution on $\mathbb{N}$. The chance $GE(t)$ is the chance that the game will end on turn t, for all $t \in \mathbb{N}$. When the game ends, the points of all cards in the player's deck are counted leading to his final score. The goal of the player is to maximaze this score.

For this paper, we have looked at the following game end functions:

- games of a fixed number of turns $t_{max} \in \mathbb{N}$, that is

$$GE(t) = \begin{cases} 0 & \text{if } t < t_{max}, \\ 1 & \text{if } t \geq t_{max}. \end{cases}$$

- games with a constant ending chance $0 \leq \alpha \leq 1$ every turn, that is $GE(t) = (1-\alpha)^t \alpha$.

Note that other game end functions are certainly possible.

2.1.4 Starting deck

The player begins the game with several cards in his deck. These cards form his starting deck Q_0. They determine the actions the player can perform in his first turn when he has not yet added any cards.

2.2 A comparison between deck building games and classical Dominion

There are many similarities between deck building games as defined in Definition 2.1 and classical Dominion. There are, however, also many differences.

For example, we allow TC and VC to be infinite, whereas in classical Dominion this is not the case for apparent reasons. We also require TC and VC to be disjoint, but there are cards in Dominion that are both treasure and victory cards.

Also Dominion has to be played with 2–4 players, while our definition of deck building games only supports a single player.

But the biggest difference between them is the fact that the number of copies of a single card in a game is finite. When a player buys a card in classical Dominion, he adds it to his deck, as usual. If this causes the maximum number of copies of this card to be reached, then the card can not be bought any more.

The game end conditions in classical Dominion make use of this, since the game is defined to have ended when the multiplicity of a certain number of cards has been decreased to 0.

Another major difference between classical Dominion and deck building games, is the notion of a discard pile. In classical Dominion a player has to discard the cards in his hand at the end of his turn, thus forming a deck of discarded cards called his discard pile. Only when a player runs out of cards to draw from his deck, does the player take all of the cards in his discard pile to form a new deck. Because of this, a player can use this information to predict his next hand more precisely, by examining the cards currently in his hand and discard pile. Real players make extensive use of this information.

We decided to define deck building games as we did, because it will allow us to make more powerful statements more easily.

2.3 Extensions for game functions

Although the functions c, w and p are strictly speaking only defined for single cards, we can intuitively extend the definitions of these functions to incorporate collections of cards. In this section we will formulate these extensions.

Definition 2.6 (Value of sets)**.** Let G be a deck building game. The value of a multiset X of cards from $TC \cup VC$ is defined as the sum of the values of the cards.

$$v(X) = \sum_{x \in X} v(x)$$

Definition 2.7 (Points of sets)**.** Let G be a deck building game. The points of a multiset X of cards from $TC \cup VC$ is defined as the sum of the points of the cards.

$$p(X) = \sum_{x \in X} p(x)$$

Definition 2.8 (Cost of sets)**.** Let G be a deck building game. The cost of a multiset X of cards from $TC \cup VC$ is defined as the sum of the cost of the cards.

$$c(X) = \sum_{x \in X} c(x)$$

2.4 States, actions and strategies

In this section we define states, actions and strategies. The state describes the current state of the game. To uniquely define this, it must encompass the cards in the player's deck, the amount of money currently available and the number of turns played.

Definition 2.9 (State)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game. The state of the game is characterized by the *state* S:

$$S = (D, m, t)$$

where D is a deck of G, t is the number of turns played and m is the amount of money available for this turn, $m \in \{m' \in \mathbb{N} \mid \exists H \in \mathcal{H}_D : v(H) = m'\}$.

Also note that there is no such thing as the initial state. This is because the first hand is randomly drawn from the starting deck Q_0 and as such there can be several states that the game could start in. We can only describe when a state could be one of the initial states, as is characterized in the following definition.

Definition 2.10 (Initial state)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game and $S = (D, m, t)$ a state of G. We call S an *initial state* if the following conditions are satisfied:

- $D = Q_0$, and
- $t = 0$, and
- $m \in \{m' \in \mathbb{N} \mid \exists H \in \mathcal{H}_{Q_0} : v(H) = m'\}$.

A player has to decide which action to take depending on his current state. These actions correspond to which card the player wants to add to his deck. Therefore each card in TC or VC induces an action. The action a player takes transforms his current state into another.

Notation 2.11 (Passing)**.** When a player chooses to add no card to his deck, he is said to be *passing*. Slightly abusing notation, this action is denoted with $\emptyset$.

Definition 2.12 (Action)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game, $S = (D, m, t)$ be a state and $x \in TC \cup VC \cup \{\emptyset\}$. The *action* induced by x is denoted by a. It transforms S with transition probability $Pr(S, a_x, S')$ into $S' = (D', m', t + 1)$, where

$$D' = \begin{cases} D, & \text{if } a = \emptyset; \\ D \cup \{x\}, & \text{otherwise.} \end{cases}$$

and $m' = v(H')$ with $H' \in \mathcal{H}_{D'}$.

Recall that D is a multiset, so if $x \neq \emptyset$, then $D \cup \{x\} \neq D$, even when $x \in D$. Note that a player can always choose to add no card to his deck. Again slightly abusing notation, we sometimes write a card x where we mean the action associated with it.

When in a given state S, a player might not be able to buy every card because he does not have enough money available. This means he will not be able to perform the actions corresponding to those cards. The set of cards he can buy determines the actions he can perform, collectively called the action set.

Definition 2.13 (Action set)**.** The set of actions that can be performed when in state $S = (D, m, t)$ is called the *action set* and is denoted with $A(S)$. It is defined as

$$A(S) = \{x \in TC \cup VC \mid c(x) \leq m\} \cup \{\emptyset\},$$

where $\emptyset$ denotes the action of passing, as before. The set of actions that can be performed for a hand H is denoted with $A(H)$. It is similarly defined as

$$A(H) = \{x \in TC \cup VC \mid c(x) \leq v(H)\} \cup \{\emptyset\}.$$

Although adding the card to the current deck and increasing the turn counter is a deterministic process, we can not deterministically describe the resulting state of an action because the money available in the next state is determined by randomly drawing from the resulting deck. The chance of ending up in a state S' is described by the transition probability.

Definition 2.14 (Transition probability)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game, $S = (D, m, t)$ and $S' = (D', m', t')$ be states of G and a be an action. We call the chance that the game will be in state S' after performing action a when in state S the *transition probability* $Pr(S, a, S')$. It is defined as:

$$Pr(S, a, S') = \begin{cases} \dfrac{\#\{H' \in \mathcal{H}_{D'} \mid v(H') = m'\}}{\#\{H' \in \mathcal{H}_{D'}\}}, & \text{if } a \in A(S), D' = D \cup \{x\} \text{ and } t' = t + 1; \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, the transition probability of a state into another over the course of several turns is also easily defined.

Definition 2.15 (Transition probability of sequences)**.** Let G be a game and $(S_i)_{i=0}^n$ be a sequence of $(n+1) > 1$ states S_i. Furthermore, let $(a_i)_{i=1}^n$ be a sequence of actions a_i.

The chance that S_0 will be transformed into S_n via the states $S_i, 1 \leq i < n$ and actions $a_i, 1 \leq i \leq n$ is defined by

$$Pr((S_i)_{i=0}^n, (a_i)_{i=0}^n) = \prod_{i=1}^{n} Pr(S_{i-1}, a_i, S_i)$$

The set of all reachable states of a game G is denoted with $\mathcal{S}_G$.

A graphical interpretation of an action a associated with a card x on a state $S = (D, m, t)$ is presented in Figure 1.

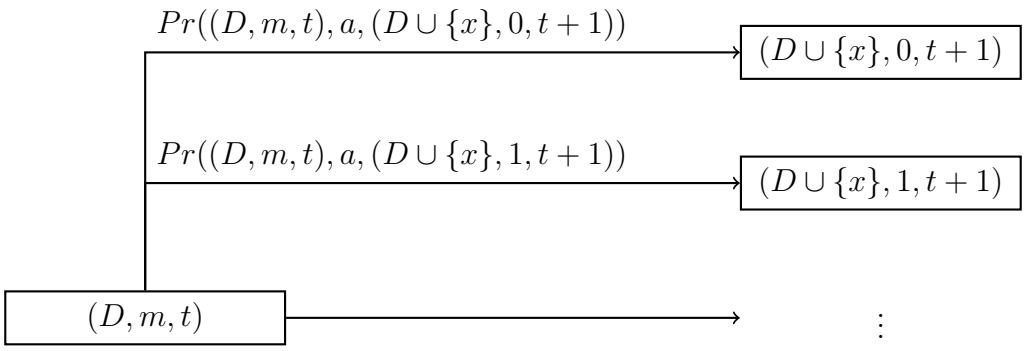


Figure 1: The effect of action a on state $S = (D, m, t)$.

Now that we know how states can be transformed into other states, we can formulate when a state can be reached.

Definition 2.16 (Reachable states)**.** Let G be a game. A state S is said to be *reachable* if

- it is an initial state, or
- there exists a sequence of states $(S_i)_{i=0}^n$ and actions $(a_i)_{i=0}^{n-1}$ where S_0 is an initial state and $S_n = S$ such that $Pr(S_i, a_i, S_{i+1}) > 0, (i = 0, \ldots, n-1)$ holds.

Notation 2.17 (Collection of reachable states)**.** The collection of reachable states is denoted with $\mathcal{S}_G$.

A strategy decides which actions a player should take for any given state.

Definition 2.18 (Strategy)**.** A *strategy* σ of a game G is a function which associates to all states $S \in \mathcal{S}_G$ an action $a = \sigma(S) \in A(S)$.

2.5 Expected score

We can extend the definition of the transition probability to allow for sets of states as destination states. For this we use the notation as described in Notation 2.19.

Notation 2.19 (Transition probability to sets)**.** Let G be a deck building game, $S \in \mathcal{S}_G$ and a an action. We define the transition probability $Pr(S, a, X)$ for a set of states $X \subseteq \mathcal{S}_G$ to be the sum of the transition probabilities of elements of X:

$$Pr(S, a, X) = \sum_{S' \in X} Pr(S, a, S').$$

Using these definitions, we can define the expected score of a state S.

Definition 2.20 (Expected score)**.** Let G be a game, $S = (D, m, t) \in \mathcal{S}_G$ a state and σ a strategy. The *expected score* of S using strategy σ is denoted with $E_\sigma(S)$. It is defined recurrently as

$$E_\sigma(S) = GE(t)p(S) + (1 - GE(t)) \sum_{S' \in \mathcal{S}_G} Pr(S, \sigma(S), S') \cdot E_\sigma(S')$$

3 Reducing calculations

For our experiments we would like to know more about the number of states there are for a given game. In this section we find that it is possible to calculate this number. After that, we introduce the concept of state equivalence. It allows us to reduce the calculations of our algorithm, because equivalent states will have the same expected score.

3.1 Number of states

Since every state contains a deck, we can try to find the size of $\mathcal{S}_G$ by looking at the number of different decks.

Simply generating decks by drawing some cards with replacement from $TC \cup VC$ will lead to many decks being generated more than once. More specifically, when a game G lasts t_{max} turns and writing $z = |TC| + |VC|$, this procedure could create at most $\sum_{i=0}^{t_{max}} z^i = \frac{1-z^{t_{max}+1}}{1-z}$ decks.

However, because first performing action a and then action b leads to the same deck as first performing b and a second, the number of different decks D_t after t turns is in fact a lot smaller. To determine this number, we look at action sequences of length t. One can define

an ordering on the actions such that these sequences can be ordered. Let a, b be two different actions and $\emptyset$ denote the passing action and look at the following sequences:

$$(\emptyset, \emptyset, a, a, a, b, b), \tag{1}$$
$$(\emptyset, a, a, b, b, b, b), \tag{2}$$
$$(\emptyset, \emptyset, a, b, b, a, a), \tag{3}$$
$$(\emptyset, a, b, b, a, a, \emptyset). \tag{4}$$

Now define the ordering $\emptyset > a > b$. This means that sequence 1 and 2 are ordered. However they are different, because they lead to different decks. On the other hand, sequences 1, 3 and 4 lead to the same deck. This can easily be seen by ordering sequences 3 and 4 and concluding that they lead to the same sequence as sequence 1.

Now we can reformulate the question of how many different decks D_t there are after t turns by looking at the number of different ordered sequences of length t. The number D_t can be calculated as

$$D_t = \binom{t + |TC| + |VC|}{t} \tag{5}$$

as proven by [3, p. 38].

However, we do not need to list all states to calculate their expected scores, since some states will have the same scores. Exactly when states will have the same scores is characterized in Secton 3.2.

3.2 State equivalence

States are induced by combining decks with an amount of money and a turn number. Unfortunately, this is not a bijective relation. For example, some values between 0 and M, the maximum amount of money a player can draw in one hand, might be impossible to draw from specific decks. Furthermore, some states might have the exact same future and therefore have the same expected score. This leads to the notion of state equivalence.

Example 3.1. Let $G = (TC, VC, c, v, p, GE, Q_0)$ be some deck building game where $VC = \{\nu_1, \nu_3, \nu_6\}$ and D a deck. Consider the following two decks D_1 and D_2:

$$D_1 = \{\nu_1, \nu_1, \nu_1, \nu_6, \nu_6\},$$
$$D_2 = \{\nu_3, \nu_3, \nu_3, \nu_3, \nu_3\},$$

where $\nu_i \in VC, (i = 1, 3, 6)$ and $p(\nu_i) = i$. Let $t \in \mathbb{N}$ and let S_1 and S_2 be the states $(D \cup D_1, 0, t)$ and $(D \cup D_2, 0, t)$ respectively. Then we know that $p(S_1) = p(S_2)$, since

$$p(S_1) = p(D \cup D_1) = p(D) + p(D_1) = p(D) + p(D_2) = p(D \cup D_2) = p(S_2)$$

holds. Similarly we know that $|D \cup D_1| = |D \cup D_2|$, and $(D \cup D_1) \cap TC = (D \cup D_2) \cap TC$. Taking $a \in A(S_1) = A(S_2)$, $D_1' = D_1 \cup \{a\}$ and $D_2' = D_2 \cup \{a\}$ we can see that

$$\forall m' \in \mathbb{N} : Pr(S_1, a, (D \cup D_1', m', t+1)) = Pr(S_2, a, (D \cup D_2', m', t+1)).$$

This means that, although their decks are different, the states S_1 and S_2 have the exact same transition probabilities. We will later see that this means it is not necessary to calculate the scores of both decks, since they will be the same.

Definition 3.2 (State equivalence)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game and $S_1, S_2 \in \mathcal{S}_G$. We say that $S_1 = (D_1, m_1, t_1)$ is *state equivalent* to $S_2 = (D_2, m_2, t_2)$ if the following conditions are met:

(i) $A(S_1) = A(S_2)$;

(ii) $t_1 = t_2$;

(iii) $p(D_1) = p(D_2)$;

(iv) $|D_1| = |D_2|$;

(v) $D_1 \cap TC = D_2 \cap TC$.

We write $S_1 \sim S_2$ when S_1 and S_2 are equivalent.

Note that $\sim$ is an equivalence relation, since it has the reflexive, symmetric and transitive properties.

As an aside, note how the states $S_1 = (D_1, m, t)$ and $S_2 = (D_2, m, t)$ with D_1 and D_2 as in Example 3.1, are equivalent and are the two equivalent states with the smallest decks in that game.

Conditions (iv) and (v) of state equivalence are chosen such that the chance of ending in state $S_1' = (D_1', m, t+1)$ when starting in S_1 under an action a is the same as when starting from S_2 and ending in $S_2' = (D_2', m, t+1)$ when S_1' and S_2' are also equivalent and the transition probabilities are non-zero. This is formalized in the following lemma.

Lemma 3.3 (Equality of transition probability to states)**.** Let $S_1 = (D_1, m_1, t), S_2 = (D_2, m_2, t)$ be equivalent states, $a \in A(S_1)$, x_a the card associated with a and $m \in \mathbb{N}$. If $S_1' = (D_1 \cup \{x_a\}, m, t+1)$ and $S_2' = (D_2 \cup \{x_a\}, m, t+1)$ are equivalent states, then the following equation holds:

$$Pr(S_1, a, S_1') = Pr(S_2, a, S_2'). \tag{6}$$

Proof The proof of this lemma follows directly from Conditions (iv) and (v) of state equivalence (Definition 3.2):

$$\frac{\#\{H' \in \mathcal{H}_{D_1'} \mid v(H') = m\}}{\#\{H' \in \mathcal{H}_{D_1'}\}} = \frac{\#\{H' \in \mathcal{H}_{D_2'} \mid v(H') = m\}}{\#\{H' \in \mathcal{H}_{D_2'}\}}.$$

□

Note that the non-zero transition probabilities of $Pr(S_i, a, S_i'), (i = 1, 2)$ guarantee that $D_1' = D_1 \cup \{x\}$ and $D_2' = D_2 \cup \{x\}$ when x is the card associated with action a. This is also a necessary condition, as demonstrated by the following example.

Example 3.4. Suppose $S_1 = (D_1, m_1, t)$ and $S_2 = (D_2, m_2, t)$ such that $S_1 \sim S_2$ and $D_1 \neq D_2$. Next take $a = \emptyset$ and $m = v(H)$ for some $H \in \mathcal{H}_{D_1}$. When S_1' is defined as $(D_1, m, t+1)$ and $S_2' = S_1'$, then certainly $S_1' \sim S_2'$, but it is clear that $Pr(S_1, a, S_1') > 0$ whereas $Pr(S_2, a, S_2') = 0$.

In most games, states that are equivalent are pretty common. For example, let G be a deck building game and $S = (D, m, t)$ a state. If $S' = (D, m+1, t)$ has the same action set as S, that is $A(S) = A(S')$, then they are equivalent. However, games with equivalent states that have different decks are no exception either, as is shown by the following lemma.

Lemma 3.5. Let G be a deck building game with $|VC| \geq 3$ and where $\nu_1, \nu_2, \nu_3 \in VC$ are three different victory cards such that $p(\nu_1) < p(\nu_2) < p(\nu_3)$. Then there exist equivalent states S_1, S_2 using ν_1, ν_2 and ν_3 in G with different decks.

Proof Let D_1 be a deck consisting of $p(\nu_3) - p(\nu_1)$ copies of ν_2, and D_2 be a deck consisting of $p(\nu_3) - p(\nu_2)$ copies of ν_1 and $p(\nu_2) - p(\nu_1)$ copies of ν_3. Then we know that D_1 has $p(D_1) = (p(\nu_3) - p(\nu_1))p(\nu_2)$ points, the same as D_2 as is shown by:

$$p(D_2) = (p(\nu_3) - p(\nu_2))p(\nu_1) + (p(\nu_2) - p(\nu_1))p(\nu_3) = (p(\nu_3) - p(\nu_1))p(\nu_2) = p(D_1).$$

This means that they satisfy conditions (iii), (iv) and (v) and therefore the states $S_1 = (D_1, 0, t)$ and $S_2 = (D_2, 0, t)$ must be equivalent, for all values of t. □

Example 3.6. Let G be the game from Example 3.1 and $S' = (D', m', t') \in \mathcal{S}_G$ a state with $D' = D_1 \cup X$, for some deck X. Then we know that

$$p(D') = p(D_1 \cup X) = p(D_1) + p(X)$$

holds. From Example 3.1 we also know that $p(D_1) = p(D_2)$, and therefore also $p(D_2 \cup X) = p(D')$ holds. Then the equivalence class $[S']$ of S' contains at least 2 different states, namely S' itself and $(D_2 \cup X, m', t')$. In fact, if $D_1 \subseteq X$ also holds, then $[S']$ contains at least 3 different states: S' itself, $(D_2 \cup X, m', t')$ and $(D_2 \cup D_2 \cup (X \setminus D_1), m', t')$.

A strategy that performs the same on equivalent states is called an equivalence preserving strategy.

Definition 3.7 (Equivalence preserving strategies)**.** Let G be a game and σ be a strategy. We call σ to be *equivalence preserving* if $\forall S_1, S_2 \in \mathcal{S}_G$ where $S_1 \sim S_2$, the action chosen in these states is the same, that is $\sigma(S_1) = \sigma(S_2)$.

The notion of state equivalence is interesting, because it gives us a way to quickly determine the payout of all decks in the same equivalence class when an equivalence preserving strategy is used.

3.3 Scoring of equivalent states

A similar statement to Lemma 3.3 is also true when looking at an entire equivalence class.

Lemma 3.8 (Equality of transition probability to classes)**.** Let S_1 and S_2 be equivalent states of a game G and let $a \in A(S_1)$. Define $S_1' = (D_1', m_1', t_1')$ and $S_2' = (D_2', m_2', t_2')$ to be equivalent states with non-zero transition probabilities $Pr(S_i, a, S_i'), (i = 1, 2)$. Use $[S]$ to denote the equivalence class of S_1' and S_2'. Then the equality

$$Pr(S_1, a, [S]) = Pr(S_2, a, [S]) \tag{7}$$

holds.

Proof If the decks of S_1 and S_2 are the same, this lemma is trivial. Therefore assume that the decks D_1 and D_2 of S_1 and S_2 respectively are different from each other. Then D_1' and D_2' are also different, since $Pr(S_i, a, S_i') > 0, (i = 1, 2)$ is given.

We can then separate $[S]$ into three disjoint parts:

$$[S] = [S]_{D'_1} \cup [S]_{D'_2} \cup [S]_{D'},$$

where

$$\begin{aligned}
[S]_{D'_1} &= \{S = (D, m, t) \in [S] \mid D = D'_1\},\\
[S]_{D'_2} &= \{S = (D, m, t) \in [S] \mid D = D'_2\},\\
[S]_{D'} &= \{S = (D, m, t) \in [S] \mid D \neq D'_1, D'_2\}.
\end{aligned}$$

Note that $\forall S' \in [S]_{D'_2} \cup [S]_{D'} : Pr(S_1, a, S') = 0$ holds, since the decks of S' and S'_1 are different. For the same reason $\forall S' \in [S]_{D'_1} \cup [S]_{D'} : Pr(S_2, a, S') = 0$ also holds.

This means that the following equation holds:

$$\sum_{S' \in [S]} Pr(S_1, a, S') = \sum_{S' \in [S]_{D'_1}} Pr(S_1, a, S'). \tag{8}$$

Next define

$$\mathcal{M}_1 = \{m \in \mathbb{N} \mid \exists H \in \mathcal{H}_{D'_1} : v(H) = m\}.$$

We know that for every $S' = (D', m', t') \in [S]_{D'_1} : m' \in \mathcal{M}_1$ holds. We also know $\forall m' \in \mathcal{M}_1 : \exists! S' = (D', m', t') \in [S]_{D'_1}$, since all the decks of the states in $[S]_{D'_1}$ are the same. This means we can rewrite Equation 8 into the following:

$$\sum_{S' \in [S]_{D'_1}} Pr(S_1, a, S') = \sum_{m \in \mathcal{M}_1} Pr(S_1, a, (D'_1, m, t'_1)). \tag{9}$$

Via analogous steps we know that the right part of Equation 7 equals

$$\sum_{S' \in [S]} Pr(S_2, a, S') = \sum_{m \in \mathcal{M}_2} Pr(S_2, a, (D'_2, m, t'_2)). \tag{10}$$

Define $\mathcal{M}_2$ analogous to $\mathcal{M}_1$ and note that they are equal, since $A(S'_1) = A(S'_2)$ holds. This means we can simply write $\mathcal{M}$ for $\mathcal{M}_1 = \mathcal{M}_2$. Let $m \in \mathcal{M}$ and look at the states (D'_1, m, t'_1) and (D'_2, m, t'_2). Because of Lemma 3.3, the transition probablities into these states when applying a must be the same:

$$Pr(S_1, a, (D'_1, m, t'_1)) = Pr(S_2, a, (D'_2, m, t'_2)).$$

Since this holds for all $m \in \mathcal{M}$, Equations 9 and 10 must be equal:

$$\sum_{m \in \mathcal{M}} Pr(S_1, a, (D'_1, m, t'_1)) = \sum_{m \in \mathcal{M}} Pr(S_2, a, (D'_2, m, t'_2)).$$

And we can conclude that the initial statement in Equation 7 also holds. □

Theorem 3.9 (Expected score of equivalent states for fixed length games)**.** Let G be a game of fixed length and σ be an equivalence preserving strategy. Let $S_1, S_2 \in \mathcal{S}_G$ be equivalent states. Then S_1 and S_2 have the same expected score:

$$E_\sigma(S_1) = E_\sigma(S_2).$$

Proof Let t_{max} be the turn on which G ends. Note that for games of length t_{max}, the definition of the expected score $E_\sigma(S)$ reduces to

$$E_\sigma(S) = \begin{cases} p(S), & \text{if } t = t_{max}; \\ \sum_{S' \in \mathcal{S}_G} Pr(S, \sigma(S), S') \cdot E_\sigma(S'), & \text{otherwise.} \end{cases}$$

We will prove this theorem using induction.

It is clear that the expected score of the equivalent states $S_1 = (D_1, m_1, t_{max})$ and $S_2 = (D_2, m_2, t_{max})$ is the same. This follows from condition (iii) of state equivalence.

Let $T \in [0, t_{max})$ and assume, as the induction hypothesis, that $\forall t \in (T, t_{max}]$ and $\forall S_1 = (D_1, m_1, t), S_2 = (D_2, m_2, t)$ with $S_1 \sim S_2$ the statement $E_\sigma(S_1) = E_\sigma(S_2)$ holds.

Then take $S_1 = (D_1, m_1, T)$ and $S_2 = (D_2, m_2, T)$ equivalent states at turn T. Because σ is equivalence preserving, we know that $\sigma(S_1) = \sigma(S_2)$. Denote this action with a and its associated card with x_a. Because of Lemma 3.3, we know that the chances to reach states $S_1' = (D_1 \cup \{x_a\}, m, T+1)$ and $S_2' = (D_2 \cup \{x_a\}, m, T+1)$ from states S_1 and S_2 respectively when a is applied, are both zero when m is not a suitable value, or both equal to a certain chance $\phi \in (0, 1]$ when it is. Now we can use the induction hypothesis to show that $E_\sigma(S_1') = E_\sigma(S_2')$, since S_1' and S_2' are equivalent. Combining these statements gives

$$Pr(S_1, a, S_1') \cdot E_\sigma(S_1') = Pr(S_2, a, S_2') \cdot E_\sigma(S_2').$$

When summing over all possible values of m, we get

$$\sum_{S_1' \in \mathcal{S}_G} Pr(S_1, \sigma(S_1), S_1') \cdot E_\sigma(S_1') = \sum_{S_2' \in \mathcal{S}_G} Pr(S_2, \sigma(S_2), S_2') \cdot E_\sigma(S_2').$$

□

A similar theorem holds for games with a non-deterministic game end function GE:

Theorem 3.10 (Expected score of equivalent states with non-deterministic game end function)**.** Let G be a game where $GE(t) = (1-\alpha)^t\alpha$ for some value $\alpha \in (0, 1]$ Let σ be an equivalence preserving strategy and $S_1, S_2 \in \mathcal{S}_G$ be equivalent states. Then S_1 and S_2 have the same expected score:

$$E_\sigma(S_1) = E_\sigma(S_2).$$

Proof Let $E_\sigma^k(S)$ denote the expected score of state S under strategie σ when assuming the game ends after at most k turns.

The limited horizon expected score is defined as

$$E_\sigma^k(S) = \begin{cases} p(S), & \text{if } k = 0; \\ GE(t)p(S) + (1 - GE(t)) \sum_{S' \in \mathcal{S}_G} Pr(S, \sigma(S), S') \cdot E_\sigma^{k-1}(S'), & \text{otherwise.} \end{cases}$$

It follows from this definition that $\lim_{k\to\infty} E_\sigma^k(S) = E_\sigma(S)$. The proof therefore is reduced to proving that $E_\sigma^k(S_1) = E_\sigma^k(S_2)$ holds for all $k \in \mathbb{N}$. We prove this using induction, similar to the proof of Theorem 3.9.

First take $k = 0$. It follows from condition (iii) of state equivalence that $E^0_\sigma(S_1) = p(S_1) = p(S_2) = E^0_\sigma(S_2)$ holds.

We write $S_1 = (D_1, m_1, t)$ and $S_2 = (D_2, m_2, t)$. Now take $K \in \mathbb{N}$ and assume as the induction hypothesis that $\forall k < K$ and for all $S, S' \in \mathcal{S}_G$ with $S \sim S'$ that $E^k_\sigma(S) = E^k_\sigma(S')$ holds. Because σ is equivalence preserving, we know that $\sigma(S_1) = \sigma(S_2)$. Denote this action with a and its associated card with x_a. Let $S'_1 = (D_1 \cup \{x_a\}, m, t+1)$ and $S'_2 = (D_2 \cup \{x_a\}, m, t+1)$ be equivalent states.

Because of Lemma 3.3, we know that the chances to reach states S'_1 and S'_2 from states S_1 and S_2 respectively when a is applied, are both zero when m is not a suitable value, or both equal to a certain chance $\phi \in (0, 1]$ when it is. In either case, they are equal.

We can use this and the induction hypothesis to write

$$\sum_{S'_1 \in \mathcal{S}_G} Pr(S_1, \sigma(S_1), S'_1) \cdot E^{K-1}_\sigma(S'_1) = \sum_{S'_2 \in \mathcal{S}_G} Pr(S_2, \sigma(S_2), S'_2) \cdot E^{K-1}_\sigma(S'_2)$$

Because $GE(t)$ is the same for states S_1 and S_2, and $p(S_1)$ equals $p(S_2)$ because of Condition iii, this equality can be extended to

$$\begin{aligned} E^K_\sigma(S_1) &= GE(t)p(S_1) + (1 - GE(t)) \sum_{S'_1 \in \mathcal{S}_G} Pr(S_1, \sigma(S_1), S'_1) \cdot E^{K-1}_\sigma(S'_1) \\ &= GE(t)p(S_1) + (1 - GE(t)) \sum_{S'_2 \in \mathcal{S}_G} Pr(S_2, \sigma(S_2), S'_2) \cdot E^{k-1}_\sigma(S'_2) = E^K_\sigma(S_2) \end{aligned}$$

This proves that $\forall k \in \mathbb{N} : \forall S, S' \in \mathcal{S}_G$ with $S \sim S'$ the statement $E^k_\sigma(S) = E^k_\sigma(S')$ holds.

Consequently, this gives us

$$E_\sigma(S_1) = \lim_{k \to \infty} E^k_\sigma(S_1) = \lim_{k \to \infty} E^k_\sigma(S_2) = E_\sigma(S_2).$$

□

4 Reachability and efficiency

It follows from Section 2, there are many different deck building games. However, when we focus on the strategies, we notice that a lot of games feature cards that will not be used in any strategies or can be replaced by other cards and lead to better strategies. This leads to the notions of reachability or inefficiency respectively.

4.1 Reachability

Suppose a game $G = (TC, VC, c, v, p, GE, Q_0)$ contains a card $x \in (TC \cup VC) \setminus Q_0$ which is so expensive it can never be bought. This means it will not be used in any strategy. Therefore, every strategy σ for the modified game $G' = (TC \setminus \{x\}, VC \setminus \{x\}, c, w, p, GE, Q_0)$ will also be a strategy for G. So when looking at strategies, the games can be considered the same. Such a card x is said to be unreachable.

Definition 4.1 (Reachable)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game and $x \in TC \cup VC$. The card x is called *reachable* if one or more of the following is satisfied

- the card x is part of the starting deck Q_0;
- there exists a hand H of reachable cards with $x \in A(H)$.

If a card is not reachable, it is *unreachable*. A game where all cards are reachable is said to be *reachable*.

Example 4.2. Let G be a deck building game, where $TC = \{1, 2, 3\}$, $VC = \{4\}, Q_0 = \{1, 4, 4, 4, 4, 4\}$, and

$$\begin{array}{llll} c(1) = 1; & c(2) = 5; & c(3) = 16; & c(4) = 16; \\ v(1) = 1; & v(2) = 3; & v(3) = 4; & v(4) = 0; \\ p(1) = 0; & p(2) = 0; & p(3) = 0; & p(4) = 1. \end{array}$$

Because $1 \in Q_0$ and $4 \in Q_0$, they are both reachable. Furthermore, note that $H = (1, 1, 1, 1, 1)$ is a hand which consists of reachable cards. Then $A(H) = \{1, 2\}$, which means that card 2 is also reachable. Now we can define the hand $H' = \{2, 2, 2, 2, 2\}$, which again consists of only reachable cards. Note that this is the hand with the most available money of all the hands consisting of the reachable cards $1, 2$ and 4, and it is not possible to buy card 3 with this hand. We can therefore conclude that card 3 is unreachable and G is also unreachable.

Also note that card 3 and 4 cost the same, but card 3 is unreachable whereas card 4 is not, even though both cards can never be bought.

4.2 Efficiency

Suppose a game $G = (TC, VC, c, v, p, GE, Q_0)$ contains a card $x \in TC \setminus Q_0$ that costs $c(x)$ and has a value of $v(x)$. Also suppose there is a card $y \in TC$ that costs $c(y) \leq c(x)$ with a value of $v(y) \geq v(x)$. This implies that any strategy σ featuring card x will perform worse than a strategy σ' obtained from σ by replacing all additions of x to the deck with additions of y:

$$\sigma'(S) = \begin{cases} y & \text{if } \sigma(S) = x \\ \sigma(S) & \text{otherwise} \end{cases}$$

So when looking at strategies, one can disregard G and only consider games G' without cards like x. Such a card x is said to be inefficient.

Definition 4.3 (Efficient)**.** Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game. Let $x \in TC \cup VC$. The card x is called *efficient* if one of the following statements holds:

- the card $x \in TC \setminus Q_0$ and $\forall y \in TC$ with $c(x) \geq c(y) : v(x) \geq v(y)$; or
- the card $x \in VC \setminus Q_0$ and $\forall y \in VC$ with $c(x) \geq c(y) : p(x) \geq p(y)$.

If x is not efficient, it is called *inefficient*. A game G where all cards are efficient is said to be *efficient*.

From now on, it is assumed that every deck building game is reachable and efficient.

4.3 A new definition for the action set

When games are efficient, one can decrease the number of cards a strategy has to consider.

Lemma 4.4. Let G be an efficient game, $S = (D, m, t) \in \mathcal{S}_G$ and $x, y \in A(S) \cap TC$ two different affordable treasure cards. Note that $c(x) \neq c(y)$. Without loss of generality, we can assume $c(x) < c(y)$. This implies $v(x) < v(y)$, since G is efficient. Then it is never worse to add y than x.

Proof Let H be a hand containing x. Let H' be the same hand, except all copies of x are replaced with copies of y: $q'_x = 0, q'_y = q_x + q_y$. This implies that $v(H) < v(H')$, since $v(x) < v(y)$. Because of this, the action set of H' might be larger than the action set of H. And since it will not be smaller, $A(H) \subseteq A(H')$ holds. So when given the choice between adding x or adding y, it is never worse to add y and it might be better. □

A similar lemma holds for victory cards:

Lemma 4.5. Let G be an efficient game, $S \in \mathcal{S}_G$ and $x, y \in A(S) \cap VC$ two different affordable victory cards. Note that $c(x) \neq c(y)$. Without loss of generality, we can assume $c(x) < c(y)$. This implies $p(x) < p(y)$, since G is efficient. Then it is better to add y than x.

Proof Let D be a deck containing x. Let D' be the same deck, except all copies of x are replaced with copies of y: $q'_x = 0, q'_y = q_x + q_y$. This implies that $p(D) < p(D')$, since $p(x) < p(y)$. Furthermore, replacing all copies of x with copies of y in a hand H will not change the action set of H, since $v(H)$ is not affected by the replacement. The action set $A(H)$ will therefore be the same before and after the replacement. So when given the choice between adding x or adding y, it is always better to add y istead of x. □

This implies that because of efficiency, any strategy only has to consider buying the most expensive treasure card, buying the most expensive victory card or buying nothing.

Theorem 4.6. For every strategy σ in an efficient game G, the best action in every state $\mathcal{S}_G$ is either buying the most expensive affordable treasure or victory card, or doing nothing.

Proof The proof follows from Lemma 4.4 and Lemma 4.5. □

Using Theorem 4.6 we can redefine the action set.

Definition 4.7 (Action set)**.** Let G be a deck building game, $S = (D, m, t) \in \mathcal{S}_G$ and H a hand. The set of actions that can be performed when in state S is called the *action set* and denoted with $A(S)$. It is defined as

$$A(S) = \arg\max_{Tr \in TC}\{c(Tr) \mid c(Tr) \leq m\} \cup \arg\max_{Vi \in VC}\{c(Vi) \mid c(Vi) \leq m\} \cup \{\emptyset\}$$

The set of actions that can be performed of a hand H is denoted with $A(H)$. It is defined as

$$A(H) = \arg\max_{Tr \in TC}\{c(Tr) \mid c(Tr) \leq v(H)\} \cup \arg\max_{Vi \in VC}\{c(Vi) \mid c(Vi) \leq v(H)\} \cup \{\emptyset\}$$

We use this theorem extensively in every strategy that we tested.

5 Strategies and algorithms

In this section we will explain the different strategies we have compared and the algorithms we implemented.

Each playthrough of a deck building game follows the steps described in Algorithm 1. Note that the implementation of the function BUY_CARD depends on the strategy of the player.

Algorithm 1 Game overview

function PLAY_GAME(starting deck Q_0)
    Draw hand $H \in \mathcal{H}_{Q_0}$
    $S \leftarrow (Q_0, v(H), t)$
    while $\neg$GAME_END(S) **do**
        PLAY_TURN(S)
    end while
    return $p(S)$
end function

function PLAY_TURN(state $S = (D, m, t)$)
    $D \leftarrow D \cup$ BUY_CARD(S)
    Draw new hand $H \in \mathcal{H}_D$
    $m \leftarrow v(H)$
    $t \leftarrow t + 1$
end function

function BUY_CARD(state $S = (D, m, t)$)
    $x \leftarrow$ the card associated with action $a = \sigma(S)$
    return x
end function

5.1 Simple strategies

The strategies in this section are called simple strategies because they can also be used in many different games and their implementation is straightforward. We have used the following strategies:

- `Random`: choose randomly to buy nothing, the most expensive affordable treasure card or the most expensive affordable victory card.
- `Greedy`: buy the most expensive, affordable victory card if able, otherwise buy the most expensive affordable treasure card.
- `Expensive`: buy the most expensive affordable card.
- `Picky`: buy the most expensive victory card. If it is not affordable this turn, buy the most expensive affordable treasure card. (This strategy is called `Picky` because it is picky about which victory cards to buy.)

The strategies are compared in Section 6.2.

5.2 Monte Carlo Tree Search

Besides the simple strategies, we have also applied Monte Carlo Tree Search (MCTS) to deck building games. In this section we will explain our implementation of the MCTS algorithm for deck building games.

Monte Carlo Tree Search is well suited for problems dealing with chance and only a handful possible actions in every state. Since there are only 3 different actions in any state (recall Theorem 4.6), we expect it to perform well for deck building games.

Pseudocode of the Monte Carlo Tree Search algorithm is shown in Algorithm 2 and is adapted from [2].

Algorithm 2 Monte Carlo tree search

```
1:  function BUY_CARD
2:      Tr ← most expensive affordable treasure card
3:      Vi ← most expensive affordable victory card
4:      for each action a ∈ {∅, Tr, Vi} do
5:          for iter ← 0 to iter_max do
6:              Determine points obtained when continuing with strategy σ on a(S)
7:          end for
8:          Average points obtained
9:      end for
10:     Perform action with highest average
11: end function
```

5.2.1 Parameters of MCTS

There are several parameters which influence the performance of MCTS.

First of all there is the number of offspring that will be created, used in line 5. We picked a value of 100 for $iter\_max$ as a value combining speed and sample size.

The strategy σ which is used for the children of S is another parameter of MCTS. We used the `Random` strategy for σ.

In line 10 we chose to average the payouts of the offspring, because we want to determine the average payout when performing (close to) optimal actions in line 10. Other options are of course possible.

5.3 Dynamic Programming

Dynamic Programming is a technique well suited for problems where subproblems are revisited many times. Interpreting a state as its own smaller problem instance for the question "What

are the optimal actions starting from this deck?", one can see how DP should be suited for solving deck building games. However, it soon runs into data and time complexity problems, even when using Theorem 3.9 on the scoring of equivalent decks. In this section we will explain how our implementation dealt with them.

Since Dynamic Programming (DP) is a bottom-up algorithm, it does not follow the outline as describe in Algorithm 1. Instead it follows the pseudocode as listed in Algorithm 3.

Algorithm 3 Dynamic Programming

```
1: function DYNAMIC_PROGRAMMING
2:     for turn t ← t_max to 0 do
3:         for each state S at turn t do
4:             DETERMINE_PAYOUT(S)
5:         end for
6:     end for
7:     return opt_payout[(Q_0, m, t)] for all values of m ≤ max{v(H) | H ∈ H_{Q_0}}
8: end function

9: function DETERMINE_PAYOUT(state S = (D, m, t))
10:    if t = t_max then
11:        opt_payout[S] ← p(S)
12:    else
13:        Tr ← most expensive affordable treasure card
14:        Vi ← most expensive affordable victory card
15:        for each action a ∈ {∅, Tr, Vi} do //identifying a card as an action
16:            D' ← D ∪ {a}
17:            payout[a] ← 0
18:            for money m' ← 0 to max{v(H) | H ∈ H_D} do
19:                payout[a] ← payout[a] + opt_payout[(D', m', t+1)] · Pr(S, a, (D', m', t+1))
20:            end for
21:        end for
22:        a* ← arg max(payout[a])
23:        opt_payout[S] ← payout[a*]
24:    end if
25: end function
```

5.3.1 Explanation of pseudocode

The output of the algorithm is the average score obtained when performing optimal actions. This value is calculated in a bottom-up fashion, starting by examining the final turn. Recall that the game end function GE is part of the definition of a game. To be able to apply Dynamical Programming, the game should end after a known, fixed number of turns.

The algorithm starts by examining every possible deck of the final turn where $t = t_{max}$. Because we know how many turns have been played, we can list all possible decks. And since this is the last turn, no more cards can be added. Therefore the payout of these states is simply the

sum of the victory cards in the deck. This value is calculated in line 11 and stored for future iterations.

In the next iteration, the algorithm examines every states at turn $t = t_{max} - 1$. For these states we have to determine which action will lead to the highest payout. To do this, we try each action in line 18. We know what deck will be the result of every action, however we do not know beforehand which hand we will draw. Because of Theorem 3.9 (Expected scores of equivalent states), this is also not necessary. It is sufficient to calculate the score of a single state, This allows us to sum over all possible values of m in line 19, as long as we calculate the probabilities of drawing that amount of money and not a single hand. This transition probability is calculated by the function $\Pr(S, a, (D', m', t+1))$ and then multiplied by the previously calculated optimal payout of the state $(D', m', t+1)$.

This formula is known as the recursive formula. It forms the core of the Dynamic Programming algorithm because it tells us how to determine the payout of the current state based on the payouts of future states. The payout, or utility value U of a state S at turn t is defined as follows:

$$U(S) = \max_{a \in A(S)} \left\{ \sum_{S' \in \mathcal{S}_G} U(S') \cdot Pr(S, a, S') \right\} \tag{11}$$

where

$$U((D, m, t_{max})) = p(D). \tag{12}$$

Recall that the transition probability $Pr(S, a, S')$ is defined in Definition 2.14. Equation 11 corresponds to lines 18–21 of the pseudocode in Algorithm 3, whereas Equation 12 corresponds to line 11.

By using the recursive formula, we can calculate the expected payout for an action when in a specific state. If we do this for every action, the optimal expected payout can be determined for states at turn $t = t_{max} - 1$.

Repeating these steps until the optimal expected payout for $t = 0$ is calculated, we can calculate the optimal average payout for the starting deck Q_0. This value can be compared to results from other algorithms.

5.3.2 Space complexity of DP

In this section we will take a closer look at the memory usage of Dynamic Programming.

For each state we want to store its payout. This payout is usually a non-integer number, since it is the product involving chances between 0 and 1. However, to keep the space complexity down, we decided to store the payout as an integer number by multiplying it by a constant. To fully utilize the complete range of integer numbers normally stored in 16 bits, we calculated the maximum number of points z a player could score and multiplied the (fractional) payouts with a constant $d = 2^{16}/z$. We used rounding to avoid cutoff errors.

We also reasoned that a state at turn t can have at most t copies of a single card in its deck beyond the starting deck Q_0. But as for alle states $S = (D, m, t)$ we know that $Q_0 \subseteq D$, we do not have to store Q_0 with the description of a state as long as we don't forget it during calculations. The number of copies of a specific card in any state can thus be described by

$\lceil \log_2(t_{max}) \rceil$ bits. As DP visits all states, we knew our implementation had to be able to index these states. We decided to concatenate the bits describing the number of copies of a card with the number of point cards, as well as the number of points and the amount of money available into a single 16-bit or 32-bit long bitstring, depending on the value of $\lceil \log_2(t_{max}) \rceil$. We did not need to incorporate the turn number in this string, because that information could be extracted from elsewhere.

This way our algorithm requires around 16GB of RAM when $t_{max} = 31$.

5.3.3 Time complexity

The time complexity of Dynamic Programming is characterized by the number of turns being played and the number of card types in the game. Our algorithm is of time complexity $\mathcal{O}(t^{|TC|+|VC|})$. This is shown by the following lemma.

Lemma 5.1. Let $G = (TC, VC, c, v, p, GE, Q_0)$ be a deck building game of fixed length t. The time complexity of Dynamic Programming on G is $\mathcal{O}(t^{1+|TC|+|VC|})$.

Proof We know from Equation 5 that there are $\binom{t+|TC|+|VC|}{t}$ different decks after t turns. This can be rewritten into

$$\begin{aligned} D_t &= \frac{(t+|TC|+|VC|)!}{t!(|TC|+|VC|)!} = \frac{1}{(|TC|+|VC|)!}\frac{(t+|TC|+|VC|)!}{t!} \\ &= \frac{1}{(|TC|+|VC|)!}\prod_{i=1}^{|TC|+|VC|}(t+i) = \frac{1}{(|TC|+|VC|)!}\cdot\mathcal{O}(t^{|TC|+|VC|}) \\ &= \mathcal{O}(t^{|TC|+|VC|}) \end{aligned}$$

If we are required to calculate the score for every deck, then this would be our time complexity. In practice, we can do it a bit faster by using state equivalence. The time complexity is therefore at most $\mathcal{O}(t^{|TC|+|VC|})$. □

6 Experiments and results

All experiments were run with values taken from the deck building game Dominion, although of course others could be used. It is defined as follows: $TC = \{1, 2, 3\}$, $VC = \{4, 5, 6\}$, $Q_0 = \{1, 1, 1, 1, 1,\ 1, 1, 4, 4, 4\}$, and

$$\begin{array}{lll} c(1) = 0; & c(2) = 3; & c(3) = 6; \\ c(4) = 2; & c(5) = 5; & c(6) = 8; \\ v(1) = 1; & v(2) = 2; & v(3) = 3; \\ v(4) = 0; & v(5) = 0; & v(6) = 0; \\ p(1) = 0; & p(2) = 0; & p(3) = 0; \\ p(4) = 1; & p(5) = 3; & p(6) = 6. \end{array}$$

The game end function GE varied on the type of experiment.

Note that this version of Dominion is both reachable and efficient.

6.1 Types of experiments

We have conducted several kinds of experiments. Note that the results of all these experiments were obtained from simulations. Except for DP, they are all subject to outcomes from random variables when drawing cards, and as such slightly different results can be expected in a future execution. However, because we have a sample size of 100 and average the results of all executions of a single algorithm, we feel these results give an acceptable idea of the actual values.

The kinds of experiments we have perforemd, is detailed in the following sections.

6.1.1 Probabilistic game end

First we ran the techniques listed in Section 5 using a non-deterministic ending function. After every turn of play the game could end with a chance α. We varied α between $\frac{1}{1}$ and $\frac{1}{100}$ and ran 100 iterations of each algorithm for each α.

We chose this kind of ending function because in traditional deck building games the end of the game is influenced by the actions of other players. Because our model only contains a single player, we tried to simulate this behaviour by randomly deciding if the game will end after every turn.

6.1.2 Fixed game length

We also ran tests using a fixed game length. The length of the game was information available to all strategies, although not all strategies actually make use of this, even though they could.

We varied the game length between 1 turn and 31 turns for DP and between 1 turn and 100 turns for other strategies.

6.1.3 Action ratios

We were also interested in the ratio of actions being performed for a given turn in a given game. Knowing these numbers could show us that for short games, it is a good idea to buy points every turn, but if a game lasts longer, buying points should be avoided early in the game. We were also interested in the number of times and in which situations the passing actions would be the best action, if it would be used at all.

6.1.4 Time complexity

We also ran tests where the execution time was part of the output, to test the time complexity of the executed algorithms.

6.2 Results

In this section we describe the results of each experiment.

6.2.1 Results of probabilistic experiments

The results of the experiments with a probabilistic game end can be found in Figure 2. We can clearly see that the `Random` and `Greedy` strategies are significantly worse than the other strategies. However, because some games in this experiment might have lasted twice as long as others as a result of the probabilistic game end, it is hard to say anything more from these results. This is different for the experiments with a deterministic game end.

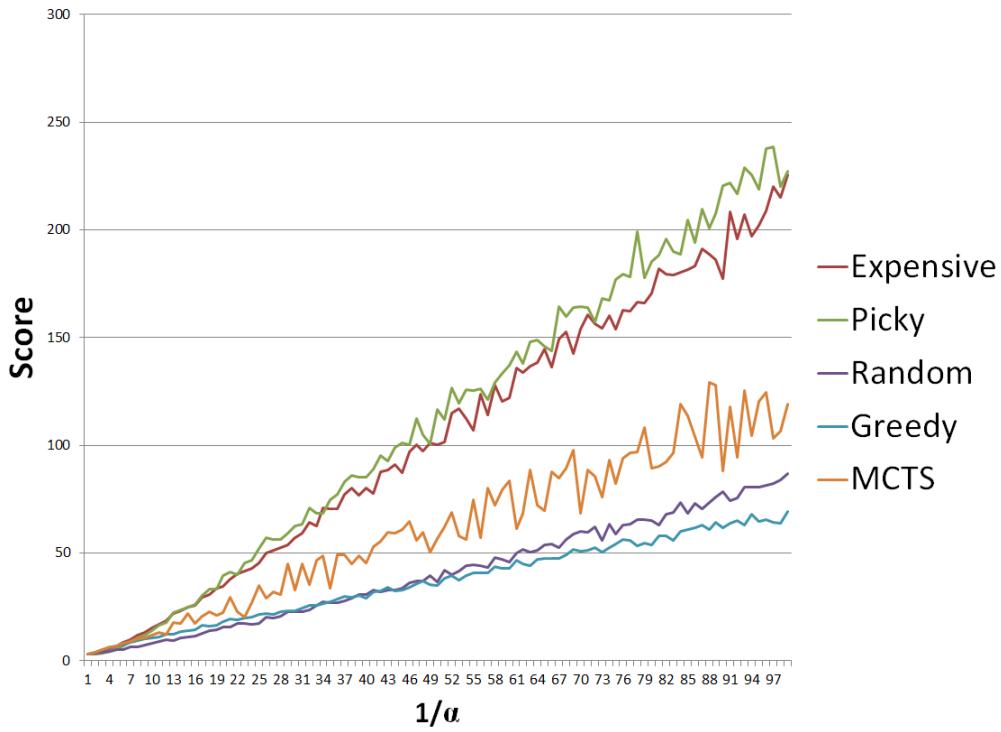


Figure 2: Average final scores of 100 executions per algorithm using a probabilistic game end of α.

6.2.2 Results of deterministic experiments

Figure 3 shows the results of the experiments with a deterministic game length. Because all the games have the same number of turns, the distribution of points is a lot narrower. Now we can clearly see some trends, especially for the shorter games. A zoomed in version of this image is pictured in Figure 4.

First of all, we immediately see that DP is the best performing strategy. This is expected, since it is clear that DP will attain an optimal solution of a problem [1].

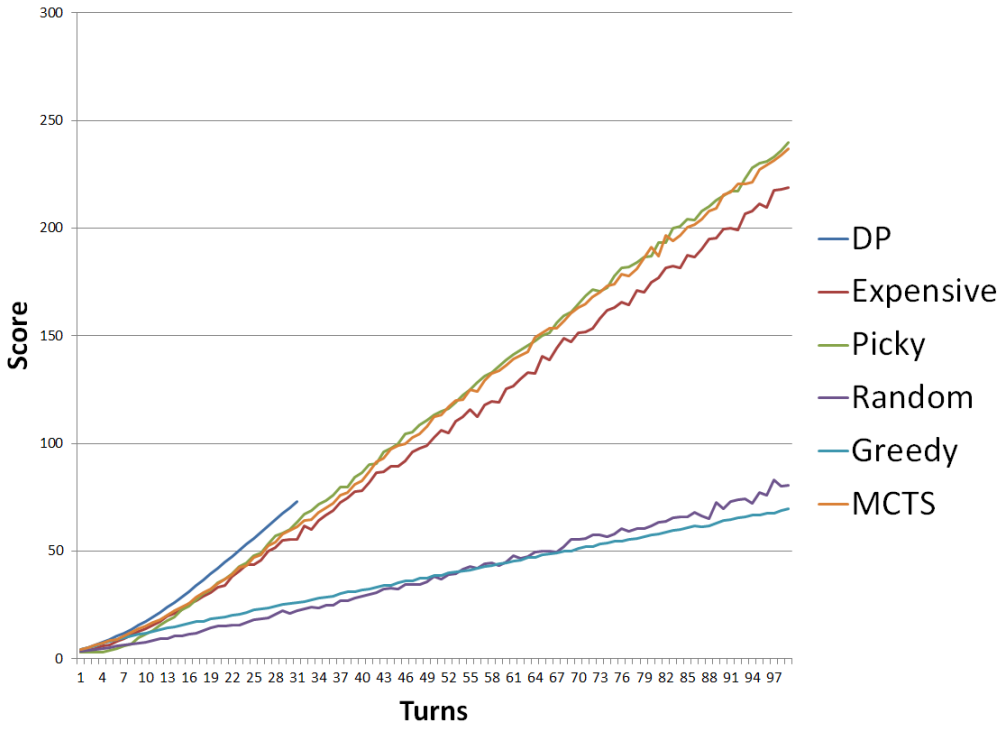


Figure 3: Average final scores of 100 executions per algorithm using a predetermined game length between 1 and 99.

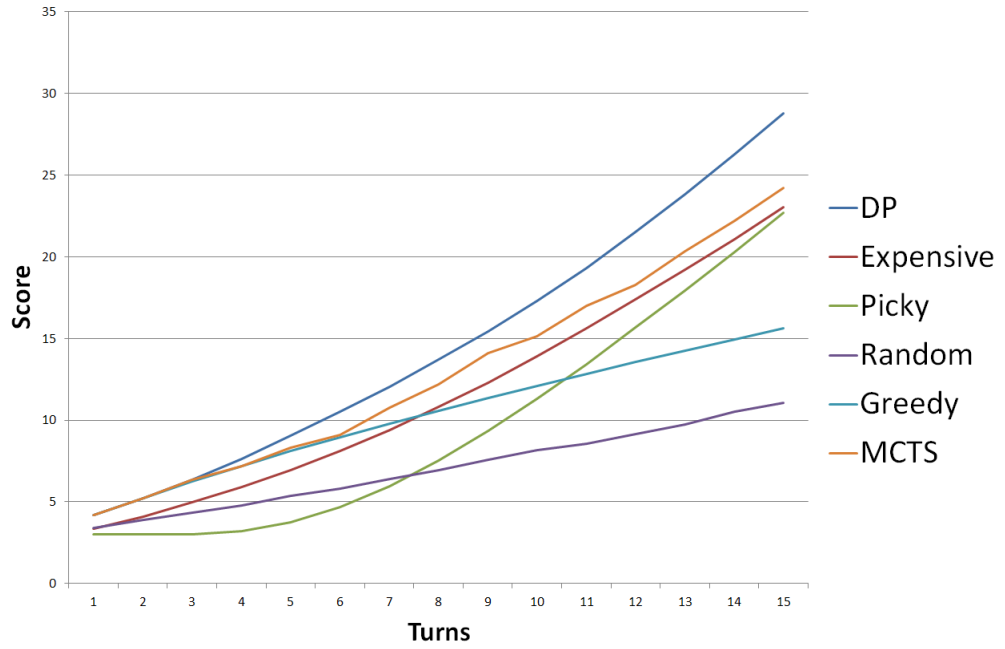


Figure 4: Average final scores of 100 executions per algorithm using a predetermined game length between 1 and 15.

Apparently this involves following a `Greedy` strategy for very short games (lasting up to 4 turns), since the scores for `Greedy` and DP coincide. After that, however, `Greedy` seems to slow down, while DP continues to climb.

In fact, when we return to the longer games as pictured in Figure 3, we can see that `Greedy` is one of the worst performing strategies. What is even more interesting: calculations have shown that for very long games, 150 turns or longer, `Greedy` even falls behind the `Random` strategy! We think that this is a fundamental property of deck building games. Although one needs victory cards to win the game, if bought too early, they will keep one's score low.

This idea is strengthened by the relationship between `Expensive` and `Picky`. For shorter games, `Expensive` seems to have the upper hand. However, for games longer that 15 turns, `Picky` takes the upper hand. Apparently buying the most efficient victory card is very important for longer games.

In fact, we think it is very surprising to see such a simple strategy as `Expensive` performing so well. We think it could be attributed to the distribution of the costs of the treasure and victory cards in the test game, because the treasure and victory cards alternate when sorted by increasing cost. This could cause the `Expensive` strategy to buy a balanced number of treasure and victory cards. Had the values been different, then `Expensive` might have performed differently.

Another surprise is the performance of MCTS. Even though it bases its actions on continuations that follow the `Random` strategy, it performs a lot better than `Random` itself. However, its scores seems to remain on par with `Picky`, even though there is a lot more computation behind it.

6.2.3 Results of time complexity experiments

Finally, in Figure 5, we can see the time complexity for DP. A trend line can be plotted of $\mathcal{O}(t^6)$ that seems to fit this data series. This indicates that the theoretical time complexity of $\mathcal{O}\left(t^{|TC|+|VC|}\right)$ is met in practice.

However, in the same figure we can see the execution time of MCTS for games of up to 250 turns. Apparently MCTS is several orders of magnitude faster than DP. So when one values execution time over optimality, MCTS seems to be the better algorithm.

However, in this case `Picky` seems to be an even better algorithm, since tests have shown it to be capable of performing 10.000 executions of 250-turn games in 25 seconds, while yielding similar scores as MCTS. Because of this, we have to say that our current implementation of MCTS seems to be unsuitable for deck building games, as apparently far better algorithms are available.

Ultimately it turned out to be possible to substantially accelerate the DP algorithm. Ben Ruijl form Universiteit Leiden reported a runtime of around 20 seconds for games of 30 turns, also allowing for longer games to be played. In the future we want to elaborate on this.

6.2.4 Results of action ratios experiments

In Figures 6, 7 and 8 we can see the action ratios of the strategies DP, MCTS and several others respectively.

At first sight, the action ratios of DP in Figure 6 seem to have a similar distribution, regardless whether the game length is 15, 25 or 31 turns. The biggest difference seems to be the first

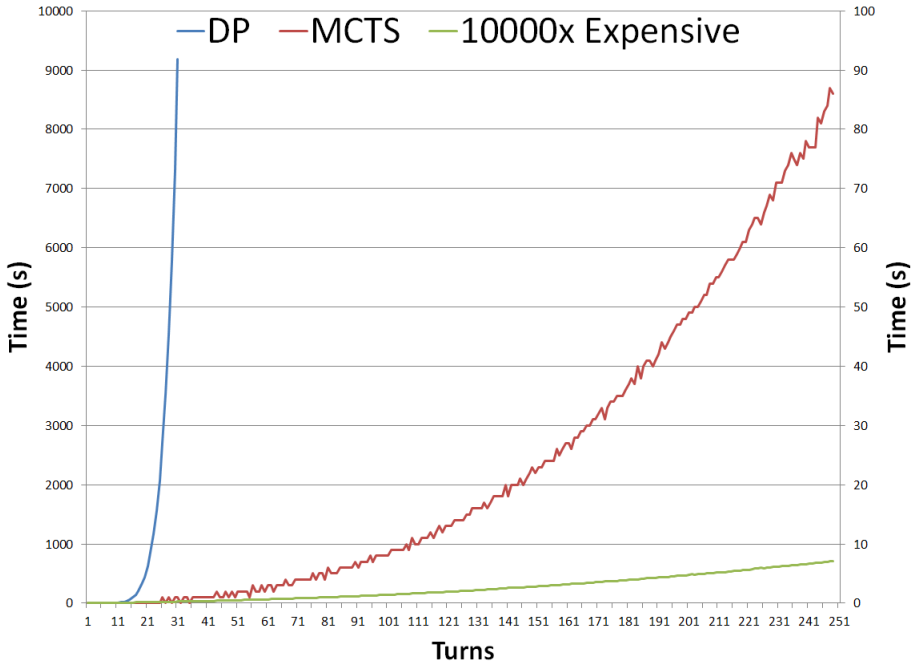


Figure 5: Execution time of DP, MCTS and 10.000 runs of `Picky` in seconds for games between 1 and 250 turns.

turn on which a victory card is ever bought. They seem to be bought later as the game length increases.

Also note that there is a blue section at every turn in all three figures. This indicates that it is optimal in some states to pass and passing seems to be an integral part of the game.

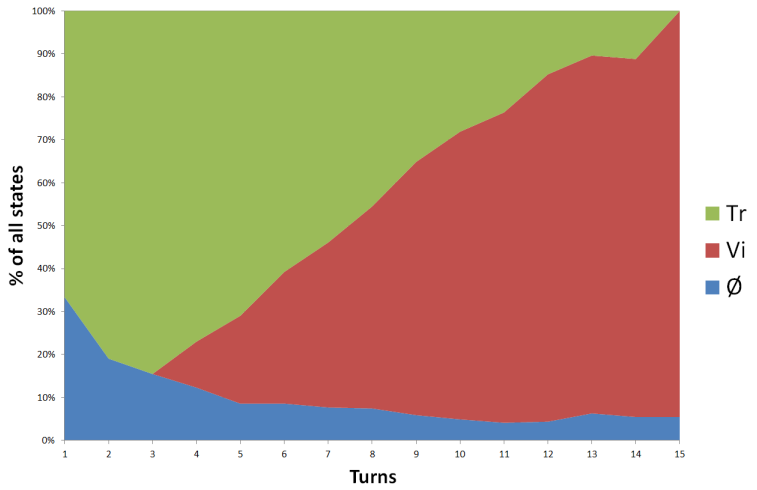
Now compare Figures 6 and 7. Just as the action ratios of DP, the action ratios of MCTS are quite similar regardless of game length. Furthermore, one could say that the action ratios of DP and MCTS look alike.

Unfortunately, this does not mean that good performance corresponds with similar action ratios, as the action ratios of `Picky` indicate in Figure 8b. It shows a completely different pattern of action ratios, when compared to the ratios of DP or MCTS, but still performs similarly. This indicates that action ratios themselves are no good indicators for effective strategies.

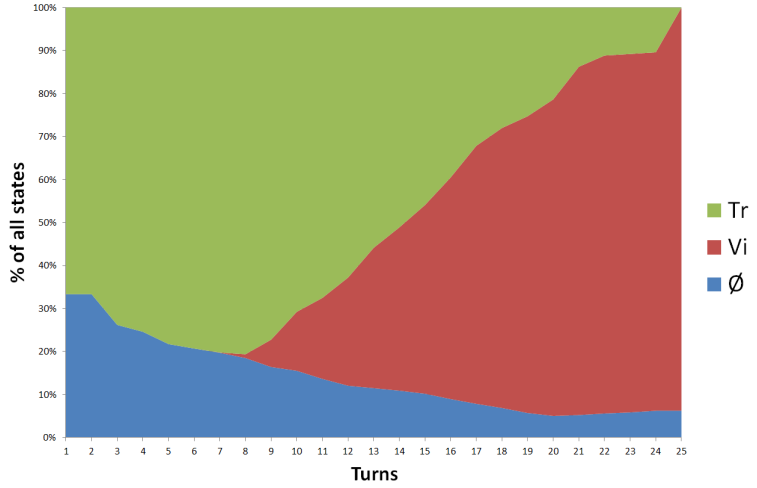
The reason behind the decelerating rise of scores for `Greedy` might be hidden in the action ratios of Figure 8c. One can clearly see that the percentage of states in which `Greedy` is still able to buy victory cards decreases as the length of the game increases.

7 Conclusions and further research

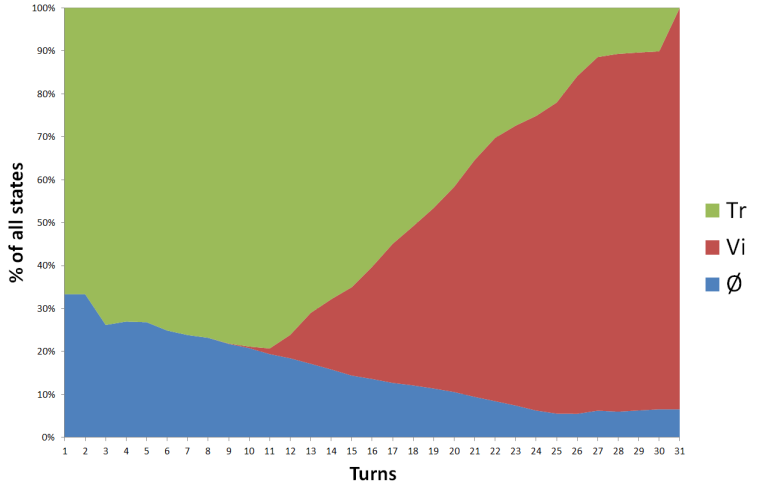
In this paper we defined a framework of definitions to reason about deck building games. We introduced a notion of equivalence of states, detailed necessary and sufficient conditions for equivalent states to exist and proved that the expected score of such states is the same.



(a)

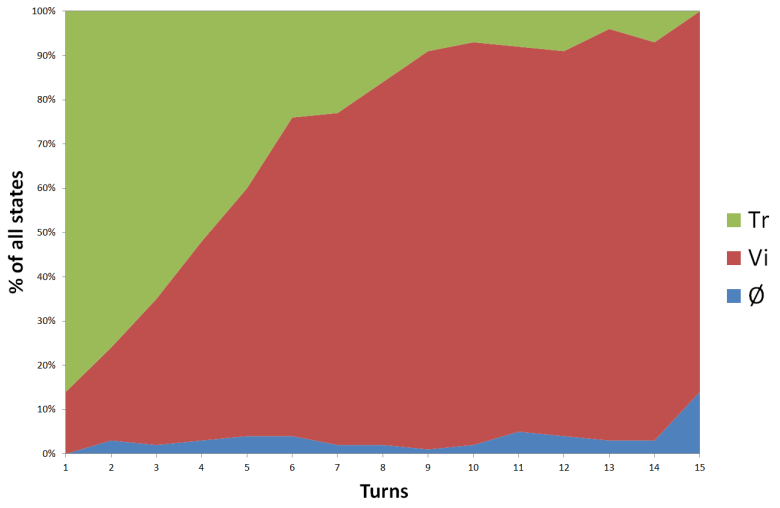


(b)

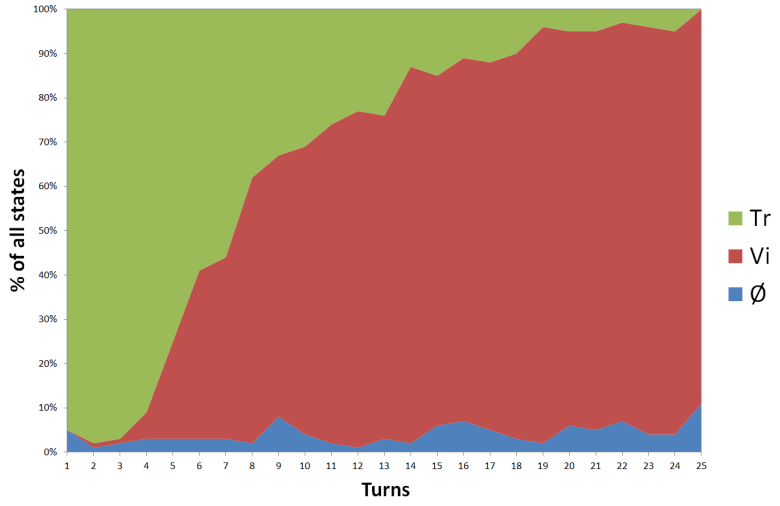


(c)

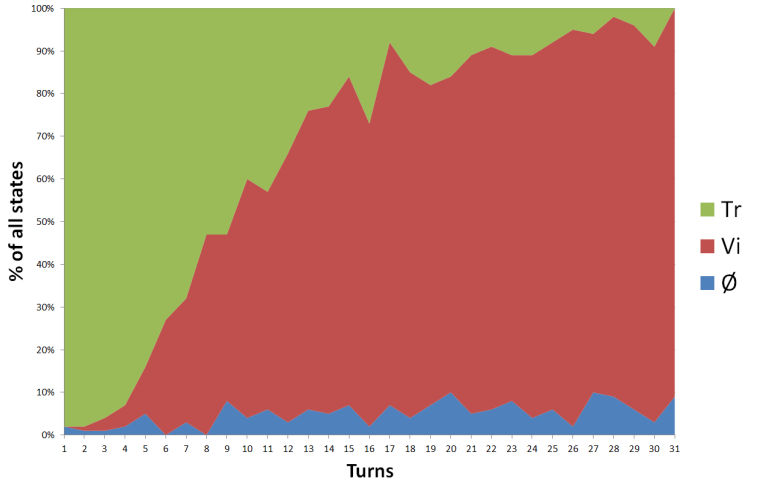
Figure 6: Action ratios of DP for various game lengths, where green signifies buying treasure cards, red victory cards and blue passing. Figures 6a, 6b and 6c picture games of 15, 25 and 31 turns respectively.



(a)

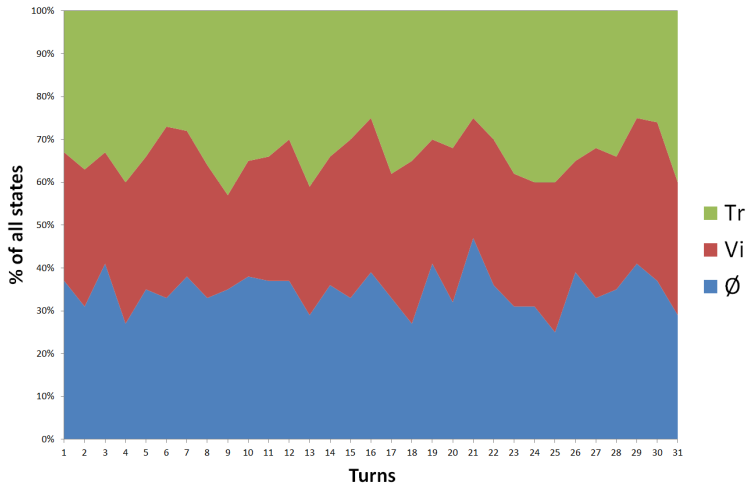


(b)

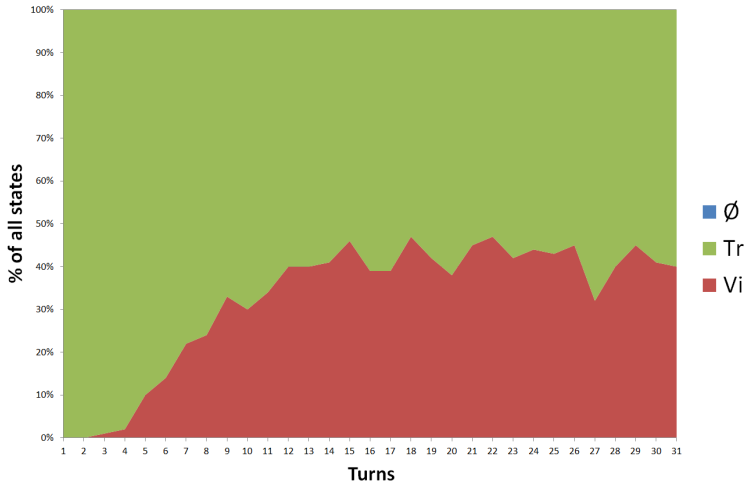


(c)

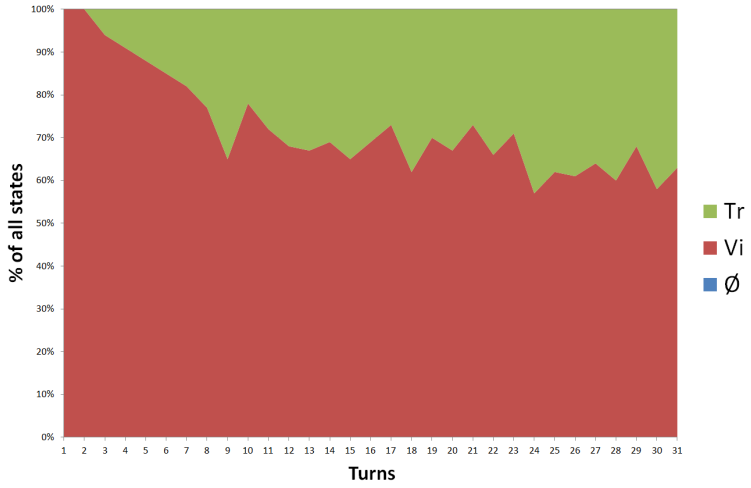
Figure 7: Action ratios of MCTS for various game lengths, where green signifies buying treasure cards, red victory cards and blue passing. Figures 7a, 7b and 7c picture games of 15, 25 and 31 turns respectively.



(a)



(b)



(c)

Figure 8: Action ratios of various strategies for 31-turn games, where green signifies buying treasure cards, red victory cards and blue passing. Figures 8a, 8b and 8c picture executions with the `Random`, `Picky` and `Greedy` strategies respectively.

Furthermore, we defined the concepts of reachability and efficiency to allow us to remove cards

from a game without affecting any interesting strategies.

Next we introduced our implementations of MCTS and DP adapted for deck building games, as well as our series of experiments for them. The most striking results could be the fact that DP is indeed able to find the opmtial action in each state, albeit slowly; the fact that MCTS is surpassed with respect to both execution time and perfomance by strategies specific to deck building games; and the fact that passing is an integral part of deck building games.

Still, a lot of interesting questions remain, and we can discern four major areas for future research.

First one could look into parallelizing the DP algorithm. It is well suited for parallelization because its calculations can be easily divided into sections with no relation to other sections. This will speed up the algorithm greatly and therefore allow for the analysis of longer games.

Besides parallelization, one could look into the data complexity of the DP algorithm. Our current implementation hardly uses Theorem 3.9, but if used properly it might allow us to circumvent most of the data complexity issues we are currently facing.

As mentioned in Section 6.2.3, we also want to further improve the runtime of the computation of exact scores.

And finally, the relation between the model of a deck building game we described in this paper and a real deck building game — such as Dominion — could be examined. When we applied DP to Dominion, we had to make several assumptions such as the fixed game length. It is unclear how different algorithms are affected by this and whether they can be modified to accomodate for this.

References

[1] R.E. Bellman, *The Theory of Dynamic Programming*, Bulletin of the American Mathematical Society, 60 (6), pages 503–516, 1954.

[2] G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, B. Bouzy, *Progressive Strategies for Monte-Carlo Tree Search*, New Mathematics and Natural Computation, 4 (3), pages 343–359, 2008.

[3] W. Feller, *An introduction to Probability Theory and its Applications*, vol. 1, 3rd edition, publisher: Wiley, 1968.

[4] C. Paget, *Pedigree of Deck Builders*. BoardGameGeek, `http://www.boardgamegeek.com/geeklist/69461/pedigree-deck-builders`, 2011.

[5] Rio Grande Games, *Dominion game information*. `http://riograndegames.com/Game/278-Dominion`, 2008.