Universiteit Leiden
The Netherlands

Opleiding Informatica

Determining the Success Rate
for the Game of Solitaire

Rob Reijtenbach

Supervisors:
Walter Kosters & Hendrik Jan Hoogeboom

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

July 13, 2021

Abstract

In this thesis we will take a look at the succes rate for the card game SOLITAIRE. As the total number of SOLITAIRE games that can be played with a normal deck is not feasible to fully examine, we will look at smaller versions of this game.
For these smaller versions of SOLITAIRE we will examine the games that are not winnable, no matter which choices the player makes, and we try to find patterns in these games. The emerging patterns will be compared between multiple versions of the game.
Determining which games are winnable will be done by trying all different choices until no more move is possible or the game is won. This effectively means we play with all cards open as there is no room for guessing when determining which games are winnable.

Contents

1 Introduction **1**

2 Definitions **2**

3 Rules of the Game **3**
3.1 Starting state . . . 3
3.2 Playing rules . . . 4
3.3 End conditions . . . 6
3.4 Variants and choices . . . 6

4 Related Work **7**
4.1 Research into the success rate . . . 7
4.2 More general Solitaire research . . . 7

5 Experiments **8**
5.1 The SOLITAIRE program . . . 8
5.2 Computing the games . . . 8
5.2.1 Stock optimization . . . 8
5.2.2 Suit and color symmetries . . . 9

6 Categorizing fully computed games **10**
6.1 The smallest games . . . 12
6.2 Games having 2 cards per suit and 3 build stacks . . . 12
6.3 Games having 3 cards per suit and 2 build stacks . . . 14
6.4 Games having 3 cards per suit and 3 build stacks . . . 17
6.5 Games having 3 cards per suit and 4 build stacks . . . 19
6.6 Not fully computable games . . . 21

7 Conclusions and Further Research **22**
7.1 Further Research . . . 22

References **23**

1 Introduction

Even though almost everyone will know, or at least have heard of, the card game SOLITAIRE, we will start with a brief introduction. The game KLONDIKE SOLITAIRE (from now on just SOLITAIRE) is a one player card game played with a standard 52-card deck. In this game, the player needs to move all the cards to the four suit stacks. This will sometimes be impossible, sometimes the game will fail due to choices the player made and sometimes the player will manage to get all the cards to their suit stacks in which case he or she will win the game. For those who want to to try this game (or feel nostalgic), there is an online Windows98 emulator [1] which includes the game of SOLITAIRE.

The rules of the game will be further explained in the appropriate section of this paper. For now we will focus on explaining what this research is about. In this paper we will take a look at the win rate for games of SOLITAIRE. These games will be played "perfectly" by traversing all possible moves only stopping if a game is won or there are no moves left. This means we disregard the player choices as a way to lose a game.

As playing all different possible games with a 52-card deck is computationally hard, we will look at different ways to find symmetry or reduce the game size to make it more viable. After fully computing the win rates for these smaller deck sizes we will also compute the win rates for a subset of the possible games for bigger deck sizes.

An example of a game of SOLITAIRE with a reduced size is a game where there are only two build stacks and the deck of cards consists of only 12 cards. Understanding this example requires a basic understanding of the game SOLITAIRE, if the reader lacks this knowledge they should consult Section 3. In a 12-card deck each suit will have three cards. The two build stacks will require three cards for the beginning state of the game (one for the first stack and two for the second stack) and the rest will be placed in the stock. This is an interesting example as there are only two unique permutations of the deck which are not winnable. This does take into account the symmetry which was named earlier and will be further explained in Section 2.

In Figure 1 a visualization of the two non-winnable games is displayed. It is clearly visible how these games are very similar as only changing the ♠2 and ♠3 in one of them results in the other.

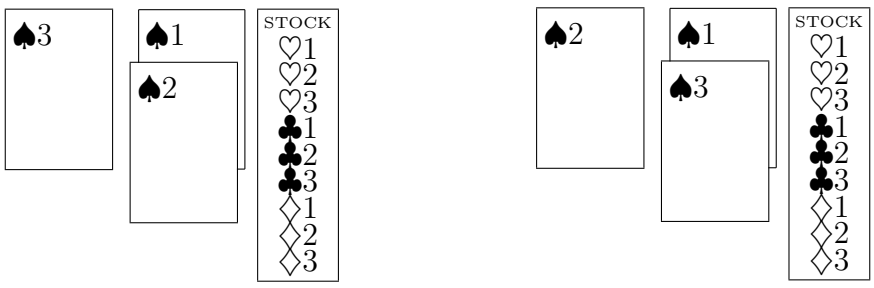

Figure 1: The non-winnable games with a deck of 12 cards and two build stacks

It should be noted that these are not the only two permutations of the deck that are not winnable, as the same setup with any other suit will give the same result. However these are covered by the symmetry and will therefore not be counted.

Many more variants of SOLITAIRE will be covered in this thesis and we will also take a more in depth look with respect to how these games are not winnable. Most of this will be covered in Section 5 and Section 7 where we will take a closer look in the success rates for different variants of SOLITAIRE. The first section after this one will be Section 2 in which the terminology which will be used throughout this thesis will be explained. After that the rules of the game of SOLITAIRE will be explained in Section 3. In Section 4 related work will be examined. The experiments that were conducted for this thesis are explained in Section 5, after which Section 6 and Section 6.6 will provide and explaing the results of the experiments. Finally the conclusions will be given in Section 7.
This thesis is written as a bachelor thesis at the Leiden Institute of Advanced Computer Science (LIACS) and is supervised by Walter Kosters and Hendrik Jan Hoogeboom.

2 Definitions

In this thesis some terms will be used to describe certain characteristics of games of SOLITAIRE. This section will define or explain these terms so the reader will know their meaning for the rest of this thesis.

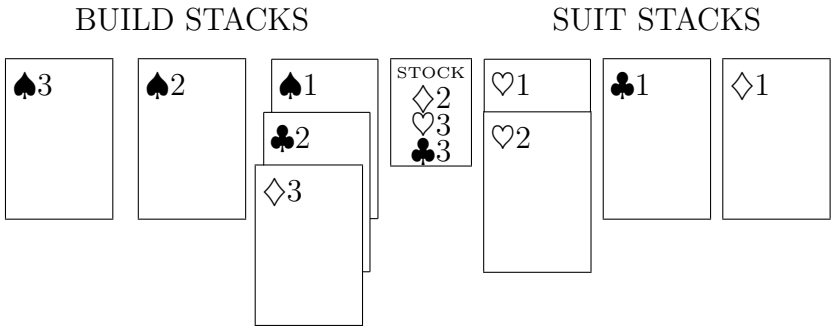

Figure 2: Example of a card game to illustrate the definitions.

- "Suit of a card": Every cards in the deck has one of the four suits. These suits are Spades, Hearts, Clubs and Diamonds, which are represented by the following icons respectively: ♠, ♡, ♣ and ♢.
- "Number of a card": The number of the card is rather, straight forward, the number on the card. Physical playing cards often include the numbers from 2 to 10 where there are other representations for the 1 (ace, later defined) and the numbers 11 to 13. In this thesis we just use the numbers in numerical representation as for SOLITAIRE this suffices and is less ambiguous when the number of cards per suit is variable.
- "Stock": Cards that are left over after filling the build stacks when the game is initialized into its starting state. How the game is initialized is explained in Section 3. The stock is clearly marked in the figures in this thesis. For example: Figure 2 in which the stock holds 3 cards.

- "Build stacks": The stacks in which most of the game is played. These are portrayed to the left of the "stock" in most figures in this thesis. For example: Figure 2.
- "Suit stacks": The stacks to which the cards eventualy have to be played in order to get to the winning state of a game. Also called ace stacks as the ace of a suit is always the first card to be placed there. There are always 4 suit stacks, one for every suit. These are portrayed to the right of the "stock" in the figures in which they are included in this thesis. For example: Figure 2.
- "Width": The number of build stacks in a game. In Figure 2 the game has 3 build stacks therefore the with is also 3.
- "Top/Bottom card of a stack of cards": The top card is defined as being the card last placed on said stack while the bottom card of that stack is the oldest card with the most cards placed on it for a certain stack. This seems rather straight forward, however with the way cards are placed in a game of Solitaire the opposite of these terms could also be considered logical.
- "Open/Closed cards": Open cards are cards facing up, closed cards are facing down.
- "King card": Kings are the highest card for each suit. When a deck is defined as having fewer than 52 cards, meaning the individual suits have fewer than the usual 13 cards, then the highest number becomes the king.
- "Ace card": Aces are the cards with the lowest number for each suit. This is always the number 1.
- "Winnable": A game in which the player can reach the winning state. In a winnable game the player might still be able to make the wrong choices resulting in a loss.

3 Rules of the Game

Solitaire is played with a standard deck of cards. This deck consists of four suits: spades, hearts, clubs and diamonds. These suits are usually depicted using the following symbols: ♠ for spades, ♡ for hearts, ♣ for clubs and ♢ for diamonds. The suits heart and diamonds are colored red while the suits spades and clubs are colored black.

3.1 Starting state

In the begin state of a game of Solitaire the deck is randomized by shuffling the cards. After this the cards will be distributed across the seven build stacks and the remainder is put on the stock. The way the cards are being distributed across the build stacks is in a left to right increasing order, where one card will be placed on the first or leftmost build stack, two cards will be placed on the second build stack and finally seven cards will be placed on the seventh or rightmost buildstack. Every card is placed on the build stack face down except for the top card which is flipped face up. The remainder of the cards is placed face up on the stock and the four suit stacks all start empty.

Using a standard 52-card deck and seven build stacks this results in $\sum_{i=1}^{7} i = 28$ cards are distributed across the build stacks and $52 - \sum_{i=1}^{7} i = 52 - 28 = 24$ cards placed in the stock. A more general way to describe the number of cards placed on the build stacks and the number of cards placed on the stock would be $\sum_{i=1}^{w} i = w(w+1)/2$ and $c \cdot 4 - \sum_{i=1}^{w} i = c \cdot 4 - w(w+1)/2$ respectively. Here w is the *width* of the *board* and c is the number of cards each suit has.

3.2 Playing rules

As long as the game has not ended the player is allowed to do certain moves. A move consists of moving one or more cards following a set of rules. The moves can be split into four categories which each have their own rules which will be explained in the next subsections. However, we will first take a look at the rules that apply for every move.
There are a couple of rules that apply to every move, the first of which is that only open cards can be moved. Another rule that always applies is that no card that has been played to one of the four ace stacks is allowed to be moved after being placed there. This also followes from the fact that none of the four categories describe a move like that. Not completely unlike the last rule, it is also not allowed to move a card back to the stock after it was placed. This also follows from how there is no category for a move like that.
An overview of the way cards are allowed to move is presented in Figure 3. The nodes represent the different places in a game and the edges represent the ways cards can move. The numbers next to the edges are to indicate which category of moves the edge represents. Some different variants of Solitaire allow different moves, more on that in Section 3.4.

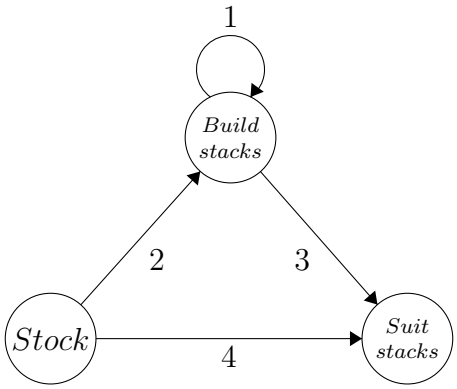

Figure 3: Visualisation of how cards are allowed to move.

Build Stack to Build Stack

The first category is playing a card from one of the build stacks to another build stack. This is allowed if the color of the card being moved is opposite to the color of the card on which it is placed and the number of the card is equal to number of the one that is being placed on minus one. This move can not only be done by a single card but also by multiple cards which already form a stack of opened cards with decreasing numbers and alternating colors. In this case a whole (sub)stack can be moved to a different build stack as long as the bottom card of this (sub)stack is

of the opposite color and has a number one lower than the top card of the stack it is being placed on. If one of the build stacks is empty then only a king can be placed on this stack. Note that cards can never be placed on the stock.

Stock to Build Stack

The second of these four categories is a move where a card from the stock is being played to one of the build stacks. The rules for a move of this kind are similar to but not exactly the same as the rules for moves in the first category as the requirements for where a card can be played are the same but which cards can be played differs. When playing a card from the stock to one of the build stacks every card from the stock can be chosen (as this is the Draw One Solitaire). The card which is chosen freely from the stock must to conform to the rule of being of the opposite color and having a number which is one lower than the top card of the stack it is placed on (similar to the first category). The rule mandating that only a king can be placed on an empty build stack also holds here.

Build Stack to Suit Stack

The third category of moves is playing a card from one of the build stacks to the appropriate suit stack. When playing a card to one of the suit stacks, this card has to either be an ace of one of the suits or has to be placed on a card of the same color with a number which is one lower than the card being placed. An ace can be placed on either one of the four suit stacks as long as the receiving suit stack is empty whereas the cards with higher number can only be placed on the suit stack where their ace (and possibly other cards) are already placed. The card being played always has to be the top card of the build stack it comes from.

Stock to Ace Stack

Finally the fourth category of moves describes moves where a card is moved from the stock directly towards one of the suit stacks. The rules for this category are like the rules for the third category. The main difference for playing cards from the stock is that the card can be chosen freely from the stock instead of having to use the top card when playing from a build stack. The other rules from the third category still apply, for example the rule with regards to stacking the cards on the build stacks.

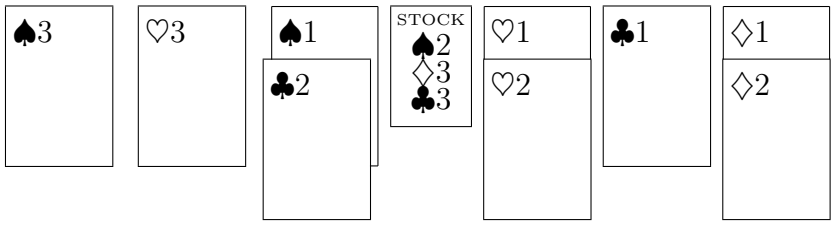

Figure 4: A game state where at least one move out of every category is possible.

3.3 End conditions

When a game is in a state where there is no possible move the game has ended.[1] There are multiple ways a game can end, either resulting in a win or a loss. A loss occurs when the game ends while there are still cards on one or more of the build stacks or in the stock. A visual representation of a game that has resulted in a loss is presented in Figure 5.

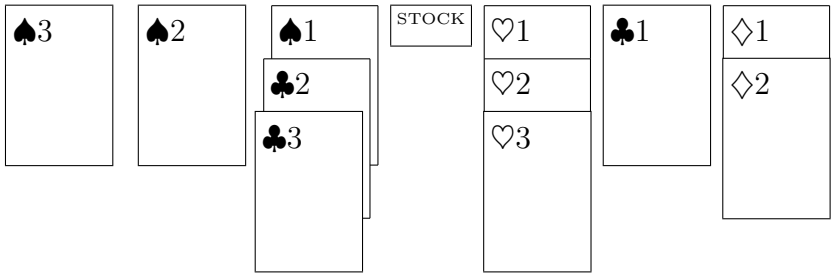

Figure 5: A game that has ended but still has cards in its build stacks. The suit stacks are presented to the right of the empty stock.

A game is won when all cards are played to their suit stacks in which case there also is no possible move left meaning the game ends aswell. Even though this might be clear from the explanation already, a game that has reached the winning game state is displayed in Figure 6.

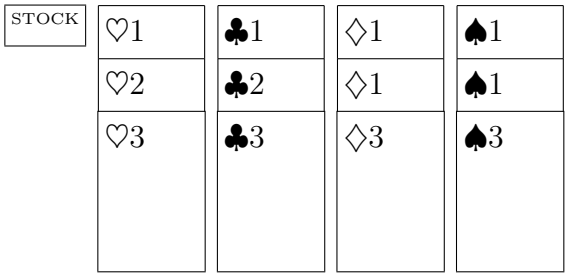

Figure 6: A game that has ended in the winning state. The suit stacks are again presented to the right of the stock.

3.4 Variants and choices

The rules of Solitaire are not the same for everyone, as with a single player card game that is mainly played for fun nothing keeps the player from making their own rules. There is a variant of Solitaire called Easthaven where the begin state of the game is slightly different as a different way to fill the build stacks is used. There is also a variant of Solitaire which follows

[1] The game also ends if the only possible moves create loops to previous states. This disallowes infinite games.

the KLONDIKE beginning (as used in this thesis) but where only every third card of the stock is allowed to be played. This version is commonly referred to as KLONDIKE BY THREES or DRAW THREE KLONDIKE. It is also possible to play one of the mentioned versions with multiple decks. This results in eight suit stacks (when playing with two decks) and usually more build stacks to facilitate the higher number of cards. Some people also allow cards being placed on one of the suit stacks to be placed back on one of the build stacks as this can in some cases be useful.
In this thesis we will only look at DRAW ONE KLONDIKE SOLITAIRE. This follows the rules laid out earlier. As also stated earlier we do not allow cards placed on one of the suit stacks to be placed back on one of the build stacks. This is however not more than a choice as the other versions of SOLITAIRE are just as valid as this one.
We will, however, take a look at games with varying deck sizes. This means having fewer than the usual 13 cards per suit. As said before the highest card will then take the role of king. The "width" of the game will also be variable. Recall how width is used to describe the number of build stacks in a game. The workings of the stock and the suit stacks will stay the same.

4 Related Work

SOLITAIRE is a very old game. According to Wikipedia [2] the earliest known recording of a game of SOLITAIRE dates back to 1788. Back then it was published in a German game anthology [9, pp. 173–174]. Given that this game is really old it is unsurprising that more research has been dedicated to studying it. In this section some of that research will be mentioned, some also studying the success rate, some more broadly studying the game of SOLITAIRE.[2]

4.1 Research into the success rate

Specific research into the success rate of SOLITAIRE is done by Bjarnason et al. [8, 7] They reported the number of games that are not winnable to be between 8.5 and 18 percent.
A different but similar statistic with regards to games that are not winnable is searching games where not a single move can be done. Donkersteeg reported in his master's thesis [5] that 0.25 percent of all possible games are *unplayable* (which is defined as games where not a single move is possible). It took him approximately one minute on a 3 GHz processor using a brute force approach. De Ruiter [3] confirmed this value using dynamic programming and only taking milliseconds.

4.2 More general Solitaire research

More general research into SOLITAIRE can be very broad. Some examples of this are simulating games with different A.I. players (e.g. random or using heuristics) and reporting the win rates. Also the rules can be varied while simulating these games. An example of such a variation is allowing or not allowing the king cards to be moved to an empty build stack. Kortsmit [4] reports that this reduced the simulated win percentage from 25.70 percent to 0.30 percent.
Xiang Yan et al. [6] specifically looked into THOUGHTFUL SOLITAIRE (where also the face down cards are known to the player in advance). They report that using *expert play*, where a skilled

[2]Most of the other research into Solitaire is done on DRAW THREE SOLITAIRE instead of the DRAW ONE variant that is used in this thesis.

combinatorialist carefully played and recorded 2,000 games, a win rate of 36.6 percent was achieved. Using their best stategy they managed a win rate of about 70 percent. This is almost twice as high and not too far of the 82 to 91.5 percent Brarnason, et al. [8] deemed possible.

5 Experiments

The experiments in this research mainly consist of determining for different versions of SOLITAIRE which and how many games were non-winnable. In order to determine if a certain game is non-winnable, all possible branches of the game have to be tried and found to be losing. This is quite some work so just doing it by hand with physical playing cards is not viable.
For the experiments of this research project a command line version of SOLITAIRE was created. This made it possible to play many different games and also allowed the deck to be shuffled according to what needed to be researched. This custom version of SOLITAIRE is able to be played with different widths and deck sizes which was needed to experiment with different versions of SOLITAIRE.

5.1 The Solitaire program

Before discussing the experiments a bit of background information is provided about the SOLITAIRE program that was developed to run the experiments. The language the program was written in is C++. This language was chosen for its performance and made it possible to experiment with larger deck and board sizes than an interpreted language like Python would have.
In order to check all different branches a recursive algorithm is used which tries all different choices until either a winning state is reached or there are no possible moves to try anymore.

5.2 Computing the games

Some variants were small enough to be fully computed, meaning that all possible permutations of the deck were tried. For some of these we are able to classify the games that are non-winnable into categories based on a set of statistics.
First we provide some explanations about how all permutations of the decks were played for these smaller games, as some optimizations were implemented in order to reduce the number of permutations without losing accuracy in the results. Playing all possible permutations of a certain deck would require $n!$ games to be played, where n is the number of cards in the deck. However, this number could be reduced in a couple of ways leaving us with a much lower number and thereby enabling more different games to be computed within a reasonable time.

5.2.1 Stock optimization

The first optimization which led to a big performance increase without reducing the accuracy of the results was keeping the stock ordered. This does not result in fewer actually different games as different permutations of the stock have no impact on whether of not a certain game is winnable. Recall how Section 3 explained that all cards of the stock can be played at all times in the variant of SOLITAIRE on which this research is based. This optimization divides the number of total permutations by the factorial of the stock. Instead of the $n!$ permutations that would have to be

tried if we did not use this optimization we are left with $n!/m!$ where n is still the number of cards in the deck and m is the number of cards in the stock. A more clear formula is given in Equation 1.

$$\text{number\_of\_different\_games} = \frac{(\text{size\_of\_deck})!}{(\text{size\_of\_stock})!} \tag{1}$$

A disadvantage of this optimization is that it is dependant on the size of the stock. The size of the stock is variable depending on the size of the deck and the number of build stacks (width) over which the deck is spread for the initial starting state. Two examples to indicate this difference are the games where both variants have 3 cards per suit resulting in a deck size of 12 where the width differs from 3 to 4. The games where the width equals 4 will have $\sum_{i=1}^{4} i = 1+2+3+4 = 10$ cards in the build stacks leaving 2 cards in the stock, whereas the variant with a width of 3 build stacks will have $\sum_{i=1}^{3} i = 1+2+3 = 6$ cards in the build stacks and also 6 in the stock.
This results in the first variant only getting a $2! = 2$ times speedup from this optimization and the second variant getting a $6! = 720$ times speedup.[3] A more in depth formula for calculating the number of games that need to be played is found in Equation 2

$$\text{number\_of\_different\_games} = \frac{(4 \cdot \text{cards\_per\_suit})!}{(4 \cdot \text{cards\_per\_suit}) - \sum_{i=1}^{\text{width}} i)!} \tag{2}$$

5.2.2 Suit and color symmetries

Another optimization, or rather set of optimizations, is symmetry optimization. The basic idea behind symmetry optimization is that some different permutations of the deck can result in games that are played exactly the same. The reason behind this is that in a certain permutation of a deck, one can substitute all red cards for all black cards as long as the numbers of these cards stay the same and the relative suits within the colors also stay the same and still have a game that will be played the same.
Another switch that is possible without changing the way a game is played is substituting all cards of the same suit within a color. Here the numbers of the cards still need to be kept the same. An example of this would be changing all the clubs cards to spades and vice versa or doing the same with all diamonds and hearts cards.
The reason these symmetries are actually the same game is that in the game of Solitaire, the actual suits do not really matter. The suits of the cards only matter when they are compared to the suits of the cards they interact with. In the build stacks the only suit based requirement for a card to be placed on another card is that if a card is of either the diamonds suit or the hearts suit, it can only be placed on a card of either the clubs or the spades suit or the other way around. It does not matter which of the suits of the same color the card belongs to only that its color is different from the color of the card it is placed on as explained in Section 3. A real world example of two symmetric games is given in Figure 7.
In the suit stacks the requirements are more strict with regards to the suits. As only cards of the same suit can of course be placed in the same stack. This was also explained Section 3. This means that switching some cards of different suits within the same color can result in a game that can be played differently as this card may now be played to a suit stack which was not possible before or

[3] The word speedup here is used to indicate the reduction factor of the amount of work, rather than the actual real time speedup.

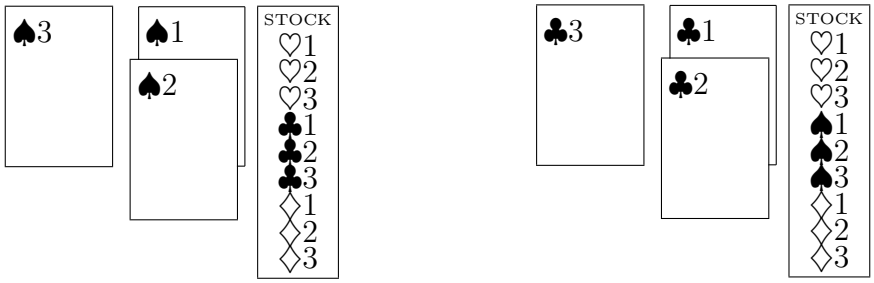

Figure 7: Example of two games that are symmetric within a color. The black suits are substituted.

may not be played to this suit stack where it would have been allowed before the switch. This is where the requirement of switching all cards from a certaing suit within a color comes from.
The way these symmetries are implemented in the program which makes all these permutations and plays them to see which ones are winnable, is as follows. The first card that is placed, which is in the way the cards are placed for the starting state in this research always the first and only card in the leftmost build stack, is assumed to be of the spades suit. This already divides the total number of games that needs to be played by 4 as it first halves the number by assuming the color of the first card to be black and then halves it again by assuming the suit of the first card to be spades. After this the first red card that will be placed is assumed to be of the hearts suit. This again halves the number of possible games by 2. All games which do not fit these assumptions are disregarded and not played, as they are all the same as one of the games which is still in the set of games that are played because of the discussed symmetry.
The optimizations in this set result in a combined reduction factor of the work that need to be done by the program of 8. Unlike the ordered stock optimization this speedup factor persists through all games of different sizes as all of the researched games have 4 suits spread over 2 colors.

6 Categorizing fully computed games

Now that it is clear how the games are played and which games are deemed unnecessary to be played we can take a look at the actual outcomes of some variants of the game. In this subsection we will take a look at some smaller variants that were small enough to be fully computed. Some of these variants were of a size where the different ways a game turned out to be non-winnable could be categorized in order to more clearly see the different ways in which a game can be non-winnable. These categories are based on certain variables which are updated throughout the playing of each game. These variables can later be used to group games that are somewhat alike into the categories as explained earlier. All variables which are kept as statistics for a game have minimal and maximal values except for one, which is a bitset of which cards are ever opened during the playing of a game. The bitset which keeps track of which cards are ever opened is useful to quickly group games together. This is because it keeps track of which cards will always remain face down, no matter which moves a player chooses to do, meaning that for all intents and purposes their order does not matter. This means that games which have the same set of cards that will never open can be considered the same game. This reduces the number of distinct games which are non-winnable for

a deck size and board width. A visual example of two games that only differ in the order of the cards that are never opened is given in Figure 8.

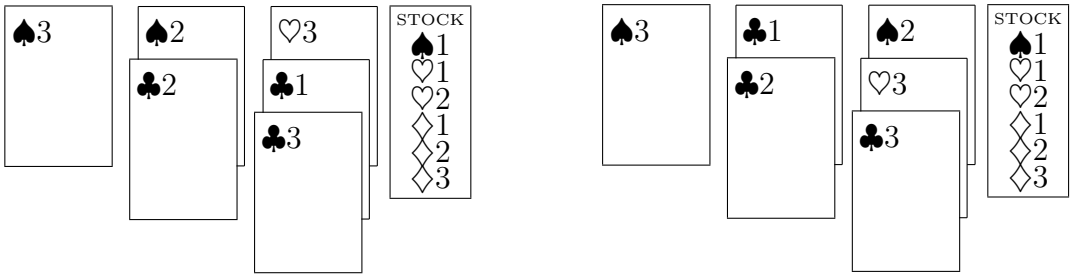

Figure 8: Example of games that have the same set of cards that are never opened. This set consists of the cards: ♣1, ♠2 and ♡3.

The other variables that are used to group games together are not as clearly trimming the games as the bitset of always closed cards. They are however useful for creating groups of games that are not winnable, thereby making it easier to fully understand a certain game size. A list of these variables and when they are updated is supplied in Table 4 and they are explained after that table. The reason most variables have their minimal values updated only when a game is lost instead of everytime a move is done is because it would just make a lot of their values 0.

Variable	min value	max value
depth	updated when game is lost	updated each move
cards to ace stack	updated when game is lost	updated each move
aces to ace stack	updated when game is lost	updated each move
cards left closed	updated each move	updated when game is lost
cards left in stock	updated each move	updated when game is lost

Table 1: Overview of the variables used to classify games.

An example of a variable which would be useless if their minimal value would be updated every move would be the depth variable. This variable keeps the depth of the current game tree. This means that each game always starts with a value of zero for depth as the starting state is always the root. Updating the minimal value here (or after the first move) would result in a minimal value of 0 (or 1) which could never be updated later as the depth could never get any lower. This is why it is only updated when a game is lost as this means that the minimal and maximal depth will be kept for a losing leaf instead of the maximum depth of a losing leaf and the depth of the root for the minimal value.
The variable that keeps how many cards are played to the ace stacks is updated in the same way as the depth variable. This is rather obvious as in the starting state no cards are played to the

ace stacks yet, which would again result in a minimal value of 0 for every game. The same is true for the variable that keeps track of how many aces are played to their ace stacks. This one is very similar to the cards to stack variable, only this one tells us how many different suits have at least one card in the ace stack.
The last two variables, containing the minimal and maximal values of the amount of cards left closed and the amount of cards which do not leave the stock are updated differently. Here the minimal value is updated each move and the maximal value is only updated when a game is lost. This is because, in contrast to the first three variables, these variables have their maximal values at the start of the games. The amount of cards in the stock can never increase. Updating this value immediatly at the start would again result in less usefull information as it would only tell how many cards are in the stock in the starting state, which is the same for all games of a certain size. Similarly less useful information would be kept in the maximum value of the amount of cards left closed as this can also decrease during a path from the root to a leaf.

6.1 The smallest games

The smallest games were all winnable. This includes all games with only 1 card per suit regardless of how many build stacks the game has, as these can of course all be played directly to their suit stacks after which the game is finnished.
Also the games with only 1 build stack are always winnable, regardless of how many cards per suit the deck has, as all cards in the stock can directly be played to the suit stacks with the one card in the build stack only having to wait its turn.
In the subsections below we will take a look at the variants of the game which actually have interesting results instead of just being trivial.

6.2 Games having 2 cards per suit and 3 build stacks

Arguably the smallest game size where not all games were winnable was 2 cards per suit spread over 3 build stacks. For every permutation this gives us a deck size of 8 cards in total of which 6 will be placed on the build stacks in the starting state and the other 2 will be placed on the stock. Of the 6 cards placed in the build stacks 3 will be placed facing up (open) and 3 will be placed facing down (Figure 9).

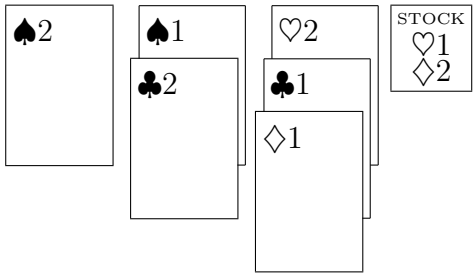

Figure 9: Example of a game with 2 cards per suit and a width of 3.

As the deck size is 8 for these games, the total number of different permutations of the deck is $8! = 40,320$. This is well within what is computable in a reasonable time, however, as explained

earlier, we have some optimizations to reduce the amount of games that need to be played. This not only reduces the amount of games the computer needs to try, but also reduces the number of games we need to understand by the same factor.
Firstly we divide by the factorial of the amount of cards in the stock. In this case that only results in a reduction factor of 2 as there are only 2 cards in the stock and $2! = 2$.[4] Secondly we know we only have to look at games where the first card is of the spades suit. This divides both the number of games for both us and the computer by 4. Thirdly we only take into consideration the games where the first red card is of the hearts suit deviding the number of games by 2 again. These optimizations leave us, or rather the computer, with only $40,320/(2 \cdot 4 \cdot 2) = 2,520$ games that need to be played.
Of these $2,520$ games, 18 were not winnable. Recall how the reduction factor of 16 for optimizing this game size also was applied to the non-winnable games. This means that without optimizing there would be $18 \cdot 16 = 288$ games to try to interpret. However, we do have the optimizations, so we can just consider the games in Table 2 to be the non-winnable games.

Decks							
Build stacks						Stock	
2S+	1S-	2H+	1H-	1C-	2C+	1D+	2D+
2S+	1H-	2H+	1S-	1C-	2C+	1D+	2D+
2S+	1S-	2H+	1C-	1H-	2C+	1D+	2D+
2S+	1C-	2H+	1S-	1H-	2C+	1D+	2D+
2S+	1C-	2H+	1H-	1S-	2C+	1D+	2D+
2S+	1H-	2H+	1C-	1S-	2C+	1D+	2D+
2S+	1H-	2C+	1S-	1C-	2H+	1D+	2D+
2S+	1S-	2C+	1H-	1C-	2H+	1D+	2D+
2S+	1S-	2C+	1C-	1H-	2H+	1D+	2D+
2S+	1C-	2C+	1S-	1H-	2H+	1D+	2D+
2S+	1H-	2C+	1C-	1S-	2H+	1D+	2D+
2S+	1C-	2C+	1H-	1S-	2H+	1D+	2D+
2S+	1S-	2H+	1H-	1D-	2D+	1C+	2C+
2S+	1H-	2H+	1S-	1D-	2D+	1C+	2C+
2S+	1S-	2H+	1D-	1H-	2D+	1C+	2C+
2S+	1H-	2H+	1D-	1S-	2D+	1C+	2C+
2S+	1H-	2D+	1S-	1D-	2H+	1C+	2C+
2S+	1H-	2D+	1D-	1S-	2H+	1C+	2C+

Table 2: All 18 non-winnable games for 2 cards per suit and a width of 3. The cards right of the vertical line are placed in the stock.

The way this table is to be read is reading every line as a permutation of the deck which is not winnable. Each card is printed in the same format where the number indicates the number on the card. As explained before, the number 1 is the ace and the highest number for a certain game size

[4]Reduction factor was called speedup factor before as the main concern used to be execution time. Now that we have to interpret the results, reduction factor seems more appropriate.

(in this case 2) behaves like a king card would in the regular 52 card deck. The capital letter after the number indicates the suit, with the straightforward 'S' for spades, 'H' for hearts, 'C' for clubs, 'D' for diamonds. Lastly the plus or minus sign indicates whether the card is placed face down (-) or face up (+) in the starting state.
The way these permutations are mapped to an actual game starting state is by placing the first (leftmost) card on the first (again leftmost) build stack. This card is always face up and of the spades suit. After that, the second card is placed in the second build stack (both of course left to right) where it is placed face down as it will be covered by the third card which is placed face up on this same stack. This will go on until all build stacks are filled with an increasing number of cards where the last placed card in each of the build stacks is always the only card facing up. This was explained more in depth in Section 3. The remaining cards are placed in the stock and all get a '+' sign as they are all open to being played. An example of how a line results in a starting game state is given in Figure 10.

2S+	1S-	2H+	1H-	1C-	2C+	1D+	2D+

Table 3: Example deck which is mapped to the starting state in Figure 10.

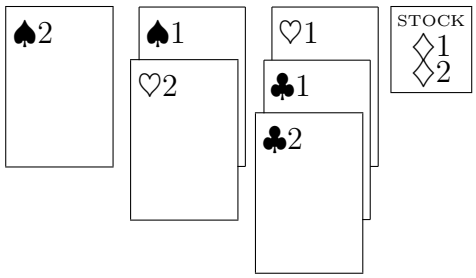

Figure 10: How the deck from Table 3 would be placed in a starting state.

As explained before in Section 6 the non-winnable games will be categorized. The way these games are categorized is based on a set of statistics which is also explained in that section. For the 18 non-winnable games in this section this categorization results in all games being put in only 1 category suggesting that all these games are basically the same.
A closer look at the values for the statistics which are found in Table 4 reveals that this is indeed the case, as they all share these statistics. Also from looking at the decks in Table 2 we see that all non-winnable games have all cards from one of the suits in the stock and the 2's of the other suits blocking the build stacks.

6.3 Games having 3 cards per suit and 2 build stacks

Similarly to the previous section, the set of games that can be created using a deck with 3 cards per suit and 2 build stacks also includes games that are not winnable. This deck size and game layout result in 3 cards being placed in the build stacks of which only 1 will be closed and as there are $4 \cdot 3 = 12$ cards in the deck, 9 remaining cards will be placed in the stock. A visual example is provided in Figure 11

Variable	min value	max value
depth	2	2
cards to ace stack	2	2
aces to ace stack	1	1
cards left closed	3	3
cards left in stock	0	0

Table 4: Statistics for the games with 2 cards per suit and a width of 3.

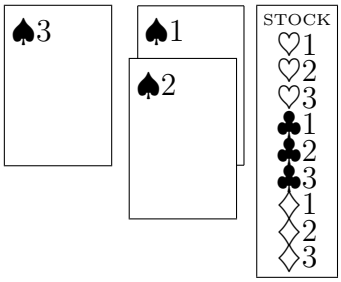

Figure 11: Example of a game with 3 cards per suit and a width of 2.

As before, the number of cards in the deck determine how many possible permutations can be made. In this case that comes down to $12! = 479,001,600$ (almost 500 million!). This is clearly a lot more than the $40,320$ permutations of the decks with 2 cards per suit. Luckily we still have the optimizations to reduce this gigantic number into someting less overwhelming.
We will first devide the number of permutations by the factorial of the number of cards in the stock. Unlike the last case where this only yielded a factor of 2 decrease in the number of games that needed to be played, here we can devide the number of games by a factor of $9! = 362,880$. This will obviously result in a huge decrease in the number of games that need to be played by the program and a just as big decrease in the number of non-winnable games that later need to be interpreted and classified.
After the first optimization we are left with only $1,320$ games. This number will be divided by a factor of 8 by the symmetry optimizations resulting in 165 games that need to be played by the algorithm. No further explanation will be given in this section with regards to these symmetry optimizations as they are already dicussed in the previous section and more general and in depth in Section 5.2.2.
The results of the games with 3 cards per suit and 2 build stacks are that only 2 were not winnable. It is not too surprising that the number of non-winnable games is low for this variant as there is only 1 cards that is not visible. However, do not confuse this number to mean only 2 games out of the $479,001,600$ total permutations are not winnable as these are 2 games out of 165 that are not winnable. To get the result of how many games out of all possible permutations of 12 cards spread over 2 build stacks and a stock are non-winnable, the number of non-winnable games out of the 165 would need to be multiplied by all the factors of optimization again.
Multiplying 2 non-winnable games by 8 for the symmetry and 9! for the permutations of the stock would result in $2 \cdot 8 \cdot 9! = 5,806,080$. This means that out of the $479,001,600$ total possible games

that could be made with this deck and game layout, 5,806,080 would not be winnable.
An overview of the non-winnable deck permutations is given in Table 5. This table includes headers for which part of the deck is placed in the build stacks and which part is placed in the stock.

Deck										
Build stacks			Stock								
2S+	1S-	3S+	1H+	2H+	3H+	1C+	2C+	3C+	1D+	2D+	3D+
3S+	1S-	2S+	1H+	2H+	3H+	1C+	2C+	3C+	1D+	2D+	3D+

Table 5: All non-winnable games for 3 cards per suit and a width of 2.

These games are again both in the same category based on the set of statistics they share (Table 6). This means that all the non-winnable games that consist of decks of 3 cards per suit and are layed out on 2 build stacks are basically the same.

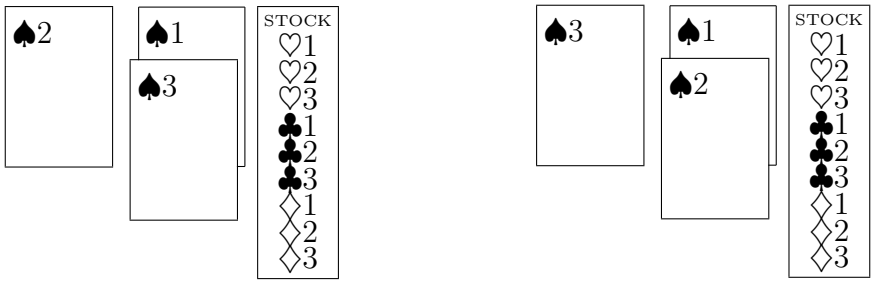

Figure 12: The two starting states of the non-winnable games for 3 cards and a width of 2.

Looking at the starting states for these 2 games in Figure 12 this is clearly visible. In both games no useful moves are ever possible on the build stacks as all spades cards are immovable. The only possible moves in this game are from the stock to the suit stacks. The red cards (hearts suit and diamonds suit) can first be placed on one of the build stacks in some case even enabling the ace of clubs to join, but none of these moves will change anything about the locked situation the spades cards are in.

Variable	min value	max value
depth	2	2
cards to ace stack	9	9
aces to ace stack	3	3
cards left closed	1	1
cards left in stock	0	0

Table 6: Statistics for the games with 2 cards per suit and a width of 3.

Another way of trimming is still possible for this category of games. The bitset (of always closed cards) approach to trimming the number of distinct games is still to be done. As the only closed

card in both these non-winnable games is the ace of spades, this would reduce the number of truly different non-winnable games to only one.
For this deck size and game layout we can summarize all non-winnable games into only one unique game. Either one of the games in Table 5 can be chosen as the unique game which all the other games of this size can be reduced to.
For this whole class of games it basically holds true that if at least one move from the build stacks can be done, then the game can be finnished. This excludes moves from cards that are first put on the build stacks from the stock, as all red 1's and 2's could first be placed on the build stacks before being moved to their suit stacks. This makes it easy to see from a starting state if a game is winnable or not.

6.4 Games having 3 cards per suit and 3 build stacks

The set of games that can be made using 3 cards per suit and a layout of 3 build stacks is arguably the most interesting set that is researched for this thesis. At first glance, one might expect it not to differ too much from the last section where we looked at the set of games using 3 cards per suit and only 2 build stacks, but this is incorrect. As per the usual there is again a visual representation of an example of a game for this version. This example is presented in Figure 13.

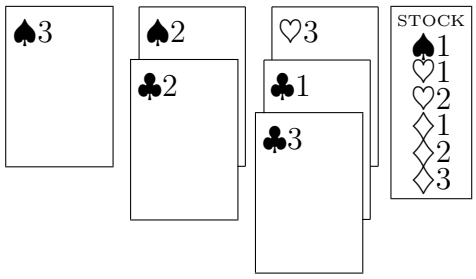

Figure 13: Example of a game with 3 cards per suit and a width of 3.

As this deck of also consists of $3 \cdot 4 = 12$ cards it does have the same number of possible permutations (the almost 500 million) however a much larger percentage of these actually have to be tried. Where the 2 build stacks version with the same number of cards had 9 cards left in stock in the beginning state, the 3 build stack version only has 6.
This seems like a minor difference but it has an enormous impact on the number of games that need to be played. Recall how the stock order does not matter, therefore we were able to devide the number of permutations of the total deck by the factorial of the stock. This results in a huge difference as $12!/9! = 1,320$ where $12!/6! = 665,280$. One can clearly see how changing the number of build stacks has an enormous impact on the number of games that need to be tried between.
This bigger number of games that needed to be tried also resulted in much more games that were not winnable, which differed from eachother more. The 3 cards per suit and 3 build stacks version of this game is therefore also the first version where the classification of the games actually made a real difference which is also part of the reason why it was previously called the most interesting version.
After running the experiment for this game size there appeared to be 742 games that were not winnable. This was after the symmetry optimiazation and of course after reducing the amount of

games to be played by not permutatig the sequence of the stock. This did not include the trimming using the bitset of closed cards.
After also trimming for this bitset the number of games that were non-winnable for 3 cards per suit and a width of 3 was 153 games. These games were spread over 18 classes. As 153 games are not easily interpreted by hand and would take a lot of space in this thesis there is no table including them all. Instead one game out of every class will be shown. These are presented in Table 7.

		Decks											
Class	# Games in class	Build stacks						Stock					
1	3	3S+	1C-	2S+	1S-	1H-	2C+	2H+	3H+	3C+	1D+	2D+	3D+
2	3	3S+	1S-	2S+	1C-	3C-	1H+	2H+	3H+	2C+	1D+	2D+	3D+
3	3	3S+	1C-	2S+	1S-	1H-	3C+	2H+	3H+	2C+	1D+	2D+	3D+
4	8	3S+	2C-	2S+	1S-	1H-	3C+	2H+	3H+	1C+	1D+	2D+	3D+
5	6	3S+	1S-	2S+	2C-	3C-	1H+	2H+	3H+	1C+	1D+	2D+	3D+
6	3	3S+	1C-	2C+	2H-	1S-	2S+	1H+	3H+	3C+	1D+	2D+	3D+
7	8	3S+	1C-	2C+	1S-	2H-	3C+	1H+	3H+	1C+	1D+	2D+	3D+
8	3	3S+	1C-	2C+	3H-	1S-	2S+	1H+	2H+	3C+	1D+	2D+	3D+
9	2	3S+	2S-	3H+	1C-	1S-	3C+	1H+	2H+	2C+	1D+	2D+	3D+
10	8	3S+	2C-	2S+	1S-	3H-	3C+	1H+	2H+	1C+	1D+	2D+	3D+
11	16	3S+	1C-	2S+	1S-	2C-	3C+	1H+	2H+	3H+	1D+	2D+	3D+
12	10	2S+	1S-	2H+	1H-	1C-	2C+	3S+	3H+	3C+	1D+	2D+	3D+
13	10	3S+	1S-	3H+	1H-	1C-	3C+	2S+	2H+	2C+	1D+	2D+	3D+
14	20	3S+	1S-	3H+	1H-	2C-	3C-	2S+	2H+	1C+	1D+	2D+	3D+
15	10	3S+	1S-	3H+	2H-	1C-	3C+	2S+	1H+	2C+	1D+	2D+	3D+
16	20	3S+	1S-	3H+	2H-	2C-	3C+	2S+	1H+	1C+	1D+	2D+	3D+
17	10	3S+	1S-	3H+	2H-	2D-	3D+	2S+	1H+	1C+	2C+	3C+	1D+
18	10	3S+	2S-	3H+	2H-	2C-	3C+	1S+	1H+	1C+	1D+	2D+	3D+

Table 7: All classes with a representing game for 3 cards per suit and 3 build stacks.

For every class which is numbered in Table 7 the values of the statistics for that class are presented in Table 9.
Games which are in the same class are again very similar in how they are played. This follows from the fact that they share the same statistics. It is intersting to see how most classes consisted of 10 or fewer games while there were a few classes which consisted of more games. Attempts to further split these classes in subclasses were tried with no success as the games making up these classes were indeed very similar.
An example of one of the smaller classes is presented in Table 8 where one can clearly see how these games are very similar. The statistics about this class can be found in Table 9 which states that for these three games always 3 cards can be put on the suit stacks. The three cards that can be placed on the suit stack are the 3 cards of the diamonds suit as these are all in the stock and can therefore always be accessed. This is further confirmed by the statistic about the number of aces that get moved to the suit stack, which is always equal to one for this class. This is just an example of how the statistics about a class translate to how the games of that class are similar.

The other statistics can be found by playing the games in a class in a similar manner.

Decks											
Build stacks						Stock					
3S+	1C-	2S+	1S-	1H-	3C+	2H+	3H+	2C+	1D+	2D+	3D+
3S+	1C-	3C+	1H-	1S-	2S+	2H+	3H+	2C+	1D+	2D+	3D+
3S+	1S-	3C+	1C-	1H-	2C+	2S+	2H+	3H+	1D+	2D+	3D+

Table 8: All games that are part of class 3 from Table 7

Class	depth	cards to ace stack	aces to ace stack	cards left closed	cards left in stock
1	4, 4	3, 3	1, 1	3, 3	2, 2
2	6, 6	6, 6	2, 2	2, 3	1, 1
3	5, 6	3, 3	1, 1	3, 3	2, 2
4	6, 7	4, 4	2, 2	3, 3	1, 1
5	7, 7	7, 7	3, 3	2, 3	0, 0
6	4, 4	4, 4	3, 3	3, 3	2, 2
7	5, 5	5, 5	2, 2	3, 3	1, 1
8	5, 5	5, 5	2, 2	3, 3	1, 1
9	6, 6	5, 6	2, 2	2, 3	0, 1
10	6, 6	6, 6	3, 3	3, 3	0, 0
11	6, 6	6, 6	2, 2	3, 3	0, 0
12	3, 3	3, 3	1, 1	3, 3	3, 3
13	6, 8	3, 3	1, 1	3, 3	1, 1
14	7, 9	4, 4	2, 2	3, 3	0, 0
15	5, 5	4, 4	2, 2	3, 3	1, 1
16	6, 6	5, 5	3, 3	3, 3	0, 0
17	7, 8	5, 5	3, 3	3, 3	0, 0
18	6, 6	6, 6	4, 4	3, 3	0, 0

Table 9: All classes with statistics values for 3 cards per suit and 3 build stacks.

6.5 Games having 3 cards per suit and 4 build stacks

The games with 3 cards per deck and 4 build stacks were the largest games that could be fully computed. This is the first game that allowed a width of 4 to be used as at least 10 cards are needed to fill the 4 build stacks in the increasing manner. Games with 3 cards per suit allow this as they have 12 cards. As per the usual there is a visual representation of the starting state of one of these games is presented in Figure 14.

This game size resulted in significantly more games to be played than before. In Section 6.3 and Section 6.4 we already saw how 3 cards per suit results in 12! possible permutations of the

deck, however in these sections the resulting number of $479,001,600$ permutations was still to be divided by 9! and 6! factorial respectively. This was because 9 and 6 were the sizes of the stock, which only holds a mere 2 cards in this variant. This means that for this size $479,001,600/2! = 479,001,600/2 = 239,500,800$ different possible games exist.

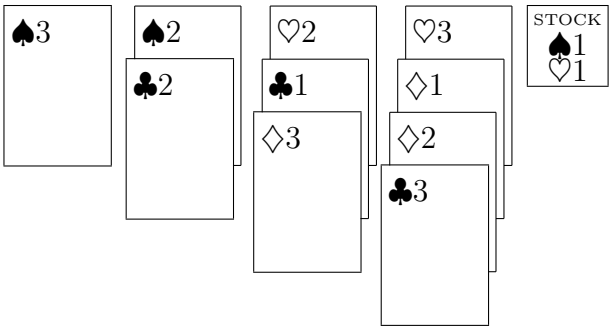

Figure 14: Example of a game with 3 cards per suit and a width of 4.

We would of course still be dividing this number because of the various symmetry optimizations. These were introduced in Section 5.2.2 and discussed in many sections thereafter and will therefore not be explicitely discussed in this section. What is interesting to note for this size is the execution time for the program. In earlier, smaller games, the total execution time to calculate all possible games and their outcomes never exceeded a few seconds. In this version running the experiments, even with all optimizations, took over 13 minutes.[5]
As there were a lot of different games to be played, there was also a lot of oppertunity to find games that were not winnable. This resulted in $668,790$ games that were not winnable out of $29,937,600$ games played.[6]
Categorizing these $668,790$ games in the same way as before resulted in many more classes. Recall how the games in the previous section could be divided into 18 different classes, in this case there were $1,212$ different classes of similar games. This is of course far too much to list in a table in this thesis therefore this thesis does not give a detailed explanation for all these classes and their statistics. However a couple of interesting examples from this long list of classes will be explained below. The numbers used in the following paragraphs are after trimming the classes using the bitset of always closed cards, this trimming brought the $668,790$ games back to $36,565$.
An example of an interesting and easily understandable class was the largest class. This class consisted of $5,760$ games which all could only be played in the same way. The statistics of this class reveal that all 6 cards are always left closed, the maximal and minimal depth are always 3, 1 card is played to the suit stack, 0 cards are left in stock and 1 ace is played to the suit stack (which of course follows from the fact that only 1 card is played to the suit stack). From these statistics we can imagine how all these games will be played. The depth of 3 implies that always 3 moves can be done before the game is lost. One of these moves is moving the single ace to a suit stack. The 2 other moves must involve the other card that started in the stock (as none of the cards starting in the build stacks can be moved without changing the $left\_closed$ statistic) which is placed on

[5]The mentioned execution times were achieved on an Intel i7 7700k with a clockspeed of 4.5 GHz
[6]This number is after all optimizations.

one of the build stacks, and then moved to another build stack after which no moves are possible anymore.[7]
Another example of an understandable and also large class is the follwing class. This class consists of 2,880 games and these are even simpler than the last example. All games in this class have only possible moves in the entire game tree. These moves are moving the 2 aces from the stock to the suit stacks. This follows from the statistics as the depth is always 2, all initially closed cards are left that way, there are always 0 cards remaining in the stock after the game has ended and the number of cards that are moved to the ace stacks is equal to 2.
From these 2 examples one gets an idea of how statistics describe how games are played. There are of course many more examples of classes (1,210 to be exact) of which some are way less obvious in how the games they consist of are played. The classes explained above were a lot larger than most of the other classes as the mean size of a class was 30 with a median of only 5. This indicates that the statistics used to classify the games in this version of SOLITAIRE may have been too specific as fewer, larger classes might have been better for this size. The reason to stick with this way of categorizing was to keep it consistant with the other versions discussed in this thesis.

6.6 Not fully computable games

The title of this section might be misleading as all SOLITAIRE games are of course computable given enough time. Therefore the games in this section are only not computable within reasonable time. This section will show examples of the biggest size games that are still playable using the algorithm from this thesis, but which not all can be played to really determine how many of them are winnable.
For the program from this thesis, the largest size of games that could be fully computed within reasonable time were the games with 3 cards per suit and 4 build stacks from Section 6.5. For larger games only a random set of the games could be played. The largest game size that could reasonably be tried was where the games had 5 cards per suit and also width of 5.

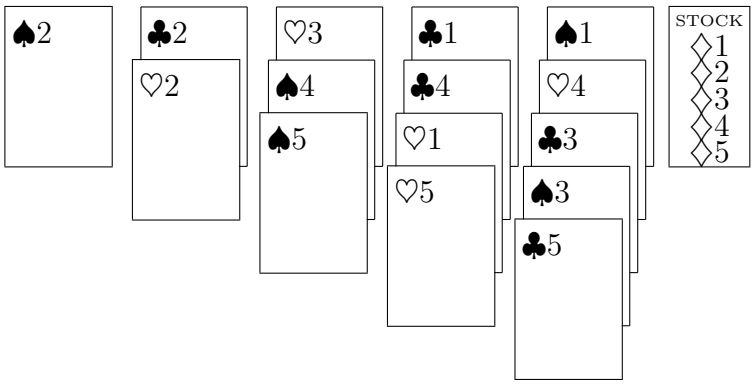

Figure 15: Example of one of the non-winnable games with 5 cards per suit and a width of 5.

To illustrate how not all games of this size could be played a couple of numbers for this size are given. The 5 cards per suit make for a total deck size of 20 which results in $20! = 2,432,902,008,176,640,000$

[7] In a normal SOLITAIRE game this card could just loop forever between these 2 build stacks, however in this program, loops are not allowed as they are never necessary when determining is a game winnable.

possible permutations. Dividing this by the factorial of the size of the stock (which only holds 5 of the 20 cards) leaves us with $20!/5! = 20,274,183,401,472,000$ games which is of course infeasible. Making matters even worse, the execution time of single games of this size also increased significantly. This is because there were more total choices for the player which is what determines the growth rate of the game tree. All of this resulted in only a small percentage of the games from this size being tested.
Experimenting with this game size the number of total losses that needed to be found were kept constant at 99. After playing a total of $3,458$ games, these 99 non-winnable games were found (which implies $3,359$ were winnable). One of the non-winnable games of this size is presented in Figure 15. The experiment took more than 20 hours which further confirms that bigger sizes were not feasible to try to compute.
The 99 non-winnable games out of $3,458$ total games suggest about 1 out of 35 games are non-winnable for this variant. However, out of the enormous number of total games that could be played with this deck size and width, the $3,458$ tested games are not really a representative subset.

7 Conclusions and Further Research

Concluding from this thesis it is clear that the success rate for the full game of SOLITAIRE is yet to be determined. It is highly unlikely that the success rate of the full 52 card version of Solitaire will ever be determined in the way that was tried in this thesis as there are just too many different possible games to play them all. Even the optimizations did not notably increase the chance for a computer being able to compute all these games.
The patterns of similar games from the smaller versions of SOLITAIRE that were found and discussed in this thesis did provide interesting results. They allowed for large sets of non-winnable games to be reduced to only a few base games (example from a class of games) to which all other games were similar. Using the statistics used to classify the set of unwinnable games a human interpreter could understand how games from a set were played giving insight into different ways how a game of SOLITAIRE is non-winnable.
More research regarding these patterns and applying them to bigger versions of the game may lead to even more interesting findings.

7.1 Further Research

Further research towards the success rate of SOLITAIRE would certainly be very interesting as there is still much more to be investigated. This thesis mainly looked into different categories of games that were not winnable based on statistics about these games, however different patterns of cards in the beginning state could also be used to form categories.
A possibilty for further research would be using an artificially intelligent neural network which could be trained on non-winnable games, to try to predict other non-winnable games. This could be done in multiple ways e.g., giving the AI agent knowledge of all cards in the beginning state or only showing the cards a player would be able to see in a normal game.
Using such an AI agent more non-winnable game candidates could be generated which could be tried using the program from this thesis. The results could then be used to update the neural

network to get even better at predicting which games are winnable. This way a feedback loop can be created between a predicting neural network and the testing algorithm used in this paper.
Patterns found by a neural network could also be used for a more mathematical approach to determining the success rate. The probability of these patterns existing in a certain permutation of the deck of cards could be determined and that way large numbers of games could be declared non-winnable.
There are several other approaches left to predict, or maybe one day even really determine, the success rate of the game of SOLITAIRE. These approaches will, however, not be discussed in this thesis.

References

[1] Online Windows98 Emulator. https://copy.sh/v86/?profile=windows98. Retrieved July 6th 2021.

[2] Wikipedia: Card Solitaire. https://en.wikipedia.org/wiki/Patience_(game). Retrieved July 6th 2021.

[3] J. DE RUITER. Counting classes of Klondike Solitaire configurations. Master's thesis, Leiden University, 2012.

[4] M. KORTSMIT. Strategies for Klondike Solitaire. Bachelor's thesis, Leiden University, 2014.

[5] P. B. DONKERSTEEG. Klondike strategies using Monte Carlo techniques. Master's thesis, Leiden University, 2010.

[6] P. DIACONIS, X. YAN, P. RUSMEVICHIENTONG AND B. VAN ROY. Solitaire: Man versus machine. In *Proceedings Advances in Neural Information Processing Systems* (2004), pp. 4–6.

[7] R. BJARNASON, A. FERN AND P. TADEPALLI. Lower bounding Klondike Solitaire with Monte-Carlo planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009* (2009), pp. 26–33.

[8] R. BJARNASON, P. TADEPALLI AND A. FERN. Searching solitaire in real time. *International Computer Games Association Journal* (2007), 131–142.

[9] UNKNOWN. *Das neue königliche l'Hombre*. 1788.