Strategies for Klondike Solita

Kortsmit, M.

Universiteit Leiden
Opleiding Informatica

Strategies for

Klondike Solitaire

Name: Marieke Kortsmit
Studentnr: 0740675
Date: 18/12/2014
Supervisors: Walter Kosters, Floske Spieksma

BACHELOR THESIS

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

Contents

1 Introduction

Klondike Solitaire is also known as Patience, or simply Solitaire. According to Parlett ([10] and [11]), the card game originated in the eighteenth century when fortune-tellers started cartomancy (fortune-telling with cards). This is supported by the fact that the earliest description of Patience occurs within a few years of the invention of card layouts for cartomancy. The popularity of the game has greatly benefited from the Microsoft Windows implementation, which has been inseparable from every version of Microsoft Windows since 1990. Today, many free Solitaire apps exist for playing on smartphone and tablet. The combination of these factors makes it highly unlikely, that one is not familiar with the game of Klondike Solitaire.

Section 1 introduces the basic game rules of Klondike Solitaire based on [5] combined with some well-known variations on these game rules. One of these variations is Thoughtful Solitaire, which probably has been studied most. In this variant the location of each card is known from the beginning of the game: all cards are face-up. Though a lot of research on this game variant has been done in [1] and [5], little information about these results is lined out. To give a notion of the complexity of the game, some theoretical results from [8] concerning the complexity are presented.

Section 3 shows that despite the fact that Klondike Solitaire is a famous game, a lot of uncertainty still exists regarding some aspects of the game. The exact probability of winning a game of Solitaire and the optimal strategy are unknown. Research on the amount of unplayable games has been done in [6], [7] and [13], and research on unsolvable games has been done in [1] and [2]. Altogether some research has been done concerning all kinds of problems about Klondike Solitaire, but none of them consider strategies that focus on handling the cards in the pile. In this section we examine one existing strategy from [5] for a game of Kondike Solitaire. This section also introduces two different, new pile-handling strategies for a game of Klondike Solitaire.

To be able to test the effectiveness of these two different pile-handling strategies we created a C++ program which is explained in Section 4. This implementation is used in Section 5, where some experiments concerning the pile-handling strategies are performed.

This thesis is written in pursuit of a Bachelor of Science degree in Computer Science and Mathematics from Leiden University in the Netherlands, and has been supervised by Walter Kosters and Floske Spieksma.

2 The game

This chapter will explain the game of Klondike Solitaire. First, the basic game rules are pointed out. Second, a few common variations on these rules will be shown. Third, we will give a notion of the complexity of the game.

2.1 Explanation

The traditional version of Klondike Solitaire is played with all 52 playing cards in a standard deck of cards without Jokers. The cards are divided into four suits: two black suits (Spades ♠ and Clubs ♣), and two red suits (Hearts ♡ and Diamonds ♢). Every suit has 13 cards with rank $1, \ldots, 13$, where rank 1 is associated with Ace (A), rank 11 with Jack (J), rank 12 with Queen (Q) and rank 13 with King (K). The game features different locations: the *pile*, the *talon*, four *suit stacks* and seven *build stacks*. A possible initial configuration of the game is displayed in Figure 1.

Figure 1: A possible initial configuration of Klondike Solitaire; screenshot from [3].

The game has the following rules (formulated based on [5]):
Build stacks:

- Build stacks are numbered from left to right, $1, \ldots, 7$.
- At the start configuration of the game, the cards are divided with one card on the first build stack, two cards on the second build stack, etc.
- The top card in every build stack is face-up. The rest of the cards in the build stack are face-down.
- The (stack of) open card(s) on one build stack is named a *card block*. A card block can be moved to another build stack, provided that the receiving build stack accepts the bottommost face-up card of the sending build stack. All face-up cards in a card block are to be moved at once. After moving a card block to a receiving build stack, all received cards and already present cards form a new build stack together. The order is preserved.
- If all cards from a build stack are moved, an empty stack remains.
- An empty stack can only accept a King. The movement of a King to an empty build stack is called a *Kings move*.
- If the upper card in a build stack is face-down, it automatically turns face-up.
- The upper card of a build stack can be moved to the top of a suit stack provided that the receiving suit stack accepts this card.
- A build stack will only accept an incoming card block if the upper card of the build stack is adjacent to and braided with the bottom card of the card block. We define:
  - ⋆ A card is *adjacent* to another card with rank r if the card has rank $r - 1$.
  - ⋆ Two cards are *braided* with each other if they have a different color.

Suit stacks:

- Every suit stack corresponds to one suit.
- At the initial configuration of the game, all suit stacks are empty.
- Cards are being accepted in ascending order.
- A suit stack can only accept cards with the same suit.
- An empty suit stack can only accept an Ace.
- If the suit stack is not empty, it will only accept cards of one rank higher than the upper card of the suit stack.

Pile and talon:

- The pile initially contains 24 cards, the talon is initially empty.
- If the pile is not empty, the upper three cards in the pile can be moved to the talon one by one. Then they can be played in a first-in-last-out order.
- If the pile is empty, all cards from the talon can be dealt back to the pile. This will preserve the ordering of the cards. This move can be done an infinite number of times.
- A topcard of the talon can be moved to a suit stack or a build stack, provided that the stack accepts the card.

The goal of the game is to move all cards to the suit stacks. If this succeeds, the game is won.

If we generalize the game, we use a deck of cards containing sn cards, divided over s suits. If the number of suits is greater than four, and even, half of the suits is black and half of the suits is red. In the most common variant of Klondike Solitaire the number of suits is four: two black suits (Spades ♠ and Clubs ♣), and two red suits (Hearts ♡ and Diamonds ♢). Every suit contains n cards with ranks $1, 2, 3, \ldots, n-2, n-1, n$. Rank 1 is associated with generalized Ace (A), rank $n-2$ with generalized Jack, (J), rank $n-1$ with generalized Queen (Q), and rank n with generalized King (K). The game consists of the *pile* initially containing p cards, *talon*, s *suit stacks* and b *build stacks*. Depending on variables s and n, the last build stack may contain some remaining cards from which the initial number of cards in the pile will be determined. An empty build stack can accept any card with rank n, and an empty suit stack can accept any card with rank 1. The rules remain unchanged.

2.2 Variations

Klondike Solitaire has a lot of proporties we can adjust, so the game will be slightly different. Small adjustments can have a major influence on the probability of winning the game. The following variations are interesting and will be described according to the differences between the variant and the game as formulated in Section 2.1.

2.2.1 Initial build stack variations

The initial build stack configuration as described in the rules in Section 2.1 can be adjusted in all kinds of forms, for example a rectangular form with the same amount of cards on all build stacks. If we consider Solitaire without a pile, we have to determine how to add the remaining cards to the build stacks. Of course more than seven build stacks is a possibility as well. In this thesis the standard initial build stack configuration is observed.

2.2.2 Moving a card block

It is only allowed to move a complete card block to another build stack. A variation is to allow movement of partial card blocks. The movement of partial card blocks is allowed in [2] and [1]. As in [5], the option for moving partial build stacks is disregarded in this thesis.

2.2.3 Move card from suit stack to build stack

In some versions of Klondike Solitaire it is allowed to move the uppermost card of a suit stack, back to a build stack, provided that the build stack accepts this card. This option is allowed in [5], [1] and [2]. We will not discuss the details of this variation, due to simplification reasons, though it does allows for a greater amount of possible moves.

2.2.4 Move cards from pile to talon

If the pile is nonempty, the upper card can be played to the talon. The upper card in the talon can be played. The pile initially contains 24 cards. This means that all 24 of these cards can be used directly. This variant is called *deal-1*. An example of all playable cards can be found in Figure 2.

Figure 2: Playable cards in pile in case of deal-1. Created with use of [3].

A common variation is *deal-3*, which is used in [5], [1] and [2]. In this case, if the talon is empty, the 3 upper cards are played to the talon in a first-in-last-out way. Only 8 out of 24 cards in the pile are now directly playable, and there is no guarantee that one will eventually be able to play all 24 cards in the pile. This reduces the number of possible moves, resulting in a harder game. Figure 3 shows which cards are directly playable in a deal-3 situation. In this figure the pile is dealt to the talon from left to right: the leftmost card is the first card in the talon. All directly playable cards are shifted up. If we now play the first possible card, ♡3, then the number of playable cards directly increases. All cards that are now playable can be found in Figure 4. This figure follows directly from Figure 2: all directly playable cards are shifted up. This method of determining the playable cards is a very greedy method. After playing a single card from the pile, the complete set of playable pile cards may be different. In general the

following holds: after playing the card with index $r \in \{0, \ldots, 23\}$, we shift all cards to the left. This means that all cards with an initial index higher than r will have their index decreased by 1. Let us assume there are p cards left in the pile. Then the cards with the following index become playable, directly after playing card r:

- index $r + 3m < p$, with $m = 1, 2 \ldots$, and $m < \frac{p-r}{3}$.
- index $r - 1 + 3m < p$, with $m = 0, 1 \ldots$, and $m < \frac{p-r+1}{3}$.
- index $2 + 3m < r$, with $m = 0, 1 \ldots$, and $m < \frac{r-2}{3}$. These cards were playable before playing card r as well.
- the last card, provided it is not caught by previous statements.

In some variants of the game the number of times the pile is playable to the talon is finite, but this option will be disregarded.

Figure 3: Playable cards (the prevailing cards) in pile in case of deal-3. Created with use of [3].

Figure 4: Playable cards (the prevailing cards) in pile in case of deal-3, after playing card ♡3. Created with use of [3].

2.2.5 Thoughtful Solitaire

Thoughtful Solitaire has the exact same rules as Klondike Solitaire. The only difference is that in Thoughtful Solitaire the location of each card is known at the beginning of the game; all cards are face-up. This provides the option to use more information about the location of cards to determine which move is optimal. As reported in [5] a heuristic player (following practically the same rules as our SIMPLEST HEURISTIC STRATEGY as formulated in Section 3.2.1) for Thoughtful Solitaire can supposedly win approximately 13.05% of the games, and with a rollout strategy (a rollout strategy computes an action that would result from an iteration of policy improvement applied to the heuristic policy; it may be considered an alternative heuristic that improves the original), this percentage gains to an alleged 70.20%. In [1] even a percentage of 82% is posed. This implies a bound for the number of winnable games of Klondike Solitaire. The used solving methods for Thoughtful Solitaire can be used to solve Klondike Solitaire. Monte-Carlo techniques are applied to the random probability distribution of the closed cards in Klondike Solitaire. With such a solver, using the same heuristic as [5], [2] claims 36.97% of the games of Klondike Solitaire can be won. Thoughtful Solitaire is not further investigated in this thesis.

2.3 Complexity

The problem of determining whether an n-card Klondike initial configuration can lead to a win is complex. This decision problem is referred to as KLONDIKE. Problems discussed in Chapter 3 are a result of this complexity. The great complexity of this problem has been extensively researched in [8]. Here the following definitions are used.

Definition 2.1 (SOLIT(b, r))**.** Let an initial b-black-suit and r-red-suit KLONDIKE configuration involving the same number n of cards per suit be given. Determine whether the $(b + r)n$ cards can be placed on the $b + r$ suit stacks by applying the standard KLONDIKE game rules starting from the given initial configuration.

Definition 2.2 (FLAT-SOLIT(b, r))**.** SOLIT(b, r) with an initial configuration having an empty pile and empty talon.

Definition 2.3 (FLAT-SOLIT$_{\text{NoKing}}(b, r)$)**.** FLAT-SOLIT(b, r) played with modified rules that forbid an empty stack from accepting a King.

In [8] the following complexity results about the different types of KLONDIKE Solitaire are proven:

- KLONDIKE is NP-complete and remains so with only three suits available.
- KLONDIKE with a black suit and a red suit is NL-hard.
- KLONDIKE with any fixed number b of black suits is in NL. The same holds for any fixed number of r red suits.

- KLONDIKE with a single suit is in $AC^0[3]$.
- Flat KLONDIKE (KLONDIKE Solitaire with an empty pile) is NL-complete for an arbitrary number of suits.
- Flat KLONDIKE with 2 black suits and without Kings is in AC^0. The same holds for 2 red suits without Kings.

These statements give a notion of how complex the practical problems stated in Chapter 3 are. Precise definitions of all used complexity classes can be found in [8] and [9]. Cited from [8] we have:

$$\mathrm{AC}^0 \subset \mathrm{AC}^0[3] \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}.$$

Here, the complexity class AC^0 is the set of languages accepted by DLOGTIME-uniform unbounded-fan-in constant depth $\{\wedge, \vee, \neg\}$-circuits of polynomial size. The larger class $\mathrm{AC}^0[3]$ is the set of languages AC^0-Turing reducible to the MOD_m Boolean function, defined to output 1 if and only if 3 does not divide the sum of its Boolean inputs. The classes L and NL stand for deterministic and nondeterministic logarithmic space respectively. The classes P and NP are deterministic and nondeterministic polynomial time respectively.

3 Game problems

Despite the fact that Klondike Solitaire is a famous game, there is still a lot of uncertainty regarding some aspects of the game. The exact probability of winning a game of Solitaire and the optimal strategy are unknown. Also, little research has been done to establish the significance of the way of handling the cards in the pile. In this section the probability of winning a game and different strategies to win a game are examined. The strategies focus mainly on finding a strategy that handles moving of the pile cards. In Section 5 some experiments concerning these strategies will be performed. This entire section will concern standard games of Unthoughtful Klondike Solitaire, so the variant of Thoughtful Klondike Solitaire explained in Section 2.2.5 is disregarded.

3.1 Unwinnable games

The initial configuration of Klondike Solitaire is determined by distributing all available cards randomly to all allowable initial locations. This has no further restrictions, so it is possible that a game of Klondike Solitaire is unwinnable. We distinguish two types: unplayable games and unsolvable games. Unplayable games of Klondike Solitaire are games for which the initial state (the state of the game in the initial configuration) of the game is unplayable: there are no possible moves. Unsolvable games of Klondike Solitaire are games which can never be won: the game will eventually result in an unplayable game state which is not necessarily equal to the initial game state. It directly follows that unplayable games are automatically unsolvable.

3.1.1 Unplayable games

The initial situation of a traditional game of Klondike Solitaire with deal-3 consists of 15 cards from which one card is potentially moved in the first move. These moveable cards consist of the 7 top cards of each build stack and the 8 playable cards in the talon. It follows that the number of different initial configurations equals:

$$\binom{52}{7} \cdot \binom{45}{8} = 28,837,689,349,669,200 \approx 3 \cdot 10^{16}.$$

If one of these 15 cards is an Ace, the first possible move is a fact. This Ace can be moved to a suit stack directly. The probability that none of these 15 directly playable cards contain an Ace, equals:

$$\binom{48}{15} \Big/ \binom{52}{15} \approx 0.2439.$$

However, this will not cover all unplayable games. It may be possible that 2 out of the 7 playable cards on the build stacks can be stacked. This is only the case if the cards are adjacent and braided. Another option is that one of the 8 playable cards in the talon can be moved to a build stack. The following conditions have to hold simultaneously in case of an unplayable game:

- There is no Ace among the 15 playable cards.
- None of the 7 playable build stack cards can be moved to another build stack.
- None of the 8 playable talon cards can be moved to a build stack.

Latif [7] used Monte Carlo simulations to show that the percentage of unplayable games equals approximately 0.25%. Donkersteeg [6] shows that this percentage can be determined exactly, by using brute force, from which it follows that the number of unplayable games equals 72,099,595,172,416. This equals a percentage of 0.25002%. De Ruiter [13] confirms this percentage by dynamic programming.

3.1.2 Unsolvable games

The exact percentage of unsolvable games of Klondike Solitaire is still unknown. In the literature, [1] states that no less than 82.0% and no more than 91.4% of Klondike Solitaire games have winning solutions, leaving the percentage of unsolvable games between 8.6% and 18.0%. In [2] this result is confirmed. In this section we will investigate the unsolvability of games of Klondike Solitaire, depending on variations in number of cards and number of suits.

One suit without Kings move In the case of one suit and an empty pile, what determines the winnability of the game is whether the cards in the build stacks are in order. In this situation, moves which move a card from a build stack to a suit stack are the only moves allowed. The only possibility to win the game is to move each card one by one directly to the suit stack. To be able to play all cards to the suit stack by only using this type of move, all cards in the build stack need to be in order. Consider the following game variation:

- An empty pile.
- The Kings move, moving a King to an empty build stack, is not permitted.
- There are b build stacks.
- The first build stack contains b_1 cards, the second build stack contains b_2 cards, etc.
- Every top card in a build stack is face-up.

The probability for such a game to be winnable equals:

$$\mathbb{P}(\text{cards in build stack all in order}) = \prod_{i=1}^{n} \frac{1}{b_i!}. \tag{3.1.2.1}$$

3.2 Strategy

An optimal strategy for winning Klondike Solitaire is unknown. However, some notion about what moves are favorable over others is present. For example moving an Ace or 2 from a build stack to a suit stack is always without risk. A little more attention is needed for further expanding the suit stacks. In general it seems to come in handy to keep the ranks of the top cards at all suit stacks close to each other. If there are multiple options in moving a card block to another build stack, it seems favorable to move the card block from the build stack which contains the most remaining face-down cards. It also seems unnecessary to move the last card in a build stack, if you do not have a King available to move to the residual empty build stack.

In this thesis we observe three strategies for Klondike Solitaire. The first strategy is an existing strategy from [5], which we will refer to as SIMPLEST HEURISTIC STRATAGY. For this strategy we try to obtain the optimal parameter values. The second and third strategies are new strategies. In the first kind, which we will refer to as GREEDY PILE STRATEGY, we play by scoring moves, meaning that we will score all possible moves and make a decision based on which move has the highest score. In the second kind, which we will refer to as POINT PILE STRATEGY, we play the game by scoring game states, meaning that we will score all game states reachable by doing a single move and choose the move which leads to the game state with the highest score. We will score the states based on giving a high score to a more favorable state. For example, a game state from

which a lot of moves are possible could be favorable over a game state with only one possible move left. By running experiments using these two kinds of strategies, we will gain more insight about how to choose which move is best. Note that all three strategies can be used for deal-1 and deal-3, but in this thesis we will focus on deal-3.

To be able to determine if one strategy is better than an other, we have to somehow validate a strategy. The strategies are measured by evaluating the end state of a game, which is the state in which the game either gets stuck or is won. The properties of the end state which validate a strategy are shown in Figure 5. For example a better strategy might be indicated by a higher number of moves before the game is lost. These properties are only used to report about different strategies in Section 5.

	The outcome which might indicate a better strategy:
If the game is lost	
Number of moves	High
Number of cards in pile	Low
Number of closed cards	Low
Number of cards in suit stack	High
If the game is won	
Winning percentage	High
Number of moves	Low
General	
Time per game	Low

Figure 5: Strategy validation properties.

3.2.1 Standard scoring of moves

To be able to program a heuristic player we will have to assign values to possible moves, and eventually we will even have to assign priorities to moves with equal values. To be able to formulate such a score-based strategy, we have to make a lot of assumptions and choices. The assumptions and choices explained in this section are mainly based on intuition gained from a lot of game playing, and on the heuristic strategy from [5]. The scoring method in this section will be referred to as the SIMPLEST HEURISTIC STRATEGY.

Value of moves We use parameters to assign a value to each move based on the heuristic strategy from [5] described below. We use parameters for each type of score:

- The initial score is 0.

- When a card is moved from a build stack to a suit stack, `SCORE_0` points are gained.
- When a card is moved from a build stack to another build stack, `SCORE_1` points are gained.
- When a card is moved from the pile to a build stack, `SCORE_2` points are gained.
- When a card is moved from the pile to a suit stack, `SCORE_3` points are gained.

A short overview of all types of moves with associated scoring parameters can be found in Figure 6. The move that maximizes the score will be executed, in case of a tie priorities of a move play a role. The used priorities are described below. In [5] the parameters in Figure 6 have values `SCORE_0 = 5`, `SCORE_1 = 0`, `SCORE_2 = 0` and `SCORE_3 = 5`. This choice is not motivated in the article.

Type	Description	Scoring parameter
0	From build stack to suit stack	`SCORE_0`
1	From build stack to build stack	`SCORE_1`
2	From pile to build stack	`SCORE_2`
3	From pile to suit stack	`SCORE_3`

Figure 6: Allowable move types with associated parameters in case of the SIMPLEST HEURISTIC STRATEGY.

Priority of moves with equal score The choice between moves according to this score system is not unique if the maximum score can be achieved by different moves. To be able to make a decision about what move to choose, we introduce the priority as follows (based on [5]):

- If the move transfers a card block from build stack to build stack, one of the following priorities holds:
  - ⋆ If the movement of the card block means a new card can be turned face-up, assign a priority of $k + 1$, in which k equals the number of face-down cards in the build stack.
  - ⋆ If the move empties a build stack, assign a priority of 1.
- If the move transfers a card from talon to build stack, the following priority holds:
  - ⋆ If the moved card equals K, assign a priority of 1.

All other moves have a priority of 0. In case of multiple moves with a score equal to the maximum score, and a priority equal to the maximum priority, we choose randomly among those moves.

3.2.2 Scoring pile moves greedy

The way of scoring and prioritizing moves as described in Section 3.2.1 gives little special attention to pile moves, since we do not have any priority rules formulated for them. The way of handling pile moves is actually a very important part of playing, and thus winning, a game of Solitaire. We introduce a greedy pile move strategy, based on a Monte Carlo Tree Search like approach for the pile moves. Monte Carlo Tree Search is a search method combining the precision of tree search with the generality of random sampling, as explained in [4]. In this greedy pile strategy we do not use the scores of integer type `SCORE_2` and `SCORE_3`, but we determine different values for these scores for each situation in the pile. This results in the following approach, reffered to as the GREEDY PILE STRATEGY:

We use parameters for each type of move:

- The initial score is 0.
- When a card is moved from a build stack to a suit stack, `SCORE_0` points are gained.
- When a card is moved from a build stack to another build stack, `SCORE_1` points are gained.
- When a card is moved from the pile to a build stack, `SCORE_`x points are gained.
- When a card is moved from the pile to a suit stack, `SCORE_`x points are gained.

The variable `SCORE_`x with initial value of 0 is determined as follows:

- Copy the complete game state.
- Execute the considered pile move in the copied game state. This will result in a game state with `nr_pile` possible pile moves (moves from the pile to another location which are moves of type 2 and 3 as in Figure 7).
- Now continue the game in the following way until there are no possible pile moves left:
  - ⋆ Take a random pile move of the `nr_pile` possible pile moves and execute this move in the copied game state.
  - ⋆ Update `SCORE_`x by adding:
    - ∗ `SCORE_NOT_PILE` multiplied by the number of possible not-pile moves in this game state.
    - ∗ `SCORE_PILE` multiplied by the number of possible pile moves in this game state.

⋆ The resulting game state has `nr_pile` possible pile moves. If `nr_pile` is at least one, then repeat. Else, break and return the value of `SCORE_`x.

A short overview of all types of moves with associated scoring parameters can be found in Figure 7. The move which maximizes the score will be executed, in case of a tie a random move of the moves with the maximum score is executed.

Type	Description	Scoring parameter
0	From build stack to suit stack	`SCORE_0`
1	From build stack to build stack	`SCORE_1`
2, 3	Pile move	Variable `SCORE_`x determined by: • `SCORE_NOT_PILE` • `SCORE_PILE`

Figure 7: Allowable move types with associated parameters in case of the GREEDY PILE STRATEGY.

3.2.3 Scoring pile moves point-based

In addition to the GREEDY PILE STRATEGY, we introduce a point-based pile move strategy. This strategy is based on specific game situation characteristics which are observed when performing a specific pile move. We do not use the scores of integer type `SCORE_2` and `SCORE_3`, but we determine different values for these scores for each situation in the pile. This results in the following approach, referred to as the POINT PILE STRATEGY:

We use parameters for each type of move:

- The initial score is 0.
- When a card is moved from a build stack to a suit stack, `SCORE_0` points are gained.
- When a card is moved from a build stack to another build stack, `SCORE_1` points are gained.
- When a card is moved from a pile to a build stack, `SCORE_`x points are gained.
- When a card is moved from a pile to a suit stack, `SCORE_`x points are gained.

The variable `SCORE_`x with initial value of 0 is determined as follows:

- Copy the complete game state.
- Execute the considered pile move in the copied game state. This will result in a new game state. Now update `SCORE_`x by adding:

- ⋆ SCORE_FIRST_PILE multiplied by the number of possible movable pile cards with previous index equal to 0, 3, 6,... as in Figure 3.
- ⋆ SCORE_SECOND_PILE multiplied by the number of possible movable pile cards with previous index equal to 1, 4, 7,... as in Figure 3.
- ⋆ NEW_BUILD_MOVE multiplied by the number of possible moves of type 1.
- ⋆ NR_SUIT_MOVES multiplied by the number of possible moves of type 0 or 3, which equals the number of possible moves to the suit stack.

A short overview of all types of moves with associated scoring parameters can be found in Figure 8. The move which maximizes the score will be executed, in case of a tie a random move of the moves with the maximum score is executed.

Type	Description	Scoring parameter
0	From build stack to suit stack	SCORE_0
1	From build stack to build stack	SCORE_1
2, 3	Pile move	Variable SCORE_x determined by: • SCORE_FIRST_PILE • SCORE_SECOND_PILE • NEW_BUILD_MOVE • NR_SUIT_MOVES

Figure 8: Allowable move types with associated parameters in case of the POINT PILE STRATEGY.

4 Implementation

To be able to simulate a large amount of Klondike Solitaire games, a C++ program has been written. In this section an outline of the implementation structure of this program will be presented.

4.1 Storage and initialization of the game state

All cards are numbered $1, 2, \ldots, sn$. The used and unused cards are saved in arrays as follows:

- card_nrs[2][MAX_SUITS * MAX_CARDS_PER_SUIT]: contains the rank and suit of each card. In the program all cards are referred to with a number. When the rank or suit of a card is needed, it is retrieved from card_nrs.
- used_cards[MAX_SUITS * MAX_CARDS_PER_SUIT]: contains all card numbers from which the location is known; all face-up cards.
- unused_cards[MAX_SUITS * MAX_CARDS_PER_SUIT]: contains all card numbers from which the location is unknown; all face-down cards.

The game state is saved in arrays as follows:

- `build_stacks[MAX_STACKS][3 * MAX_CARDS_PER_SUIT]`: contains all build stacks. All numbers of face-up cards are stored. The first entry of each stack is reserved for the number of face-down cards in this particular build stack. The value `3 * MAX_CARDS_PER_SUIT` ensures there is enough space reserved for the build stacks during the game.
- `suit_stack[MAX_SUITS]`: for each suit it contains the highest card rank present in the corresponding suit stack.
- `pile[MAX_PILE]`: contains all card numbers of cards in the pile.
- `playable_pile_cards[2][MAX_PILE]`: contains all cards from the pile which can be directly played.

4.2 Variables for different game rules

We want to investigate the influence of certain game rules on the game. To be able to do so, we have to create variables that determine the rules. These are as follows:

- `bool MOVE_KING`: is valued true if we allow for a King to move to an empty build stack.
- `bool ASCENDING_BUILD_STACKS`: is valued true if we want to create ascending build stacks, with one card in the first build stack, two cards in the second build stack, etc. If this variable is false, then we want to create build stacks with equal height, where
  - `int NR_OF_STACKS`: states the amount of stacks in case of stacks of equal height.
  - `int NR_OF_CARDS_PER_STACK`: states the height of stacks in case of stacks of equal height.
- `int DEAL`: is valued 3 if we want the pile to handle a deal-3 move as stated in Section 2.2.4. All other integer values are also possible, but only the value 1, meaning deal-1, describes a game which is common to play.
- `int NR_OF_SUITS`: the number of used suits.
- `int NR_OF_CARDS_PER_SUIT`: the number of cards per suit.
- `int NR_OF_CARDS_IN_PILE`: the number of cards present in the pile.

4.2.1 Implementation of the pile

As explained in Section 2.2.4, not all of the cards in the pile can always be used directly. In case of deal-1, all cards in the pile are accessible directly. But: in case of deal-3 each card with index `mod 3 == 2` is available directly, meaning the cards with index `mod 3 == 1` and `mod 3 == 0` are specifically not available directly. The possibilities to access cards with index `mod 3 == 1` and `mod 3 == 0` are described in Section 2.2.4.

In our implementation of the game we keep an array `playable_pile_cards[][]`, containing all directly accessible cards. The array is updated each time a move from the pile is executed.

4.3 Function find_possible_moves

The function `find_possible_moves` searches for all possible moves in the given game situation. The possible moves are categorized in different types. The categorization can be found in Table 6. The function stores the following information per move:

- the type of move, as stated in Table 6.
- the index of the to be moved card in the location of origin.
- the new index of the to be moved card in the location of destination.

In case we use `play_strategy_heuristic` or `play_strategy_evaluation` it also stores the associated score, and in case of `play_strategy_heuristic` it additionally stores the associated priority. `find_possible_moves` determines how many different moves are possible and stores this integer in `nr_of_possbile_moves` as well.

4.4 Function do_move

The function `do_move` carries out a certain move. It updates the build stacks, suit stacks, pile and the directly playable pile cards to the new situation. If after performing a move one of the build stacks has a face-down card on top, the card automatically gets turned face-up. In this case the arrays `used_cards` and `unused_cards` will be updated as well.

4.5 Implementation of different strategies

To be able to test different strategies, there is a first major distinction in strategy for which two different functions are used. For the non-random strategy, different variables are used to be able to make slight adjustments to the strategy.

Function play_strategy_random The function `play_strategy_random` determines, as long as possible, a random move and calls the function `do_move` to execute the determined random move. This iteration stops when there are no possible moves left. In this case the game is won, or it is stuck.

Function play_strategy_heuristic The function `play_strategy_heuristic` determines, as long as possible, the best heuristic move to carry out for the SIMPLEST HEURISTIC STRATEGY and calls the function `do_move` to execute the determined move. The best heuristic move is the move with maximum score. If this results in more than one possible move, the move with maximum priority is chosen. (the exact scores and priority rules which are used can be found in Section 3.2.1). If this is ambiguous it chooses a random move amongst all moves with maximum score and maximum priority. This iteration stops when there are no possible moves left. In this case the game is won, or it is stuck.

Variables for different strategies To be able to test certain slight strategy adjustments for game playing, variables for turning these strategies on or off are implemented:

- `bool MAX_DIFF_SUIT_STACK`: is valued true if we prefer a move to the suit stack if the maximum difference between the top ranks in the different suits is restricted. Of course, it there is no other possible move, a difference greater then the maximum difference will always be allowed. If this difference does not matter to us, this variable is valued false.
- `int NR_OF_MAX_DIFF_SUIT_STACK`: the maximum difference between top ranks in different suit stacks. This value is only used if `bool MAX_DIFF_SUIT_STACK` is true.
- `bool greedy_pile_strategy` is valued true if we want to use the GREEDY PILE STRATEGY described in Section 3.2.2.
- `bool point_pile_strategy` is valued true if we want to use the POINT PILE STRATEGY described in Section 3.2.3.

5 Experiments

To perform experiments to investigate certain parts of Klondike Solitaire we use the program explained in Section 4. To be able to determine the importance of a certain rule or part of Klondike Solitaire, we compare the results with a game of Klondike Solitaire played by the basic rules as formulated in Section 2.1. To be able to compare different strategies to each other, all valuations of the end state of a game as explained in Section 3.2 are used.

All experiments are run on a Mac OS X 10.9.4 with a 2,4 GHz Intel Core i5 processor and 4 GB 1333 MHz DDR3 memory.

5.1 Assumptions about game rules and priorities

The game, and everything to it, logically completely depends on the used rules and the associated priorities. If more moves are possible, the probability of winning will improve. But it remains unclear what influence all kinds of variations have on the winning probability. To be able to perform experiments to improve the winning percentage, we make some assumptions about the game rules we use. In this section we will explain certain assumptions we make, and underpin these assumptions with some experimental results. We will take a look at these differences based on the SIMPLEST HEURISTIC STRATEGY as formulated in Section 3.2.1 and we use deal-3 if not stated otherwise. In these first experiments our parameter values are based on [5]: `SCORE_0 = 5`, `SCORE_1 = 0`, `SCORE_2 = 0` and `SCORE_3 = 5`.

Kings move; allowed or not? The permission to fill empty build stacks seems to be a rather important part of the game. The simulated win percentage after 1,000,000 games with the Kings move allowed, equals 25.70%. The simulated win percentage after 1,000,000 games with the Kings move *not* allowed, equals 0.30%. So it follows that allowing the Kings move is very important to the winning rate. If we think about it, this is easy to understand. If a build stack containing a King also contains a card from the same suit and a lower rank below this specific King, the game can never be won! So in all experiments from now on we allow the Kings move.

Way of increasing suit stacks During the game it is tempting to move a card directly to the suit stack, if possible. In the end, all cards have to be moved to the suit stack, so why not perform the move as soon as one has the chance to do so? However, if a lot of cards with high rank are already moved to the associated suit stack, it gets harder to stack cards in a build stack, which reduces the chance of turning face-down cards. Simulation of 1,000,000 games gives the winning percentages if we simulate the maximum difference between ranks in the suit stack in Table 9. We discriminated a deal-1 situation and a deal-3 situation and fixed the variable which indicates the maximum difference between ranks in the suit stacks, `NR_OF_MAX_DIFF_SUIT_STACK`. It appears that in both the deal-1 situation as in the deal-3 situation, it is optimal to give priority to a move to the suit stack if its difference has a maximum difference of 2. So in every experiment from now on we will give priority to moves which keep this maximum difference of 2 intact.

Maximum difference between ranks in suit stacks	Winning %	
	Deal-1	Deal-3
1	37.8%	9.0%
2	40.4%	9.8%
3	37.2%	7.7%
4	36.5%	7.9%
5	30.5%	7.2%
6	28.3%	6.4%
7	28.3%	6.6%
8	28.8%	6.2%
9	28.7%	6.2%
10	26.7%	5.8%
11	27.5%	6.4%
12	27.5%	5.9%

Figure 9: Simulated percentage of winnable games, with a simulated maximum difference between card ranks in the suit stacks.

Number of cards in the pile In the original game, 28 of the cards are placed in build stacks, and the remaining 24 cards are initially placed in the pile. This gives rise to the question why the cards are distributed this way. Simulation gives us Figure 10 and Table 11. This shows a certain periodicity. If we distribute 52 cards in increasing build stacks, this results in build stacks of size 1, 2, 3, 4, 5, 6, 7, 8, 9, 7. If we observe Figure 10 from left to right, the first minimum is achieved when the number of cards in the pile is equal to 7. This number corresponds with the last build stack containing only 7 cards. The second minimum (seen from left to right) occurs when the number of cards in the pile is equal to $7 + 9 = 16$. This number corresponds to the last build stack containing 7 cards *and* the second-to-last build stack containing 9 cards. This explanation holds for all minima in the figure. Inituitively this is clear. An incomplete build stack placed in the pile will result in a higher probability of winning because at least one card is left in the build stack, so that the top card in this build stack can be used to stack cards on. If we had placed all these cards in the pile, this would not have been possible.

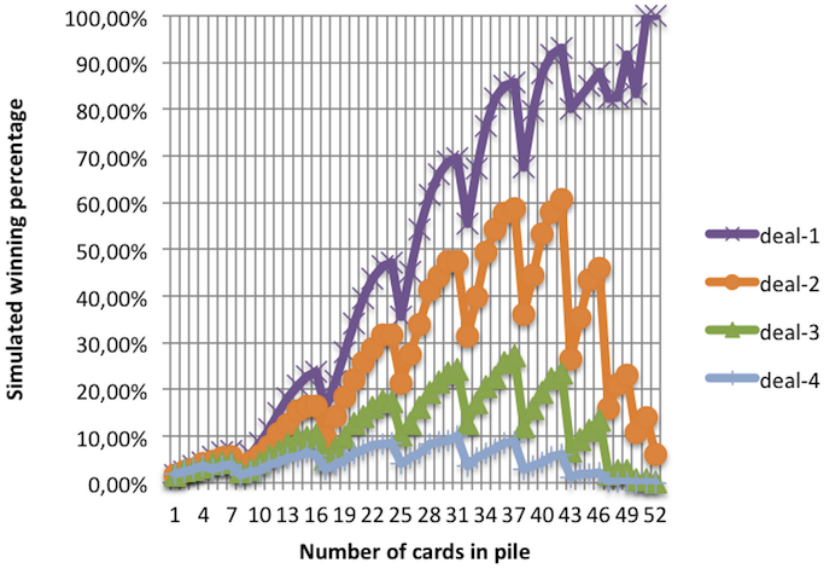

Figure 10: Simulated percentage of winning, increasing number of cards in pile. Based on simulation of 1,000,000 games.

Variant	Winning percentage
deal-1	35.6%
deal-2	21.4%
deal-3	10.2%
deal-4	4.1%

Figure 11: Simulated winning percentage with 24 cards in the pile, based on 1,000,000 games.

5.2 Variations in strategy

In Section 3.2, three types of strategies have been discussed: the SIMPLEST HEURISTIC STRATEGY, the GREEDY PILE STRATEGY and the POINT PILE STRATEGY. Of course there always is the random strategy, in which a random move is carried out. In this subsection we will investigate the different types of strategies and their influence on winning rates. Note that due to reasons explained in Section 5.1 on page 19 we will allow the Kings move and give priority to moves which keep the maximum difference of 2 between ranks in suit stacks intact in all experiments.

5.2.1 Random player

If we simulate 1,000,000 games played by the random player using deal-1, the winning percentage is about 10.41%. The total experiment had a duration of 7.38 minutes. In case of deal-3 the winning percentage is about 3.78%, and the duration of the experiment was 5.00 minutes. From [2] anecdotal evidence suggests that typical human players win around 15% of the games, so we know that a good strategy will definitely increase the winning percentage.

5.2.2 Heuristic player

When a game is played, choices are made based on some sort of strategy. A heuristic player also uses a strategy, as explained in Section 3.2 and Section 4.5. In this section we show the results of all executed strategy experiments.

Experiments for the simplest heuristic player In this paragraph we want to find the best values for the parameters `SCORE_0`, `SCORE_1`, `SCORE_2` and `SCORE_3` which are used for the SIMPLEST HEURISTIC STRATEGY as formulated in Section 3.2.1. In Section 3.2.1 we have used values 5, 0, 0 and 5, but because the strategy is determined by the ratio of the parameters, multiplying them by the same integer results in the same strategy. For this reason we use the parameter values multiplied by 100. To search for a good combination of these scores, we fix `SCORE_0` at 500 and run experiments for all possible combinations with all other score values equal to one of the values `100`, `300`, `500`, `700`, `900` or `1100`. We have tested 216 different score combinations used to play 100,000 games per score combination. This experiment took about 2.84 hours, so using this strategy takes about 0.00047 seconds per game of Solitaire. These experiments led to a first result of the best combination of parameters to lead to a high winning percentage. The combinations resulting in the top ten winning percentages are picked and run again for 1,000,000 games per combination for a better validation of the result. Results are shown in Figure 12.

As displayed in Figure 12, we observe the 10 parameter combinations resulting in the highest winning percentages. The games played with these combinations give a notion of the value of the strategy as explained in Section 3.2. The average

SCORE_1	SCORE_2	SCORE_3	winning percentage based on 100,000 games	winning percentage based on 1,000,000 games
100	100	100	9.84%	9.84%
100	100	300	9.90%	9.80%
100	100	500	9.84%	9.80%
300	300	100	9.85%	9.83%
300	300	300	9.84%	9.88%
700	100	100	9.84%	9.75%
700	100	300	9.93%	9.73%
700	300	100	9.86%	9.72%
900	100	500	9.84%	9.73%
1100	100	300	9.90%	9.76%

Figure 12: Parameter combinations with `SCORE_0 = 500` resulting in top 10 highest winning percentages after playing 100,000 games and associated winning percentages for playing 1,000,000 games with the SIMPLEST HEURISTIC STRATEGY.

values of these results are displayed in Figure 13. The average number of cards on the suit stack in the end state of the game is represented in Figure 14.

If the game is lost	
Number of moves	25.11
Number of cards in pile	12.40
Number of closed cards	12.11
Number of cards in suit stack	2.15
If the game is won	
Winning percentage	9.78%
Number of moves	95.30
General	
Time per game	0.00047 s

Figure 13: Strategy validation properties of the 10 best parameter combinations determining the SIMPLEST HEURISTIC STRATEGY.

Based on these experiments, the parameter combination of `SCORE_0 = 500`, `SCORE_1 = 300`, `SCORE_2 = 300` and `SCORE_3 = 300` has the best winning percentage. This is remarkable because this result implies that only a move from a build stack to a suit stack is more favorable over all other kind of moves.

Experiments using a greedy pile strategy In this paragraph we want to find the best values for the parameters `SCORE_0`, `SCORE_1`, `SCORE_NOT_PILE` and `SCORE_PILE`, which are used for the GREEDY PILE STRATEGY, as formulated in

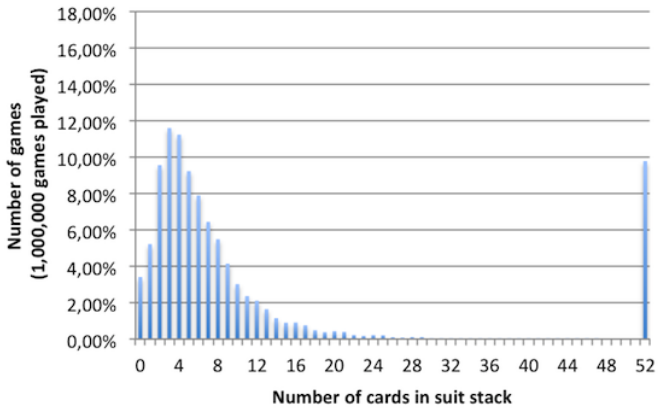

Figure 14: Average number of cards in suit stack at the end state. The SIMPLEST HEURISTIC STRATEGY is used with parameter values equal to the top 10 winning combinations.

Section 3.2.2. To search for a good combination of these scores, we fix `SCORE_0` at 500 and vary all other score values. We have tested 216 different combinations as shown in Figure 15. These are used to play 10,000 games per score combination. This experiment took about 9.09 hours, so using this strategy takes 0.01516 seconds per game.

SCORE_1	SCORE_NOT_PILE	SCORE_PILE
100	1	1
300	2	2
500	3	3
700	4	4
900	5	5
1100	6	6

Figure 15: Used score combinations of parameters with `SCORE_0 = 500` for the GREEDY PILE STRATEGY.

These experiments led to a first result of the best combination of parameters to lead to a high winning percentage. The combinations resulting in the top ten winning percentages are picked and run again for 1,000,000 games per combination for a better validation of the result. Results are shown in Figure 16. As displayed in Figure 16, we observe the 10 parameter combinations resulting in the highest winning percentages. The games played with these combinations give a notion of the value of the strategy as explained in Section 3.2. The average values of these results are displayed in Figure 17. The average number of cards on the suit stack in the end state of the game is represented in Figure 18.

SCORE_1	SCORE_ NOT_PILE	SCORE_ PILE	winning percentage based on 10,000 games	winning percentage based on 1,000,000 games
300	3	1	17.58%	17.76%
500	5	4	17.65%	17.76%
1100	4	1	17.62%	17.73%
500	6	2	18.01%	17.68%
700	4	1	18.01%	17.66%
900	6	3	17.84%	17.65%
900	4	2	17.92%	17.62%
900	2	1	17.28%	17.62%
500	5	3	18.04%	17.59%
500	3	2	17.88%	17.57%

Figure 16: Parameter combinations with `SCORE_0 = 500` resulting in top 10 highest winning percentages after playing 10,000 games and associated winning percentages for playing 1,000,000 games with the GREEDY PILE STRATEGY.

If the game is lost	
Number of moves	30.02
Number of cards in pile	10.05
Number of closed cards	10.68
Number of cards in suit stack	3.80
If the game is won	
Winning percentage	17.76%
Number of moves	93.86
General	
Time per game	0.015 s

Figure 17: Strategy validation properties of the 10 best parameter combinations determining the GREEDY PILE STRATEGY.

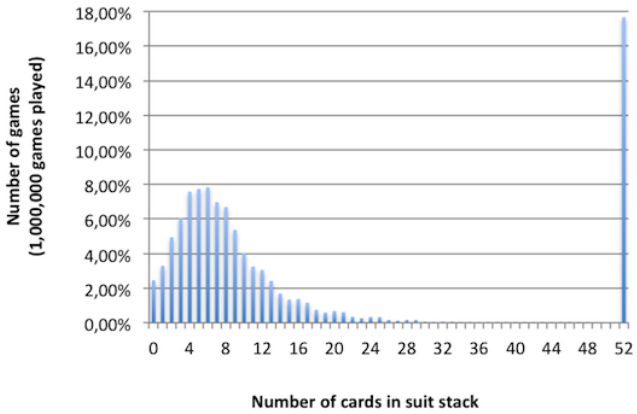

Figure 18: Average number of cards in suit stack at the end state. The GREEDY PILE STRATEGY is used with parameter values equal to the top 10 winning combinations.

Based on these experiments, the parameter combination of `SCORE_0 = 500`, `SCORE_1 = 300`, `SCORE_NOT_PILE = 3` and `SCORE_PILE = 1` has the best winning percentage.

Experiments using a point based pile strategy In this paragraph we want to find the best values for the parameters `SCORE_0`, `SCORE_1`, `SCORE_FIRST_PILE`, `SCORE_SECOND_PILE`, `NEW_BUILD_MOVE` and `NR_SUIT_MOVES`, which are used for the POINT PILE STRATEGY as formulated in Section 3.2.3. To search for a good combination of these scores, we fix `SCORE_0` at 50 and vary all other score values. We have tested 3125 different combinations as shown in Figure 19. These are used to play 1,500 games per score combination. This experiment took about 7.46 hours, so using this strategy takes 0.0057 seconds per game.

SCORE_1	SCORE_ FIRST_PILE	SCORE_ SECOND_PILE	NEW BUILD_MOVE	NR_ SUIT_MOVE
25	10	10	10	10
50	20	20	20	20
75	30	30	30	30
100	40	40	40	40
125	50	50	50	50

Figure 19: Used score combinations of parameters with `SCORE_0 = 50` for the POINT PILE STRATEGY.

These experiments led to a first result of the best combination of parameters to lead to the highest winning percentage. The combinations resulting in the top

ten winning percentages are shown in Figure 20.

SCORE_1	SCORE_ FIRST_ PILE	SCORE_ SECOND_ PILE	NEW BUILD_ MOVE	NR_ SUIT_ MOVE	winning percentage based on 1,500 games
125	10	10	10	10	14.13%
125	10	10	30	10	13.53%
125	10	20	10	10	13.53%
100	10	10	10	10	13.33%
125	10	10	10	20	13.27%
125	10	10	20	20	13.20%
125	20	10	10	10	12.93%
125	10	30	10	10	12.87%
125	20	20	10	10	12.80%
125	10	10	10	20	12.20%

Figure 20: Parameter combinations with `SCORE_0 = 50` resulting in top 10 highest winning percentages after playing 1,500 games with the POINT PILE STRATEGY.

Some notable things in these results are the following:

- in 0.02% of the cases, the value of `SCORE_1` is greater than or equal to 100.
- the values of all other variables are almost always around the value of 10 for the top winning percentages.

From this observations we choose to do another experiment with a fixed value of `SCORE_0` = 50, `SCORE_1` =125, and varying the other score parameters with slighter changes around the value of 10, resulting in the top 10 highest winning percentages displayed in Figure 21. As displayed in Figure 21, we observe the 10 parameter combinations resulting in the highest winning percentages. The games played with these combinations give a notion of the value of the strategy as explained in Section 3.2. The average values of these results are displayed in Figure 22. The average number of cards on the suit stack in the end state of the game is represented in Figure 23.

Based on these experiments, the parameter combination of `SCORE_0 = 50`, `SCORE_1 = 125`, `SCORE_FIRST_PILE = 6`, `SCORE_SECOND_PILE = 6`, `NEW_BUILD_MOVE = 8` and `NR_SUIT_MOVE = 8` have the best winning percentage.

SCORE_ FIRST_ PILE	SCORE_ SECOND_ PILE	NEW BUILD_ MOVE	NR_ SUIT_ MOVE	winning percentage based on 1,500 games	winning percentage based on 100,000 games
6	6	8	8	15.73%	14.10%
6	10	12	6	15.73%	14.00%
6	8	8	8	15.73%	13.92%
6	12	12	6	16.13%	13.87%
10	6	8	8	15.53%	13.78%
8	8	8	10	15.67%	13.71%
8	12	6	6	16.47%	13.63%
8	10	6	8	15.93%	13.63%
14	10	12	8	15.73%	13.44%
10	14	6	8	16.07%	13.38%

Figure 21: Parameter combinations with `SCORE_0 = 50` and `SCORE_1 = 125` resulting in top 10 highest winning percentages after playing 1,500 games and associated winning percentages for playing 1,000,000 games with the POINT PILE STRATEGY.

If the game is lost	
Number of moves	28.52
Number of cards in pile	10.99
Number of closed cards	11.99
Number of cards in suit stack	3.21
If the game is won	
Winning percentage	13.75%
Number of moves	86.64
General	
Time per game	0.0057 s

Figure 22: Strategy validation properties of the 10 best parameter combinations determining the POINT PILE STRATEGY.

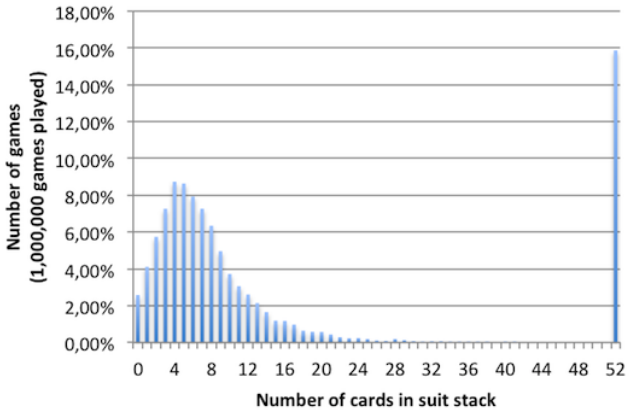

Figure 23: Average number of cards in suit stack at the end state. The POINT PILE STRATEGY is used with parameter values equal to the top 10 winning combinations.

5.3 Summary of results

A summary of the results of playing a game of Klondike Solitaire with one of the three discussed strategies is shown in Figure 24. In this figure a few statistics are striking. In this section the differences between strategies are discussed.

	SIMPLEST H.	GREEDY PILE	POINT PILE
If the game is lost			
Number of moves	25.11	30.02	28.52
Number of cards in pile	12.40	10.05	10.99
Number of closed cards	12.11	10.68	11.99
Number of cards in suit stack	2.15	3.80	3.21
If the game is won			
Winning percentage	9.78%	17.76%	13.75%
Number of moves	95.30	93.86	86.64
General			
Time per game	0.00047 s	0.015 s	0.0057 s

Figure 24: Properties of every tested strategy.

The GREEDY PILE STRATEGY has the highest winning percentage. If you change your strategy from SIMPLEST HEURISTIC to GREEDY PILE, the winning rate improves 81.59%, but the time needed to accomplish a win increases with a factor 32. Changing strategy from SIMPLEST HEURISTIC to POINT PILE, leads to a winning rate improvement of 40.59% by having the time increase with a factor 12.

A striking difference is that the POINT PILE STRATEGY wins a game using less card moves. This is explained by the fact that the GREEDY PILE STRATEGY prefers game states in which more cards are present in build stacks over more cards present in the pile. Meaning that the GREEDY PILE STRATEGY tends to move cards more often from the pile to a build stack, compared to the POINT PILE STRATEGY.

The experiments bring forward a very remarkable aspect of a lost game of Klondike Solitaire. As we can see in Figure 24, a game of Solitaire gets stuck in a situation where the average amount of closed cards is very close to the average amount of cards in the pile.

5.3.1 Comparison of thesis results to literature results

While all experiments in this thesis are based on unThoughtful Solitaire, all results in the literature, except from results in [2], are based on Thoughtful Solitaire. The difference between these to variants is described in Section 2.2.5.

The heuristic player (a player following a heuristic) for Thoughtful Solitaire, described in [5], can supposedly win approximately 13.05% of the games with an average time of 0.021 seconds per game. With a rollout strategy (a rollout strategy computes an action that would result from an iteration of policy improvement, applied to the heuristic policy; it may be considered an alternative heuristic that improves the original), this percentage gains to an alleged 70.20%. This strategy takes an average 1 hour and 45 minutes per game. All strategies discussed in this thesis take fewer time per game than the heuristic player from [5] and all strategies, except the SIMPLEST HEURISTIC STRATEGY, have a higher winning percentage than the the heuristic player from [5].

The heuristic player from [1] achieves a winning percentage of 16.17% in an average time of 0.02 seconds per game. This result is quite close to the results of our GREEDY PILE STRATEGY.

The solving methods used for Thoughtful Solitaire can be used to solve Klondike Solitaire. Monte-Carlo techniques are applied to the random probability distribution of the closed cards in Klondike Solitaire. Such a solver is used in [2], where a claimed percentage of 36.97% of the games of Klondike Solitaire can be won. This strategy takes on average 2280.55 seconds per game, which is much higher than the duration of our strategies. The quickest strategy in [2] takes on average 44.50 seconds per game and has a winning percentage of 24.64% which is a winning percentage none of our three strategies comes close to.

6 Conclusions and Further Research

We have analyzed three different strategies for playing a game of Klondike Solitaire, two of which have a focus on pile moves. These pile moves appear to be important since games get stuck with the average amount of closed cards and the average amount of cards in the pile very close to each other.

The GREEDY PILE STRATEGY has a partial Monte-Carlo like greedy approach for considering pile moves, aside from a simple heuristic approach for other kind of moves. Having a different kind of approach, for different types of moves appear to be useful since this strategy leads to a winning percentage of 17.76% in only 0.015 seconds. Such a high winning percentage in so little time is remarkable.

A suggestion for future work is that it could be interesting to further split these pile moves into different types of moves and search for a different strategy for each type of move. As we have seen in this thesis, pile moves are of very high importance in a game of Klondike Solitaire, but this GREEDY PILE STRATEGY is only a start.

References

[1] R. Bjarnason, P. Tadepalli and A. Fern. Searching Solitaire in real time. *International Computer Games Association Journal*, 30:131–142, 2007.

[2] R. Bjarnason, P. Tadepalli and A. Fern. Lower bounding Klondike Solitaire with Monte-Carlo planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 26–33, 2009.

[3] Brainium Studios LLC. Mobile Solitaire Application. `http://www.brainiumstudios.com/app/solitaire`, accessed December 2014.

[4] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samathrakis and S. Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[5] P. Diaconis, X. Yan, P. Rusmevichientong and B. Van Roy. Solitaire: Man Versus Machine. *Proceedings Advances in Neural Information Processing Systems*, 17:1553–1560, 2004.

[6] P.B. Donkersteeg. Klondike strategies using Monte Carlo techniques. Master's thesis, Universiteit Leiden, 2010.

[7] U. Latif. The Probability of Unplayable Solitaire (Klondike) Games. `http://www.techuser.net/klondikeprob.html`, accessed December 2014.

[8] L. Longpré and P. McKenzie. The Complexity of Solitaire. *Theoretical Computer Science*, 410:5252–5260, 2009.

[9] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[10] D. Parlett. *A History of Card Games*. Oxford University Press, 1991.

[11] D. Parlett. Patience and playing-card Solitaires. `http://www.davpar.eu/histocs/patience.html`, accessed December 2014.

[12] J.N. van Rijn. Playing games; The complexity of Klondike, Mahjong, Nonograms and Animal Chess. Master's thesis, Universiteit Leiden, 2012.

[13] J. de Ruiter. Counting classes of Klondike Solitaire configurations. Master's thesis, Universiteit Leiden, 2012.