DAT530
Discrete Simulation and Performance Analysis
Final Project
Solitaire game strategy

Racin W. Nygaard

Universitetet i Stavanger

Abstract. This project is such and such... +++

Table of Contents

List of Figures

List of Tables

1 Introduction

This project aims to study the popular card game, Solitaire[Site]. Solitaire is bundled with most Windows[Site] installations, as well as being available for free on several sources. It is also easy to play the game with a physical card deck. A detailed explaination of the games rules can be found in the next chapter, Solitaire Rules[REF]
Since the game utilizes all 52 cards of the deck, the number of possible initial game states is 52!, which is a very high number. A large number of these inital game states can be merged, as they offer no difference in the difficulty to solve. Some of these intial states are unsolvable, but even given a solvable game state, one often find oneself in an unsolvable game state, due to certain actions in the game are non-reversible,. There has been attempts to find the distribution of solvable and unsolvable initial game states [ref]. This is roughly 75 percent are solvable, however the study also shows that only 35 percent of the games are won by an experienced player.
This project contains a complete model of the game, a GUI to play the game, and a basic bot to simulate user actions.

1.1 Solitaire Rules

1.2 Abbreviations

Finite State Machine ?

2 Method and Design

2.1 Naming Policy

2.2 Overall Design

The model developed is pretty large, and contains 94 transition and 42 places. It is developed using the modualar approach, and encompasses 6 different modules. Some of the modules are duplicated, with the only difference being the names of the transitions and places.
The model developed is pretty large, and contains 94 transition and 42 places. It is developed using the modualar approach, and encompasses 6 different modules. Some of the modules are duplicated, with the only difference being the names of the transitions and places.

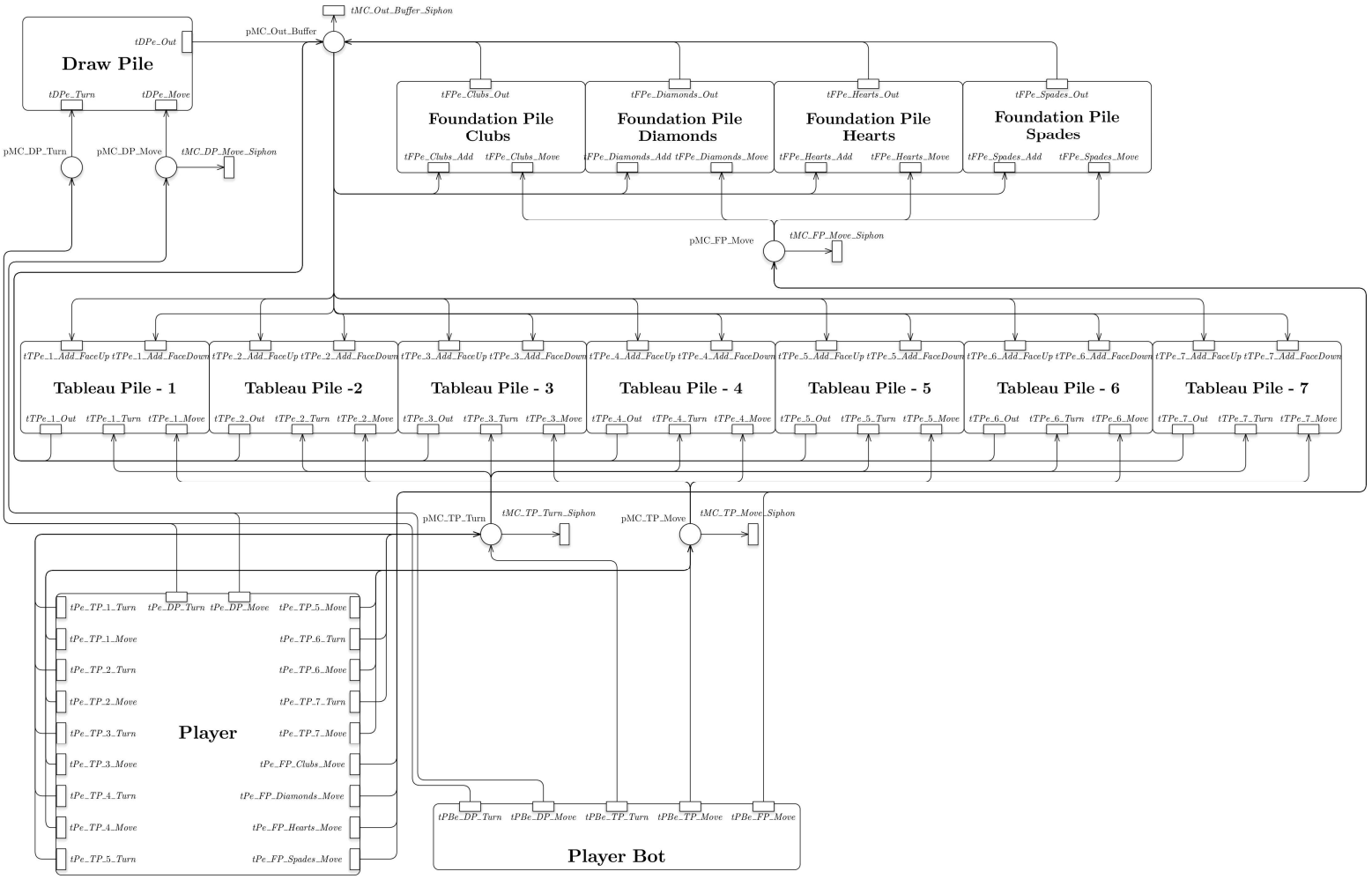

Fig. 1. The complete model - Without the internal components of the modules.

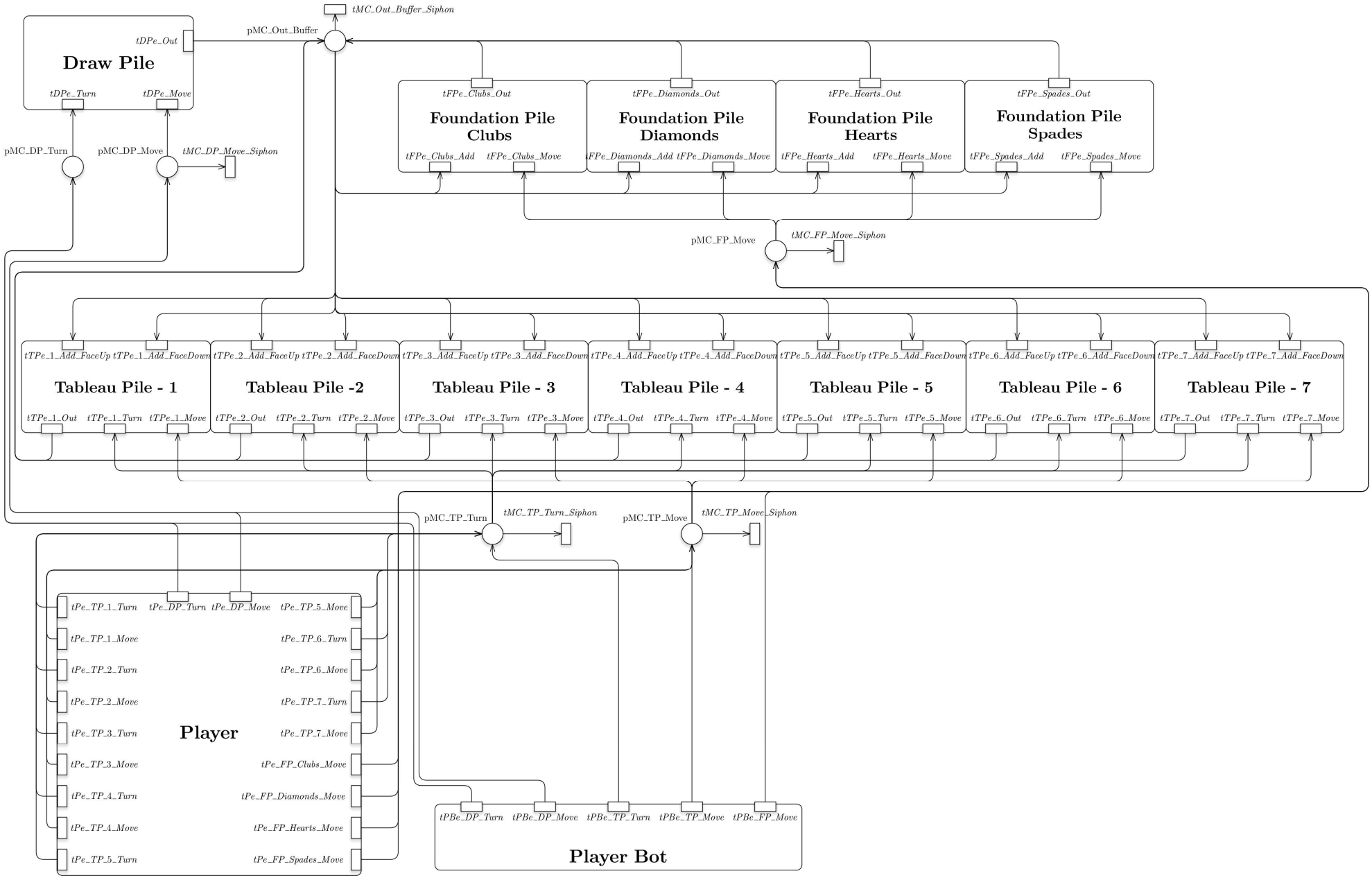

Fig. 2. The complete model - Without the internal components of the modules.

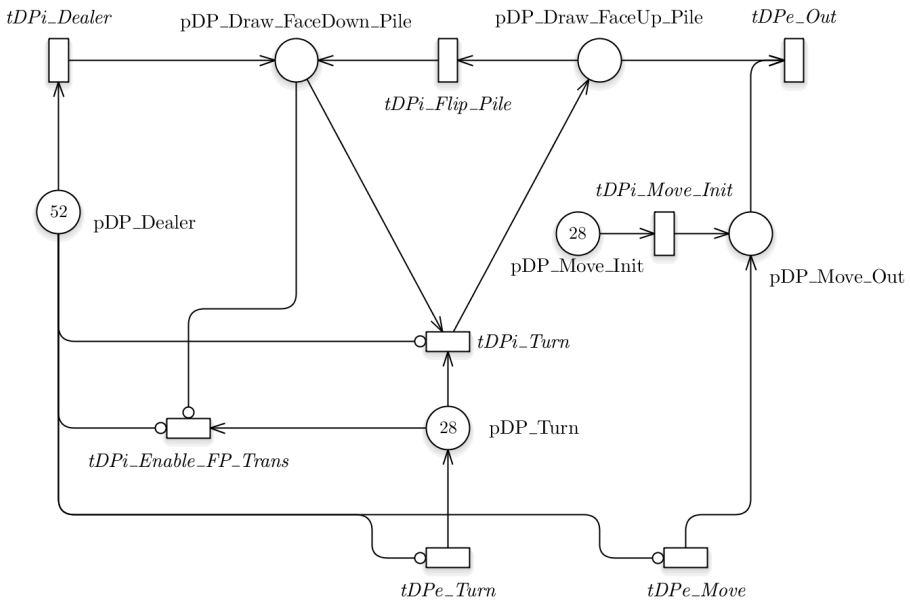

Fig. 3. Draw Pile Module

2.3 Draw Pile Module

The Draw Pile module has several key responsibilities. When first running the model, the initial tokens of `pDP_Dealer` will be sent to `tDPi_Dealer`. This transition will give each token a color which represents a card in the deck. Possible colors are initially stored in the cell `global_info.DECK`. If `global_info.RANDOM_DECK` is set, a random permutation of the colors will be given to the tokens. By having `global_info.RANDOM_DECK` set to false, it is possible to run analytics which require that the cards are dealt equally each time.

After all tokens are given a color, `tDPi_Turn` will be enabled. This transition will move cards from the pile which represents face-down cards, `pDP_Draw_FaceDown_Pile` to the one representing face-up cards, `pDP_Draw_FaceUp_Pile`. This transition will fire as many times as the length of `global_info.INITIAL_DEAL_MOVE`, which is 28 in a normal game. This is not something that would be done if the game where played with physical cards, as they would just be dealt without turning them. In this model however, this is required so that existing logic could be re-used.

Concurrently to the firing of `tDPi_Turn`, the transition `tDPi_Move_Init` will fire an equal amount of times. The transition will give each of the tokens in `pDP_Move_Init` a color which represents to which tableau pile the card

should be moved to. The color given to each token is augmented by the cell, `global_info.INITIAL_DEAL_MOVE`. An example of a color given is *Move:TP1:DP* which means; *Moving a card from source DP to destination TP1.*

Table 1. Transitions used in Draw Pile

	Name	Description
1	tDPe_Move	
2	tDPe_Out	
3	tDPe_Turn	
4	tDPi_Dealer	
5	tDPi_Enable_FP_Trans	
6	tDPi_Flip_Pile	
7	tDPi_Move_Init	
8	tDPi_Turn	

Table 2. Places used in Draw Pile

	Name	Description
1	pDP_Dealer	
2	pDP_Draw_FaceDown_Pile	
3	pDP_Draw_FaceUp_Pile	
4	pDP_Move_Init	
5	pDP_Move_Out	
6	pDP_Turn	

2.4 Foundation Pile Module

2.5 Tableau Pile Module

2.6 Module Connector Module

2.7 Player Module

2.8 Player Bot Module

3 Implementation

3.1 Algorithms

Atomicity In order to preventdd

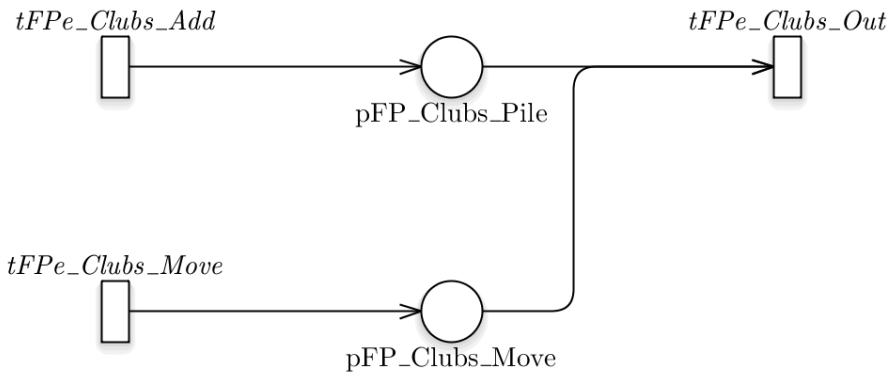

Fig. 4. Foundation Pile Module

Table 3. Transitions used in Foundation Piles

	Name	Description
9	tFPe_Clubs_Add	
10	tFPe_Clubs_Move	
11	tFPe_Clubs_Out	
12	tFPe_Diamonds_Add	
13	tFPe_Diamonds_Move	
14	tFPe_Diamonds_Out	
15	tFPe_Hearts_Add	
16	tFPe_Hearts_Move	
17	tFPe_Hearts_Out	
18	tFPe_Spades_Add	
19	tFPe_Spades_Move	
20	tFPe_Spades_Out	

Table 4. Places used in Foundation Piles

	Name	Description
7	pFP_Clubs_Move	
8	pFP_Clubs_Pile	
9	pFP_Diamonds_Move	
10	pFP_Diamonds_Pile	
11	pFP_Hearts_Move	
12	pFP_Hearts_Pile	
13	pFP_Spades_Move	
14	pFP_Spades_Pile	

Table 5. Transitions used in Tableau Piles

	Name	Description
53	tTPe_1_Add_FaceDown	
54	tTPe_1_Add_FaceUp	
55	tTPe_1_Move	
56	tTPe_1_Out	
57	tTPe_1_Turn	
58	tTPe_2_Add_FaceDown	
59	tTPe_2_Add_FaceUp	
60	tTPe_2_Move	
61	tTPe_2_Out	
62	tTPe_2_Turn	
63	tTPe_3_Add_FaceDown	
64	tTPe_3_Add_FaceUp	
65	tTPe_3_Move	
66	tTPe_3_Out	
67	tTPe_3_Turn	
68	tTPe_4_Add_FaceDown	
69	tTPe_4_Add_FaceUp	
70	tTPe_4_Move	
71	tTPe_4_Out	
72	tTPe_4_Turn	
73	tTPe_5_Add_FaceDown	
74	tTPe_5_Add_FaceUp	
75	tTPe_5_Move	
76	tTPe_5_Out	
77	tTPe_5_Turn	
78	tTPe_6_Add_FaceDown	
79	tTPe_6_Add_FaceUp	
80	tTPe_6_Move	
81	tTPe_6_Out	
82	tTPe_6_Turn	
83	tTPe_7_Add_FaceDown	
84	tTPe_7_Add_FaceUp	
85	tTPe_7_Move	
86	tTPe_7_Out	
87	tTPe_7_Turn	
88	tTPi_1_Move_Multiple	
89	tTPi_2_Move_Multiple	
90	tTPi_3_Move_Multiple	
91	tTPi_4_Move_Multiple	
92	tTPi_5_Move_Multiple	
93	tTPi_6_Move_Multiple	
94	tTPi_7_Move_Multiple	

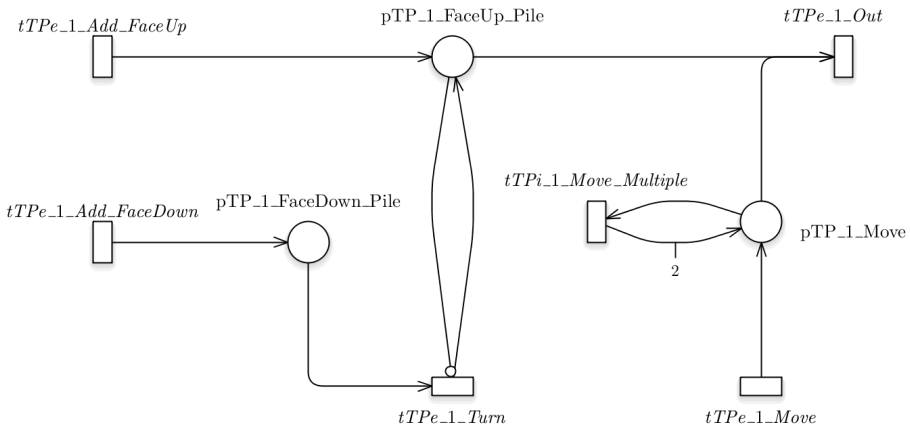

Fig. 5. Tableau Pile Module

Table 6. Places used in Tableau Piles

	Name	Description
22	pTP_1_FaceDown_Pile	
23	pTP_1_FaceUp_Pile	
24	pTP_1_Move	
25	pTP_2_FaceDown_Pile	
26	pTP_2_FaceUp_Pile	
27	pTP_2_Move	
28	pTP_3_FaceDown_Pile	
29	pTP_3_FaceUp_Pile	
30	pTP_3_Move	
31	pTP_4_FaceDown_Pile	
32	pTP_4_FaceUp_Pile	
33	pTP_4_Move	
34	pTP_5_FaceDown_Pile	
35	pTP_5_FaceUp_Pile	
36	pTP_5_Move	
37	pTP_6_FaceDown_Pile	
38	pTP_6_FaceUp_Pile	
39	pTP_6_Move	
40	pTP_7_FaceDown_Pile	
41	pTP_7_FaceUp_Pile	
42	pTP_7_Move	

3.2 Initial Dealing

3.3 Resources

3.4 Moving Multiple Cards

```
def mapper_from_to(self, key, email):
    if 'to' in email.keys() and 'from' in email.keys() and 'body_count' in email.key
```

4 Testing, Analysis and Results

4.1 Algorithms

Atomicity In order to preventdd

4.2 Initial Dealing

4.3 Resources

4.4 Moving Multiple Cards

5 Discussion

References

1. Wikipedia article on Tf-idf. `https://en.wikipedia.org/wiki/Tf?idf`
2. Tom White, Hadoop: The Definitive Guide, 2015, *ISBN: 978-1-491-90163-2*
3. Docker API Docs, `https://docs.docker.com`
4. Slides from DAT630, Krisztian Balog
5. Kaggle. The Enron Email Dataset. `https://www.kaggle.com/wcukierski/enron-email-dataset`
6. Data Intensive Systems Compendium, Tomasz Wiktorski et al.
7. Source code of all tasks developed. GitLab `https://gitlab.com/mindejulian/projectDAT500/tree/master`