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

Abbreviations

DP Draw Pile Module
FIFO First In First Out (Queue)
FP Foundation Pile Module
GUI Graphical User Interface
LIFO Last In First Out (Stack)
P Player Module
PB Player Bot Module
TP Tableau Pile Module

Nomenclature

card (In the Petri Net context) A token with a color which represents a card in the deck.
command A token with a color which represents a turn or movement command.

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 explanation 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 initial 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

Patience games are assumed to have originated in Germany or Scandinavia in the 18th century. Klondike, or Solitaire as it has been called in North America is one of the most popular in this family of games.

The game became very popular in the 19th century and the name Klondike is believed to have originated from the prospectors that were mining for gold in the Klondike region in Canada, and whom have either created or promoted this game while on their quest. Solitaire is played with a standard deck of 52 cards and no Jokers.

The deck is shuffled and the cards are laid out in seven piles from left to right. There is one upturned card at the beginning of each pile. The first pile to the left contains one card, the second pile contains two cards and so on up to the seventh pile which contains seven cards. There is one upturned card on each pile - the first and left-most pile is comprised therefore only of the upturned card. The piles are illustrated in figure...

2 Method and Design

2.1 Naming Policy

2.2 File structure

To reduce the number of files, most of the pre- and post-processor files of the FP and TP modules have been combined in one single file. An example of this can be shown in listing 1.1, which shows parts of `COMMON_PRE`

Listing 1.1. COMMON_PRE.m lines 1-5

```
1 function [fire, transition] = COMMON_PRE(transition)
2
3 if ismember(transition.name, {'tFPe_Clubs_Add', 'tFPe_Diamonds_Add', ...
4         'tFPe_Hearts_Add', 'tFPe_Spades_Add'}),
5     [fire, transition] = pre_tFPe_Add(transition);
```

By doing this it is possible to reduce the number of files required without overloading the `COMMON_PRE` and `COMMON_POST` files. It also makes it much easier to work and maintain the code as the logic is only located in one place, as opposed to four or seven places if each transition had their own file.

With this approach it is no longer possible to hard-code the names of the related transitions and places, so two additional functions; `get_tableau_from_transname` and `get_suit_from_transname` were developed. These functions take the name of the transition as input, and then return the unique identifier for which module it belongs to. The actual code is pretty simple, and parts of `get_suit_from_transname` is shown in listing 1.2. The reasoning behind not using the Matlab command `contains` is simply that it is not supported in older versions.

Listing 1.2. get_suit_from_transname.m lines 7-17

```
1     if ~isempty(strfind(transitionname,'Clubs')),
2         suit = 'Clubs';
3     elseif ~isempty(strfind(transitionname,'Diamonds')),
4         suit = 'Diamonds';
5     elseif ~isempty(strfind(transitionname,'Hearts')),
6         suit = 'Hearts';
7     elseif ~isempty(strfind(transitionname,'Spades')),
8         suit = 'Spades';
9     else,
10        suit = 0; % Invalid suit.
11    end
```

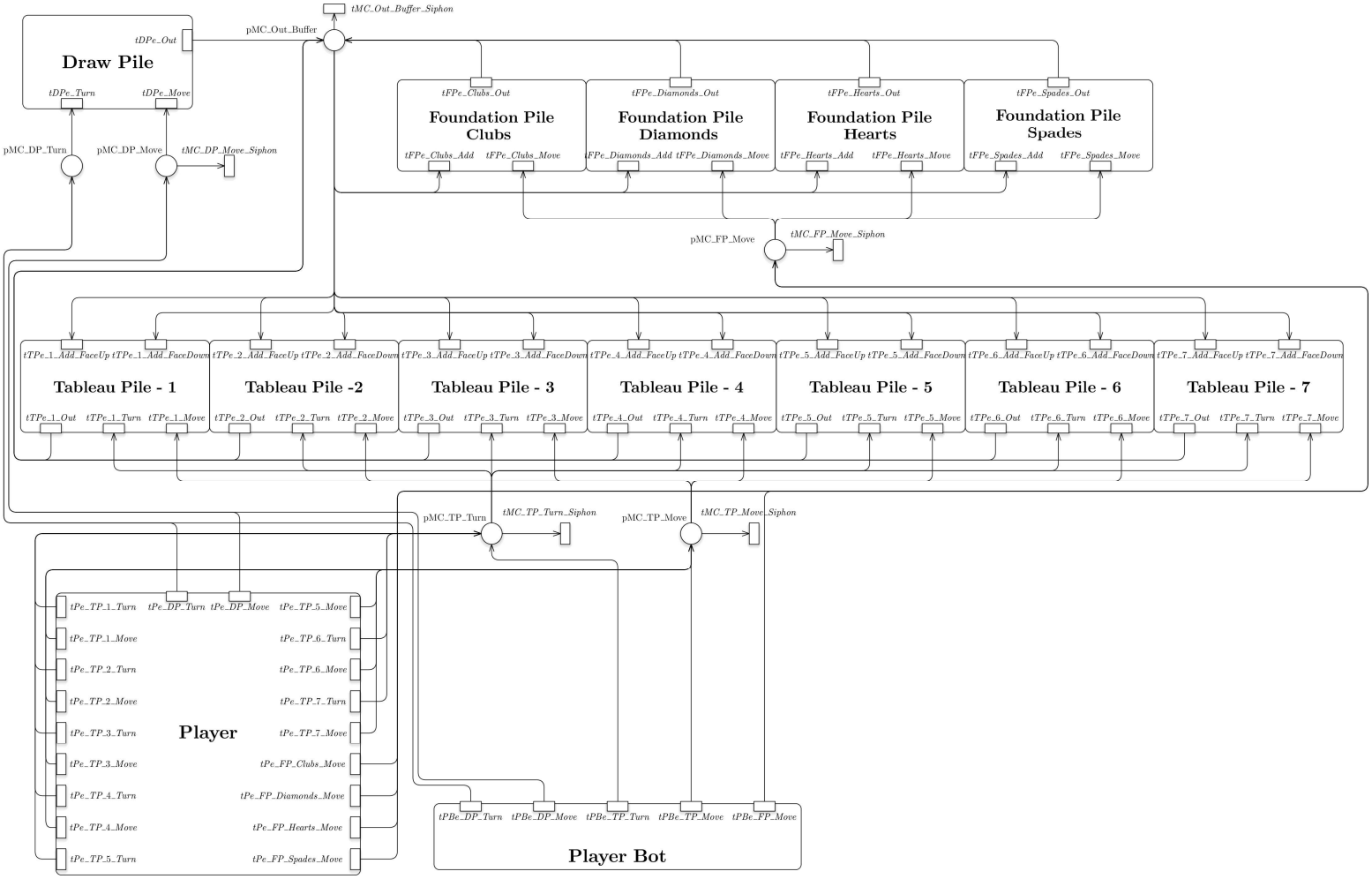

Fig. 1. The complete model - Without the internal components of the modules.

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.

2.4 Draw Pile Module

The Draw Pile module is depicted in figure 2, and has several key responsibilities, once of which is to do the initial dealing of cards. In order to preserve the correctness of the gameplay, external input is not allowed during this phase. When first running the model, all 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`.

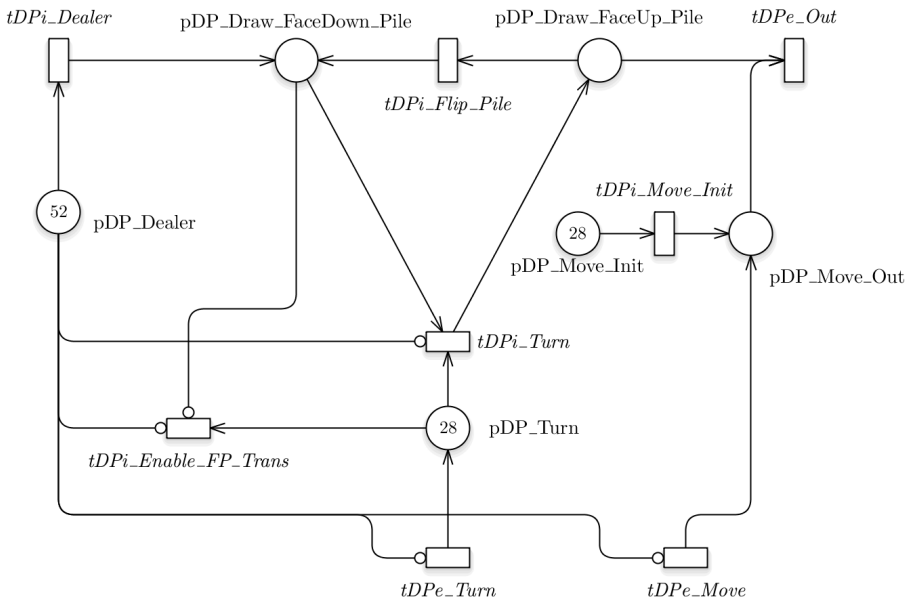

Fig. 2. Draw Pile Module

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.* Every time a card reaches its destined tableau pile, the variable `global_info.CARDS_DEALT` will be incremented by one in `COMMON_POST`. Once it becomes equal to the length

of `global_info.INITIAL_DEAL_MOVE`, the initial dealing phase is over, and the normal phase starts.

During the normal phase, external input is allowed. The first input of the Draw Pile Module is `tDPe_Move`. This transition has an pre-processor file, which makes it only fire if there are tokens in `pDP_Draw_FaceUp_Pile`. Additionally, the Player and Player Bot modules ensures that the enabling token has color on the format *Move:(destination):DP*.

Listing 1.3. tDPe_Move_pre.m

```
function [fire, transition] = tDPe_Move_pre(transition)

fire = 0;
if ~isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
    fire = 1;
end
```

The second input, `tDPe_Turn` is used to used to simply move cards from the face-down pile to the face-up pile during the normal phase. An interesting thing about this is that once all the cards are in the face-up pile, the next time one attempts to turn a card, all cards should be moved back to the face-down pile in LIFO style, just as they would if you simply flip the deck of cards around in real-life.
This is accomplished by the transitions `tDPi_Flip_Pile` and `tDPi_Enable_DP_Trans`. The `tDPi_Enable_FP_Trans` is actually an siphon, and becomes enabled once `pDP_Draw_FaceDown_Pile` is empty, and there is an active turn action on-going so that `pDP_Turn` has at least one token. The transition has one post-processor file, shown in listing 1.4. Given that there are actually any tokens left in `pDP_Draw_FaceUp_Pile` it will set the global flag, `global_info.DP_Flip_Pile_Running` to `true`, if there are no tokens in the face-up pile, it will simply release the `playerAction` resource. The use of resources is discussed further in chapter 3.5. The reason for not having an arc directly from the face-up pile is due to this transition being a siphon, so the card would be removed from the game if it fired.

Listing 1.4. tDPi_Enable_FP_Trans_post.m

```
function [] = tDPi_Enable_FP_Trans_post(transition)

global global_info;
if ~isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
    global_info.DP_Flip_Pile_Running = true;
else,
    % Release playerAction resource to allow for another player action.
    release(global_info.last_command_source);
end;
```

Once `global_info.DP_Flip_Pile_Running` is set to `true` and there are tokens in `pDP_Draw_FaceUp_Pile`, the transition `tDPi_Flip_Pile` will start firing. The pre-processor file is listed in 1.5, and will keep selecting the latest arrived card from `pDP_Draw_FaceUp_Pile` and fire. In the post-processor file, listed in 1.6, it will check for the length of the face-up pile, once it becomes empty it will set the flag `global_info.DP_Flip_Pile_Running` to `false`, and the cards have been successfully turned around.

Listing 1.5. tDPi_Flip_Pile_pre.m

```
function [fire, transition] = tDPi_Flip_Pile_pre(transition)

global global_info;
fire = 0;
if global_info.DP_Flip_Pile_Running == true,
    transition.selected_tokens = tokenArrivedLate('pDP_Draw_FaceUp_Pile',1);
    fire = 1;
end
```

Listing 1.6. tDPi_Flip_Pile_post.m

```
function [] = tDPi_Flip_Pile_post(transition)

global global_info;
if isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
    global_info.DP_Flip_Pile_Running = false;
    global_info.SCORE = max(global_info.SCORE - 100, 0);
    % Release playerAction resource to allow for another player action.
    release(global_info.last_command_source);
end;
```

Lastly, there is the `tDPe_Out` transition. This is the only external output of the module. When enabled, its pre-processor will take the lastest card arrived at `pDP_Draw_FaceUp_Pile`, but the earliest command arrived at `pDP_Move_Out` when firing. By taking the earliest command arrived in a FIFO manner, we ensure that the inital dealing will be correct. If we were to take the latest command, we would have to add additional logic such as alternating firing to make certain the ordering of cards would be correct. The code is shown in lising 1.7

Listing 1.7. tDPe_Out_pre.m

```
function [fire, transition] = tDPe_Out_pre(transition)

% Want to make sure that we get the earliest move-token, and the latest
% card. This is so that we can have a natural ordering of the cards during
% the initial dealing.
moveToken = tokenArrivedEarly('pDP_Move_Out', 1);
% Explicitly sure to get the card at the top of the stack.
cardToken = tokenArrivedLate('pDP_Draw_FaceUp_Pile', 1);

transition.selected_tokens = [moveToken cardToken];
fire = 1;
```

Interestingly, moving cards out of the `tDPe_Out` transition is a non-reversible action as the module has no external input. So by doing this one could potentially put the game in an unsolvable state.

Table 1. Places and transitions used in Draw Pile

	Name	Description
1	pDP_Dealer	Holds the initial tokens which will become cards.
2	pDP_Draw_FaceDown_Pile	Holds the face-down cards. These are not visible to the player.
3	pDP_Draw_FaceUp_Pile	Holde the face-up cards. Only the top card is visible to the player.
4	pDP_Move_Init	Holds initial tokens used for generating move-commands.
5	pDP_Move_Out	Buffer for move-commands.
6	pDP_Turn	Buffer for turn-commands.
7	tDPe_Move	External input for the move-command
8	tDPe_Out	External output
9	tDPe_Turn	External input for the turn-command
10	tDPi_Dealer	Gives every token a color to represent a card in the deck.
11	tDPi_Enable_FP_Trans	Used to facilitate the flipping of the face-up pile.
12	tDPi_Flip_Pile	Moves cards from face-up pile to face-down pile in a LIFO manner.
13	tDPi_Move_Init	Generates initial move-commands to facilitate initial dealing of the cards.
14	tDPi_Turn	Moves a card from the face-down pile to the face-up pile.

2.5 Foundation Pile Module

The Foundation Pile module is depicted figure 3. It is duplicated four times, once for every suit, clubs, diamonds, hearts, and spades. The only difference between these modules is the names of their respective transitions and names, so the description given for clubs will count for the other duplicates as well. All the pre- and post-processor files are shared between all the suits.

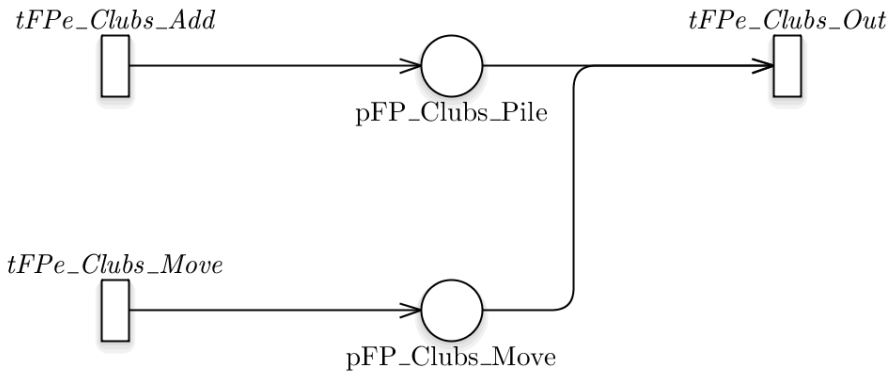

Fig. 3. Foundation Pile Module

This module is inactive during the initial phase, and only becomes interactable once the normal phase starts. It has two external inputs, the first of which is `tFPe_Clubs_Add`. This transition has a shared pre-processor file, `pre_tFPe_Add`. Listing 1.8 shows parts of the logic, the full file can be found in B.19. The pre-processor fetches the token arrived earliest at `pMC_Out_Buffer`. This is an important step, but is not strictly required because of the limitations the resource `playerAction` enforces on the transitions of the Player and Player Bot modules. Still, it makes sense to fetch the earliest token in FIFO style to make sure that the first moved card reaches its destination first.

Given that the colors of the token have the correct length, the `get_suit_from_transname` function will be run to determine which FP the executing transition belongs to. More information about this step can be found in 2.2. Lastly, the `checkCommand_Move` function is ran to determine the validity of the command in the context of this particular transition. The `checkCommand_Move` function is quite involved, and is discussed in detail in chapter 3.3.1.

Listing 1.8. pre_tFPe_Add.m lines 5-17

```
moveToken = tokenArrivedEarly('pMC_Out_Buffer',1);
tokenColor = get_color('pMC_Out_Buffer',moveToken);
if(length(tokenColor) ~= 2),
    return;
end;
[~, suit, handle_err] = get_suit_from_transname(transition.name);
[doCommand, cmdDest, card, cmdSource] = ...
    checkCommand_Move(tokenColor,suit,'',handle_err);
if(doCommand),
    transition.selected_tokens = moveToken;
    transition.new_color = card;
    transition.override = 1;
    fire = 1;
```

The second external input is `tFPe_Clubs_Move`. Its used for moving cards to other modules, and works similarly to how movement in handle in the Draw Pile, with the additional caveat that all four Foundation Piles becomes enabled at the time from `pMC_FP_Move`. Due to this its necessary to introduce additional logic to ensure that the issued move-command from the Player or Player Bot modules are meant for this particular module. As with the other modules, the actual validity of the move-command are handles by the P and PB modules.

Listing 1.9. pre_tFPe_Move.m lines 4-10

```
moveToken = tokenArrivedLate('pMC_FP_Move',1);
[suit_abbr, suit, ~] = get_suit_from_transname(transition.name);
[moveCmd, ~] = splitCommand(get_color('pMC_FP_Move',moveToken));
if(length(moveCmd) >= 3 && strcmp(moveCmd{3},strcat('FP',suit_abbr))),
    transition.selected_tokens = moveToken;
    fire = 1;
end
```

The only external output of the fp is `tFPe_Clubs_Out`. It works similarly to the output of the Draw Pile, where the pre-processor takes the latest arrived card `pFP_Clubs_Pile` and the earliest arrived command from `pFP_Clubs_Move`. This ensures that the first issued command will be processed first, should there be more than one. The only time there would be more than one command executing concurrently is if neither the Player or the Player Bot modules where enabled, and the command was issued from another module which did not use the `playerAction` resource. Listing 1.10 shows parts of the code.

Listing 1.10. pre_tFPe_Out.m lines 4-10

```
[~, suit, ~] = get_suit_from_transname(transition.name);

moveToken = tokenArrivedEarly(strcat('pFP_',suit,'_Move'), 1);
cardToken = tokenArrivedLate(strcat('pFP_',suit,'_Pile'), 1);

transition.selected_tokens = [moveToken cardToken];
fire = 1;
```

Another interesting fact about the Foundation Pile modules is the place `pFP_Clubs_Pile`. Once this place is filled with 13 tokens for all the suits, the game is won, and the simulation ends. There is no check done on the actual color or order of the tokens, as that is done when adding them by the pre-processor of `tFPe_Clubs_Add`. The win condition can be found in `COMMON_POST`. Parts of the code is shown in listing 1.11.

Listing 1.11. COMMON_POST.m lines 31-40

```
% Check if game is won. Win condition: 13 tokens on each of the foundation
```

```
% piles.
if(length(tokIDs('pFP_Clubs_Pile')) == 13 && ...
        length(tokIDs('pFP_Diamonds_Pile')) == 13 && ...
        length(tokIDs('pFP_Hearts_Pile')) == 13 && ...
        length(tokIDs('pFP_Spades_Pile')) == 13),
    set_handle('GameStatus', 'String', 'GAME_WON!');
    disp('GAME_WON!');
    global_info.STOP_SIMULATION = 1;
end
```

Table 2. Places and transitions used in Foundation Pile - Clubs

	Name	Description
1	pFP_Clubs_Move	Buffer for move-commands
2	pFP_Clubs_Pile	Holds the cards which are added to the Foundation Pile.
3	tFPe_Clubs_Add	External input for adding cards to the Foundation Pile.
4	tFPe_Clubs_Move	External input for the move-command.
5	tFPe_Clubs_Out	External output

2.6 Tableau Pile Module

The Tableau Pile module is depicted in figure 4. It is duplicated 7 times, once for every pile in the tableau. The main difference between these modules is the names of their respective transitions and names, so the description given for the first pile will count for the other duplicates as well. Another difference is how many cards each pile are dealt during the initial phase. This is discussed in more detail in chapter 4.3 and 2.4. All the pre- and post-processor files are shared between all piles.

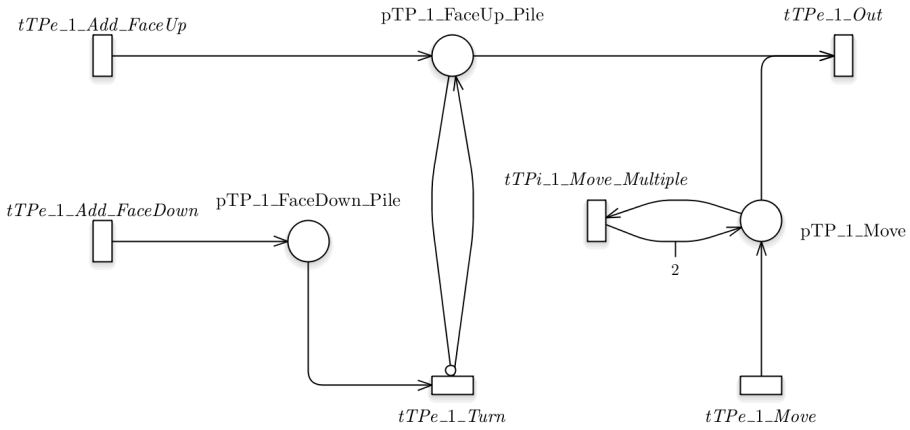

Fig. 4. Tableau Pile Module

As with the Draw Pile module, the module encompasses both a face-down and a face-up pile. The inner workings between these two piles is different however, as moving a card from the face-down pile to the face-up pile is an non-reversible action. Doing so could put the game in an unsolvable state, but at the same time if there were no cards in any of the face-down piles the game would be trivial to solve.

The only time it's possible to add cards to `pTP_1_FaceDown_Pile` is during the initial phase. During this phase, one less than the piles identification number will be added, so that pile one will have zero cards in the face-down pile, whilst pile 7 will have 6. All the piles will have a single card added to `pTP_1_FaceUp_Pile`. This is discussed in more detail in chapter 4.3.

Once the initial phase is over, the pre-processor of `tTPe_1_Add_FaceDown` will prevent any more firings. This is shown in listing 1.12, and is done by simply counting how many cards have been dealt in the variable `global_info.CARDS.DEALT`.

The secondary condition in the if statement is used to control the amount of firings during the initial phase.

Listing 1.12. pre_tTPe_Add_FaceDown.m lines 5-11

```
1 [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
2 % Can only add FaceDown cards during the initial dealing.
3 if global_info.CARDS_DEALT >= global_info.INITIAL_DEAL_MOVE_LENGTH ...
4     || length(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))) + 1 ...
5     == str2double(tableau),
6     return;
7 end;
```

Instead, the `tTPe_1_Add_FaceUp` transition will have potential to fire, given that all of its conditions in the pre-processor is fulfilled. Parts of the code for the pre-processor can be found in listing 1.13, whilst the whole file is found in B.26. As with the Add-transition of FP, it relies on `checkCommand_Move`

Listing 1.13. pre_tTPe_Add_FaceUp.m lines 5-11

```
1 fire = 0;
2
3 % Can only add FaceUp cards once the initial dealing is complete.
4 isFDFull = length(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))) + 1 ...
5     == str2double(tableau);
6 isDealingInProgress = global_info.CARDS_DEALT < ...
7     global_info.INITIAL_DEAL_MOVE_LENGTH;
```

Table 3. Places and transitions used in Tableau Pile - 1

	Name	Description
1	pTP_1_FaceDown_Pile	
2	pTP_1_FaceUp_Pile	
3	pTP_1_Move	
4	tTPe_1_Add_FaceDown	
5	tTPe_1_Add_FaceUp	
6	tTPe_1_Move	
7	tTPe_1_Out	
8	tTPe_1_Turn	
9	tTPi_1_Move_Multiple	

2.7 Module Connector Module

Table 4. Places and transitions used in Module Connector

	Name	Description
1	pMC_DP_Move	
2	pMC_DP_Turn	
3	pMC_FP_Move	
4	pMC_Out_Buffer	
5	pMC_TP_Move	
6	pMC_TP_Turn	
7	tMC_DP_Move_Siphon	
8	tMC_FP_Move_Siphon	
9	tMC_Out_Buffer_Siphon	
10	tMC_TP_Move_Siphon	
11	tMC_TP_Turn_Siphon	

2.8 Player Module

Table 5. Transitions used in Player

	Name	Description
1	tPe_DP_Move	
2	tPe_DP_Turn	
3	tPe_FP_Clubs_Move	
4	tPe_FP_Diamonds_Move	
5	tPe_FP_Hearts_Move	
6	tPe_FP_Spades_Move	
7	tPe_TP_1_Move	
8	tPe_TP_1_Turn	
9	tPe_TP_2_Move	
10	tPe_TP_2_Turn	
11	tPe_TP_3_Move	
12	tPe_TP_3_Turn	
13	tPe_TP_4_Move	
14	tPe_TP_4_Turn	
15	tPe_TP_5_Move	
16	tPe_TP_5_Turn	
17	tPe_TP_6_Move	
18	tPe_TP_6_Turn	
19	tPe_TP_7_Move	
20	tPe_TP_7_Turn	

2.9 Player Bot Module

Table 6. Places and transitions used in Player Bot

	Name	Description
1	pPB_Cmd	
2	tPBe_DP_Move	
3	tPBe_DP_Turn	
4	tPBe_FP_Move	
5	tPBe_TP_Move	
6	tPBe_TP_Turn	
7	tPBi_Gen	
8	tPBi_Siphon	

tPBi_Gen_Stop is deleted. But code is still there. Make comment on why this is removed

3 Implementation

3.1 GUI

3.2 Algorithms

3.2.1 Atomicity

3.3 Commands

3.3.1 Move Command

The move command is contains four parts; the command, destination, source and amount. Each part is concatenated together, with colon as a separator. An example of a move command would be: *Move:TP1:TP5:3*, which means *Move 3 cards from TP1 to TP5*. If amount is not given, it will assume one card to be moved.

In order to make sure that only validity of the move-commands, the function `checkCommand_Move` has been developed. It is used both for validation before sending a command, and validation after receiving a command. The function takes the input parameters; `command`, `destination`, `source`, and `handle_err`.

The input parameter `command` contains the actual command, `destination` contains the unique identifiers of the FP or TP modules. Valid input for `destination` would be *C, D, H, S, 1, 2, 3, 4, 5, 6 or 7*, and is used to ensure that the command is received by the destined module. Parameter `source` is only used when sending a command, and contains the actual name of the transition which issued the game. This is mainly used to set the variable `global_info.last_command_source` which will be the name of the transition holding the `playerAction` resource. Resources are discussed in more detail in chapter 3.5. Lastly, the parameter `handle_err` holds the GUI-component where error mesasges will be written. Full code for the function can be found in the appendix, at chapter B.1.

Figure 5 shows a flowchart of the logic in the function.

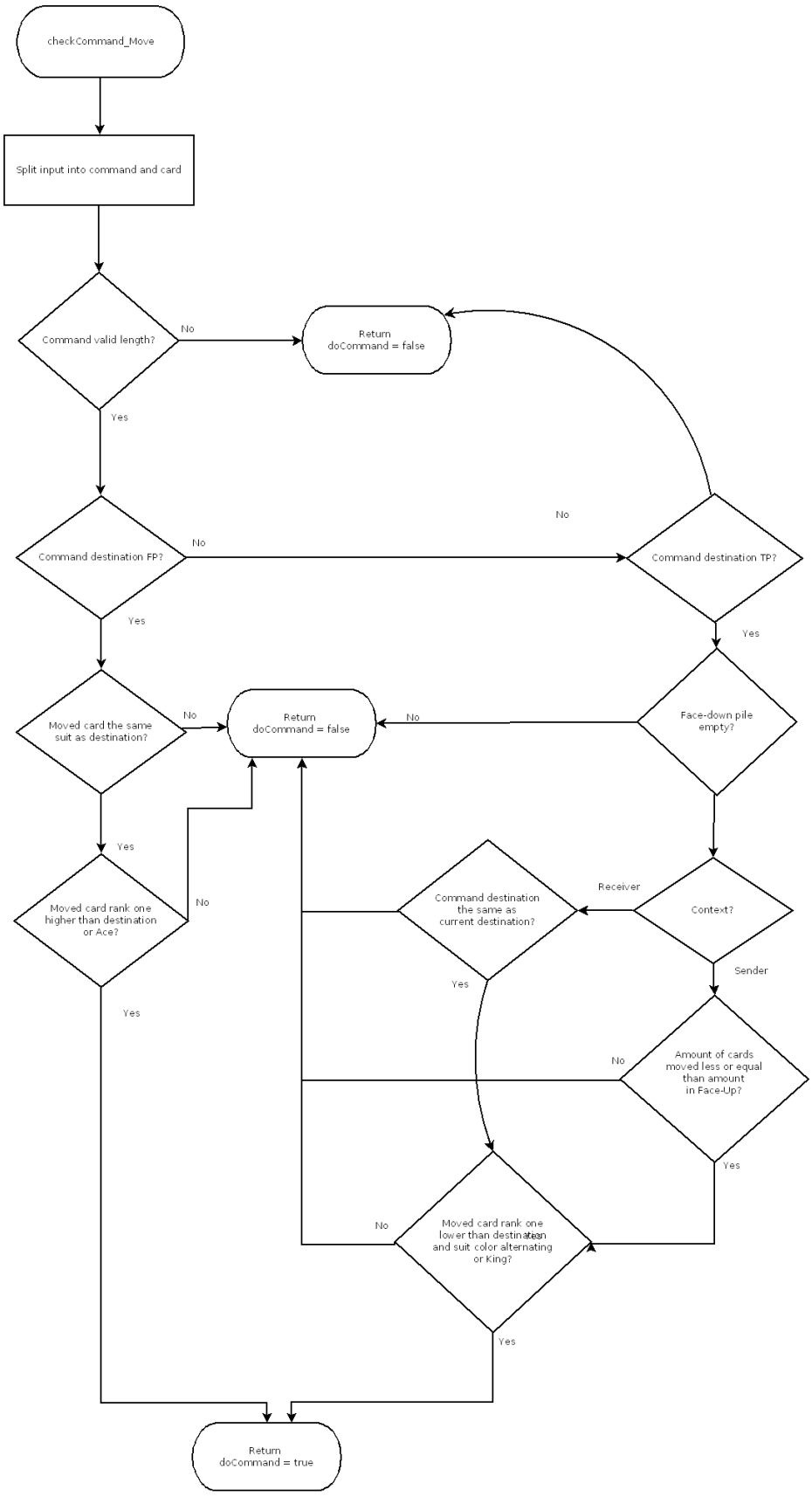

Fig. 5. Flowchart - checkCommand_Move

3.4 Initial Dealing

3.5 Resources

3.6 Moving Multiple Cards

3.7 Scoring

3.8 Possible improvments

A major drawback of the siphon `tMC_Out_Buffer_Siphon` is that if it fires, the card will actually be removed from the game, and the game becomes unsolvable. This transition will fire if the move-command of the token has an invalid destination. Due to how the Player and Player Bot modules are set up, this will never happen as they will check the validity of the move command before actually issuing the command. Still, I think it would be an improvement add an additional transition to the Draw Pile module which would accept cards from `tMC_Out_Buffer_Siphon`, instead of totally discarding them.

Another improvement would be to re-factor the code base by moving more of the validity check of the commands from the Player and Player Bot modules to the destination transitions. The Player Bot modules uses roughly 200 lines of code to always issue valid commands, I think this could be drastically reduced. By doing this it would be easier to create additional modules which could interface with the game, for example a hardware-based module.

It might also be a slight improvement to combine the Add-face-up and Add-face-down transitions of the TP modules. By doing this, it would remove the need for the Add-face-down's pre-processor to execute once the initial phase is over. However, doing so for every Tableau Pile would require 7 more transitions and 14 more places.

3.9 Future work

dladl

4 Testing, Analysis and Results

4.1 Matlab version

The project has been developed and tested in versions R2013b and R2017a. Due to using two versions of Matlab it was necessary to only use functionality that is supported in both versions. Examples of this is using `GUIDE` for developing the GUI, and omitting to use the `contains` command.

4.2 Algorithms

4.2.1 Atomicity

In order to preventdd

4.3 Initial Dealing

4.4 Resources

4.5 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.

A Overall design - horizontal view

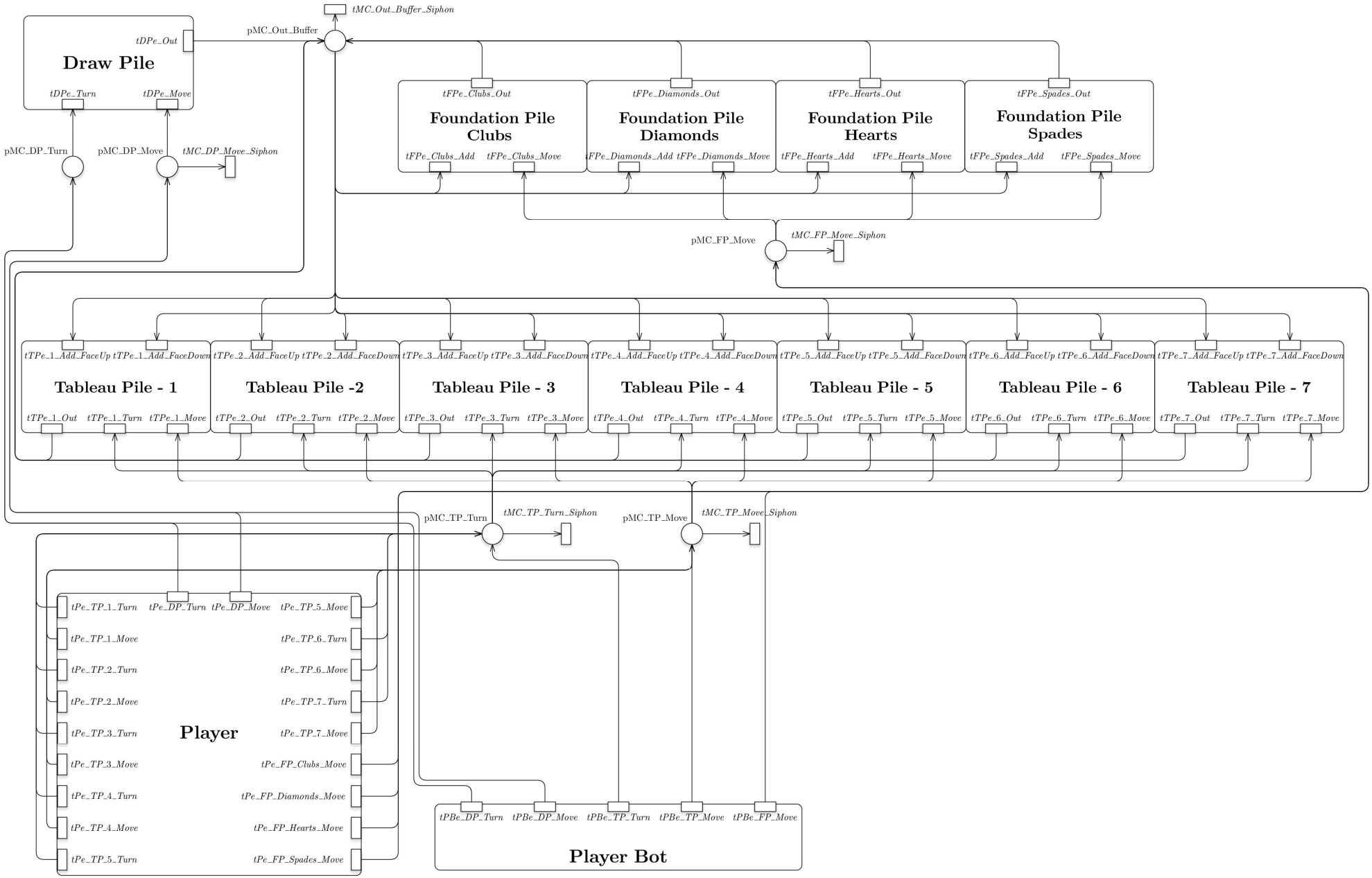

Fig. 6. The complete model in horizontal view

B Matlab code

B.1 checkCommand_Move.m

```
function [ doCommand, cmdDest, card, cmdSource ] = ...
    checkCommand_Move( command, destination, source, handle_err)

global global_info;
[moveCmd, card] = splitCommand(command);
cmdDest = moveCmd{2};
cmdSource = moveCmd{3};

doCommand = false;
if length(cmdDest) < 3,
    set_handle(handle_err,'String','INCOMPLETE_COMMAND');
    return;
elseif ~ismember(cmdDest, global_info.FP_TP_PILES),
    set_handle(handle_err,'String','INVALID_MOVE_COMMAND');
    return;
end

% Foundation Piles
if ismember(cmdDest, global_info.FP_PILES),
    if ~isempty(destination) && destination(1) ~= cmdDest(3),
        return;
    end;
    movedCard_split = strsplit(card,'_');
    moved_suit = movedCard_split(1);
    moved_rank = movedCard_split(2);
    if(~isfield(global_info.SUITS,cmdDest(3))),
        set_handle(handle_err,'String','INVALID_SUIT');
        return;
    end;
    if moved_suit{1} ~= cmdDest(3),
        set_handle(handle_err,'String','INVALID_LOCATION');
        return;
    end;
    global_suit = global_info.SUITS.(cmdDest(3));
    fp_Pile = strcat('pFP_',global_suit(1),'_Pile');
    if(iscell(fp_Pile)),
        fp_Pile = fp_Pile{1};
    end;
    dest_topCard_Id = tokenArrivedLate(fp_Pile,1);
    moved_rank_value = global_info.CARDVALUE_MAP(moved_rank{1});
    if dest_topCard_Id,
        dest_topCard_Color = get_color(fp_Pile,dest_topCard_Id);
        dest_topCard_split = strsplit(dest_topCard_Color{1},'_');
        dest_topCard_Rank = dest_topCard_split(2);
        diffRank = moved_rank_value - global_info.CARDVALUE_MAP(
            dest_topCard_Rank{1});
        if(diffRank ~= 1), % Added card must be 1 value higher than the
            current card.
            set_handle(handle_err,'String','INVALID_CARD_VALUE');
            return;
        end;
    elseif moved_rank_value ~= 1,
        set_handle(handle_err,'String','FIRST_CARD_MUST_BE_ACE');
        return;
    end;
elseif ismember(cmdDest, global_info.TP_PILES),
    tableau_dest = cmdDest(3);

    if ~isempty(destination) == 1 && destination(1) ~= tableau_dest,
        return;
    end;
    movedCard_split = strsplit(card,'_');
    moved_suit = movedCard_split(1);
    moved_rank = movedCard_split(2);

    tp_FU_Pile_Dest = strcat('pTP_',tableau_dest,'_FaceUp_Pile');

    % Can not add to tableau piles where face up is empty and there exist
    % cards in face down pile.
    if ~isempty(tokIDs(strcat('pTP_',tableau_dest,'_FaceDown_Pile'))) && ...
            isempty(tokIDs(tp_FU_Pile_Dest)),
```

```
70          set_handle(handle_err,'String','FACE_DOWN_PILE_MUST_BE_EMPTY');
71          return;
72      end
73
74      if(iscell(tp_FU_Pile_Dest)),
75          tp_FU_Pile_Dest = tp_FU_Pile_Dest{1};
76      end;
77
78      % Do not check amount once the command has reached it's destination.
79      if length(moveCmd) >= 4 && ~isempty(source),
80          if ismember(moveCmd{3}, global_info.TP_PILES),
81              tableau_src = moveCmd{3};
82              tableau_src = tableau_src(3);
83              tp_Pile_Src = strcat('pTP_',tableau_src,'_FaceUp_Pile');
84              if(iscell(tp_Pile_Src)),
85                  tp_Pile_Src = tp_Pile_Src{1};
86              end;
87          else,
88              set_handle(handle_err,'String','INVALID_MOVE_COMMAND');
89              return;
90          end
91          amount = str2double(moveCmd{4});
92          if amount > length(tokIDs(tp_Pile_Src)) || amount < 1,
93              set_handle(handle_err,'String','INVALID_AMOUNT');
94              return;
95          end;
96      end
97      % Check against the latest (lowest) card at destination.
98      dest_topCard_Id = tokenArrivedLate(tp_FU_Pile_Dest,1);
99      moved_rank_value = global_info.CARDVALUE_MAP(moved_rank{1});
100     if dest_topCard_Id,
101         dest_topCard_Color = get_color(tp_FU_Pile_Dest,dest_topCard_Id);
102         dest_topCard_split = strsplit(dest_topCard_Color{1},'_');
103         dest_topCard_Suit = dest_topCard_split(1);
104         dest_topCard_Rank = dest_topCard_split(2);
105
106         moved_global_suit = global_info.SUITS.(moved_suit{1});
107         dest_global_suit = global_info.SUITS.(dest_topCard_Suit{1});
108
109         diffRank = moved_rank_value - global_info.CARDVALUE_MAP(
                dest_topCard_Rank{1});
110         % Added card must be 1 value lower than the current card.
111         if(diffRank ~= -1),
112             set_handle(handle_err,'String','INVALID_CARD_VALUE');
113             return;
114         end;
115         % Moved and current suit color must be different (red/black).
116         if(strcmp(moved_global_suit{2},dest_global_suit{2})),
117             set_handle(handle_err,'String','SUIT_COLOR_MUST_BE_ALTERNATING');
118             return;
119         end;
120     elseif moved_rank_value ~= 13,
121         set_handle(handle_err,'String','FIRST_CARD_MUST_BE_KING');
122         return;
123     end;
124 else,
125     set_handle(handle_err,'String','INVALID_PILE');
126     return
127 end;
128
129 if ~isempty(source),
130     global_info.last_command_source = source;
131 end;
132
133 set_handle(handle_err,'String','');
134 doCommand = true;
```

B.2 COMMON_POST.m

```
1  function [] = COMMON_POST(transition)
2
3  global global_info;
4
```

```
% Release playerAction resource to allow for another player action.
if ismember(transition.name, {'tTPe_1_Add_FaceDown', 'tTPe_2_Add_FaceDown', ...
        'tTPe_3_Add_FaceDown', 'tTPe_4_Add_FaceDown', 'tTPe_5_Add_FaceDown', ...
        'tTPe_6_Add_FaceDown', 'tTPe_7_Add_FaceDown'}),
    global_info.CARDS_DEALT = global_info.CARDS_DEALT + 1;
elseif ismember(transition.name, {'tTPe_1_Add_FaceUp', 'tTPe_2_Add_FaceUp', ...
        'tTPe_3_Add_FaceUp', 'tTPe_4_Add_FaceUp', 'tTPe_5_Add_FaceUp', ...
        'tTPe_6_Add_FaceUp', 'tTPe_7_Add_FaceUp'}),
    post_tTPe_Add_FaceUp(transition);
elseif ismember(transition.name, {
        'tFPe_Clubs_Add','tFPe_Diamonds_Add', 'tFPe_Hearts_Add', ...
        'tFPe_Spades_Add', 'tTPe_1_Turn', 'tTPe_2_Turn', 'tTPe_3_Turn', ...
        'tTPe_4_Turn', 'tTPe_5_Turn', 'tTPe_6_Turn', 'tTPe_7_Turn', ...
        'tMC_DP_Move_Siphon', 'tMC_FP_Move_Siphon', 'tMC_Out_Buffer_Siphon', ...
        'tMC_TP_Move_Siphon', 'tMC_TP_Turn_Siphon'}),
    release(global_info.last_command_source);
elseif ismember(transition.name, {
        'tPBe_DP_Move','tPBe_DP_Turn', 'tPBe_FP_Move','tPBe_TP_Move', ...
        'tPBe_TP_Turn',}),
    global_info.BOT_ACTIONS_NEW_CMD = 1;
elseif ismember(transition.name, {'tTPi_1_Move_Multiple', 'tTPi_2_Move_Multiple', ...
        'tTPi_3_Move_Multiple', 'tTPi_4_Move_Multiple', 'tTPi_5_Move_Multiple', ...
        'tTPi_6_Move_Multiple', 'tTPi_7_Move_Multiple'}),
    global_info.TP_Move_Multi_Gen_Tokens = global_info.TP_Move_Multi_Gen_Tokens - 1;
end;

% Check if game is won. Win condition: 13 tokens on each of the foundation
% piles.
if(length(tokIDs('pFP_Clubs_Pile')) == 13 && ...
        length(tokIDs('pFP_Diamonds_Pile')) == 13 && ...
        length(tokIDs('pFP_Hearts_Pile')) == 13 && ...
        length(tokIDs('pFP_Spades_Pile')) == 13),
    set_handle('GameStatus', 'String', 'GAME_WON!');
    disp('GAME_WON!');
    global_info.STOP_SIMULATION = 1;
end

if global_info.CARDS_DEALT >= global_info.INITIAL_DEAL_MOVE_LENGTH,
    if global_info.GUI_ENABLED,
        player_update_GUI();
    end
end
```

B.3 COMMON_PRE.m

```
function [fire, transition] = COMMON_PRE(transition)

if ismember(transition.name, {'tFPe_Clubs_Add', 'tFPe_Diamonds_Add', ...
        'tFPe_Hearts_Add', 'tFPe_Spades_Add'}),
    [fire, transition] = pre_tFPe_Add(transition);
elseif ismember(transition.name, {'tFPe_Clubs_Move', 'tFPe_Diamonds_Move', ...
        'tFPe_Hearts_Move', 'tFPe_Spades_Move'}),
    [fire, transition] = pre_tFPe_Move(transition);
elseif ismember(transition.name, {'tPe_FP_Clubs_Move', 'tPe_FP_Diamonds_Move', ...
        'tPe_FP_Hearts_Move', 'tPe_FP_Spades_Move'}),
    [fire, transition] = pre_tPe_FP_Move(transition);
elseif ismember(transition.name, {'tPe_FP_Clubs_Out', 'tPe_FP_Diamonds_Out', ...
        'tPe_FP_Hearts_Out', 'tPe_FP_Spades_Out'}),
    [fire, transition] = pre_tFPe_Out(transition);
elseif ismember(transition.name, {'tTPe_1_Add_FaceDown', 'tTPe_2_Add_FaceDown', ...
        'tTPe_3_Add_FaceDown', 'tTPe_4_Add_FaceDown', 'tTPe_5_Add_FaceDown', ...
        'tTPe_6_Add_FaceDown', 'tTPe_7_Add_FaceDown'}),
```

```
18      [fire, transition] = pre_tTPe_Add_FaceDown(transition);
19  elseif ismember(transition.name, {'tTPe_1_Add_FaceUp', 'tTPe_2_Add_FaceUp',
        ...
20          'tTPe_3_Add_FaceUp', 'tTPe_4_Add_FaceUp', 'tTPe_5_Add_FaceUp', ...
21          'tTPe_6_Add_FaceUp', 'tTPe_7_Add_FaceUp'}),
22      [fire, transition] = pre_tTPe_Add_FaceUp(transition);
23  elseif ismember(transition.name, {'tTPe_1_Add_FaceUp', 'tTPe_2_Add_FaceUp',
        ...
24          'tTPe_3_Add_FaceUp', 'tTPe_4_Add_FaceUp', 'tTPe_5_Add_FaceUp', ...
25          'tTPe_6_Add_FaceUp', 'tTPe_7_Add_FaceUp'}),
26      [fire, transition] = pre_tTPe_Add_FaceUp(transition);
27  elseif ismember(transition.name, {'tPe_TP_1_Turn', 'tPe_TP_2_Turn', ...
28          'tPe_TP_3_Turn', 'tPe_TP_4_Turn', 'tPe_TP_5_Turn', ...
29          'tPe_TP_6_Turn', 'tPe_TP_7_Turn'}),
30      [fire, transition] = pre_tPe_TP_Turn(transition);
31  elseif ismember(transition.name, {'tTPe_1_Turn', 'tTPe_2_Turn', ...
32          'tTPe_3_Turn', 'tTPe_4_Turn', 'tTPe_5_Turn', ...
33          'tTPe_6_Turn', 'tTPe_7_Turn'}),
34      [fire, transition] = pre_tTPe_Turn(transition);
35  elseif ismember(transition.name, {'tPe_TP_1_Move', 'tPe_TP_2_Move', ...
36          'tPe_TP_3_Move', 'tPe_TP_4_Move', 'tPe_TP_5_Move', ...
37          'tPe_TP_6_Move', 'tPe_TP_7_Move'}),
38      [fire, transition] = pre_tPe_TP_Move(transition);
39  elseif ismember(transition.name, {'tTPe_1_Move', 'tTPe_2_Move', ...
40          'tTPe_3_Move', 'tTPe_4_Move', 'tTPe_5_Move', ...
41          'tTPe_6_Move', 'tTPe_7_Move'}),
42      [fire, transition] = pre_tTPe_Move(transition);
43  elseif ismember(transition.name, {'tTPe_1_Out', 'tTPe_2_Out', ...
44          'tTPe_3_Out', 'tTPe_4_Out', 'tTPe_5_Out', ...
45          'tTPe_6_Out', 'tTPe_7_Out'}),
46      [fire, transition] = pre_tTPe_Out(transition);
47  elseif ismember(transition.name, {'tTPi_1_Move_Multiple', '
        tTPi_2_Move_Multiple', ...
48          'tTPi_3_Move_Multiple', 'tTPi_4_Move_Multiple', 'tTPi_5_Move_Multiple'
                , ...
49          'tTPi_6_Move_Multiple', 'tTPi_7_Move_Multiple'}),
50      [fire, transition] = pre_tTPi_Move_Multiple(transition);
51  else
52      fire = 1;
53  end
54
55  end
```

B.4 draw_pile_pdf.m

```
1   function [png] = draw_pile_pdf()
2   png.PN_name = 'Draw_Pile';
3
4   png.set_of_Ps = {'pDP_Draw_FaceUp_Pile','pDP_Draw_FaceDown_Pile', ...
5       'pDP_Move_Out','pDP_Dealer', 'pDP_Turn', 'pDP_Move_Init'};
6   png.set_of_Ts = {'tDPi_Dealer','tDPe_Out','tDPe_Turn','tDPi_Turn', ...
7       'tDPe_Move', 'tDPi_Enable_FP_Trans', 'tDPi_Flip_Pile', 'tDPi_Move_Init'};
8   png.set_of_As = {
9       'tDPi_Dealer','pDP_Draw_FaceDown_Pile',1, ...
10      'pDP_Draw_FaceUp_Pile','tDPe_Out',1, ...
11      'pDP_Dealer', 'tDPi_Dealer', 1, ...
12      'pDP_Draw_FaceDown_Pile', 'tDPi_Turn', 1, ...
13      'tDPi_Turn', 'pDP_Draw_FaceUp_Pile', 1, ...
14      'pDP_Move_Out', 'tDPe_Out', 1, ...
15      'tDPe_Move', 'pDP_Move_Out', 1, ...
16      'pDP_Draw_FaceUp_Pile', 'tDPi_Flip_Pile', 1, ...
17      'tDPi_Flip_Pile', 'pDP_Draw_FaceDown_Pile', 1, ...
18      'pDP_Turn', 'tDPi_Enable_FP_Trans', 1, ...
19      'pDP_Turn', 'tDPi_Turn', 1, ...
20      'tDPe_Turn', 'pDP_Turn', 1, ...
21      'pDP_Move_Init', 'tDPi_Move_Init', 1, ...
22      'tDPi_Move_Init', 'pDP_Move_Out', 1, ...
23      };
24  png.set_of_Is = {
25      'pDP_Dealer', 'tDPe_Move', 1, ...
26      'pDP_Dealer', 'tDPe_Turn', 1, ...
27      'pDP_Dealer', 'tDPi_Enable_FP_Trans', 1, ...
28      'pDP_Dealer', 'tDPi_Turn', 1, ...
```

```
    'pDP_Draw_FaceDown_Pile', 'tDPi_Enable_FP_Trans', 1
    };
```

B.5 foundation_pile_clubs_pdf.m

```
function [png] = foundation_pile_clubs_pdf()
modname = 'Clubs';
png.PN_name = strcat('Foundation_Pile_-',{'_'},modname);

png.set_of_Ps = {strcat('pFP_',modname,'_Pile'),...
    strcat('pFP_',modname,'_Move')};
png.set_of_Ts = {strcat('tFPe_',modname,'_Add'),...
    strcat('tFPe_',modname,'_Move'),strcat('tFPe_',modname,'_Out')};
png.set_of_As = {
    strcat('tFPe_',modname,'_Add'),strcat('pFP_',modname,'_Pile'),1, ...
    strcat('pFP_',modname,'_Pile'),strcat('tFPe_',modname,'_Out'),1, ...
    strcat('tFPe_',modname,'_Move'), strcat('pFP_',modname,'_Move'), 1, ...
    strcat('pFP_',modname,'_Move'), strcat('tFPe_',modname,'_Out'), 1, ...
    };
```

B.6 foundation_pile_diamonds_pdf.m

```
function [png] = foundation_pile_diamonds_pdf()
modname = 'Diamonds';
png.PN_name = strcat('Foundation_Pile_-',{'_'},modname);

png.set_of_Ps = {strcat('pFP_',modname,'_Pile'),...
    strcat('pFP_',modname,'_Move')};
png.set_of_Ts = {strcat('tFPe_',modname,'_Add'),...
    strcat('tFPe_',modname,'_Move'),strcat('tFPe_',modname,'_Out')};
png.set_of_As = {
    strcat('tFPe_',modname,'_Add'),strcat('pFP_',modname,'_Pile'),1, ...
    strcat('pFP_',modname,'_Pile'),strcat('tFPe_',modname,'_Out'),1, ...
    strcat('tFPe_',modname,'_Move'), strcat('pFP_',modname,'_Move'), 1, ...
    strcat('pFP_',modname,'_Move'), strcat('tFPe_',modname,'_Out'), 1, ...
    };
```

B.7 foundation_pile_hearts_pdf.m

```
function [png] = foundation_pile_hearts_pdf()
modname = 'Hearts';
png.PN_name = strcat('Foundation_Pile_-',{'_'},modname);

png.set_of_Ps = {strcat('pFP_',modname,'_Pile'),...
    strcat('pFP_',modname,'_Move')};
png.set_of_Ts = {strcat('tFPe_',modname,'_Add'),...
    strcat('tFPe_',modname,'_Move'),strcat('tFPe_',modname,'_Out')};
png.set_of_As = {
    strcat('tFPe_',modname,'_Add'),strcat('pFP_',modname,'_Pile'),1, ...
    strcat('pFP_',modname,'_Pile'),strcat('tFPe_',modname,'_Out'),1, ...
    strcat('tFPe_',modname,'_Move'), strcat('pFP_',modname,'_Move'), 1, ...
    strcat('pFP_',modname,'_Move'), strcat('tFPe_',modname,'_Out'), 1, ...
    };
```

B.8 foundation_pile_spades_pdf.m

```
function [png] = foundation_pile_spades_pdf()
modname = 'Spades';
png.PN_name = strcat('Foundation_Pile_-',{'_'},modname);

png.set_of_Ps = {strcat('pFP_',modname,'_Pile'),...
    strcat('pFP_',modname,'_Move')};
png.set_of_Ts = {strcat('tFPe_',modname,'_Add'),...
    strcat('tFPe_',modname,'_Move'),strcat('tFPe_',modname,'_Out')};
png.set_of_As = {
    strcat('tFPe_',modname,'_Add'),strcat('pFP_',modname,'_Pile'),1, ...
    strcat('pFP_',modname,'_Pile'),strcat('tFPe_',modname,'_Out'),1, ...
    strcat('tFPe_',modname,'_Move'), strcat('pFP_',modname,'_Move'), 1, ...
    strcat('pFP_',modname,'_Move'), strcat('tFPe_',modname,'_Out'), 1, ...
    };
```

B.9 get_handle.m

```
function [value] = get_handle(Handle, PropertyName)
    % Extend Matlab GET command to first check if GUI is enabled.
    % GET(H,'PropertyName')
    global global_info;
    if global_info.GUI_ENABLED,
        value = get(global_info.handles.(Handle),PropertyName);
    else,
        value = 0;
    end;
end
```

B.10 get_suit_from_transname.m

```
function [suit_abbr, suit, handle_err, move_btn, handle_move_loc] ...
    = get_suit_from_transname(transitionname)
    global global_info;
    handle_err = 0;
    move_btn = 0;
    handle_move_loc = 0;
    if ~isempty(strfind(transitionname,'Clubs')),
        suit = 'Clubs';
    elseif ~isempty(strfind(transitionname,'Diamonds')),
        suit = 'Diamonds';
    elseif ~isempty(strfind(transitionname,'Hearts')),
        suit = 'Hearts';
    elseif ~isempty(strfind(transitionname,'Spades')),
        suit = 'Spades';
    else,
        suit = 0; % Invalid suit.
    end

    suit_abbr = suit(1);
    if global_info.GUI_ENABLED,
        handle_err = strcat('FP_',suit_abbr,'_ErrorMsg');
        move_btn = strcat('FP_',suit_abbr,'_Move_Btn');
        handle_move_loc = strcat('FP_',suit_abbr,'_Move_Location');
    end
end
```

B.11 get_tableau_from_transname.m

```
function [tableau, handle_err, move_btn, turn_btn, handle_move_loc,
    handle_move_amount] ...
    = get_tableau_from_transname(transitionname)
    global global_info;
    handle_err = 0;
```

```
    move_btn = 0;
    turn_btn = 0;
    handle_move_loc = 0;
    handle_move_amount = 0;
    if ~isempty(strfind(transitionname,'1')),
        tableau = '1';
    elseif ~isempty(strfind(transitionname,'2')),
        tableau = '2';
    elseif ~isempty(strfind(transitionname,'3')),
        tableau = '3';
    elseif ~isempty(strfind(transitionname,'4')),
        tableau = '4';
    elseif ~isempty(strfind(transitionname,'5')),
        tableau = '5';
    elseif ~isempty(strfind(transitionname,'6')),
        tableau = '6';
    elseif ~isempty(strfind(transitionname,'7')),
        tableau = '7';
    else,
        tableau = 0; % Invalid tableau.
    end
    if global_info.GUI_ENABLED,
        handle_err = strcat('TP_',tableau,'_ErrorMsg');
        move_btn = strcat('TP_',tableau,'_Move_Btn');
        turn_btn = strcat('TP_',tableau,'_Turn_Btn');
        handle_move_loc = strcat('TP_',tableau,'_Move_Location');
        handle_move_amount = strcat('TP_',tableau,'_Move_Amount');
    end
end
```

B.12 main_simulation_file.m

```
% Solitaire main simulation file
clear all; clc;
global global_info;

%%%% SIMULATION SETTINGS %%%%
global_info.GUI_ENABLED = 1;
global_info.BOT_ENABLED = 1;
global_info.DISP_CHANGES = 1;
global_info.DELTA_TIME = 1;
global_info.MAX_LOOP = 15000;

%%%% GAME SETTINGS %%%%
% The bot generates a number from 1-100, this is number is used with the
% array below to determine which action is to be taken. The cutoffs are,
% DP_Turn, DP_Move, FP_Move, TP_Turn, TP_Move. Given array [20, 50, 70,
% 80], a number between 1-20 would attempt a DP_Turn action, 21-50 DP_Move,
% and so on.
global_info.BOT_ACTIONS = [10, 42, 44, 60];
% The probability of moving to a tableau pile versus a foundation pile.
global_info.BOT_ACTIONS_TP_FP = 15;
% The probability that the bot will attempt to move the full stack versus a
% part of it.
global_info.BOT_ACTIONS_TP_FULL_PARTIAL_MOVE = 35;

global_info.RANDOM_DECK = 0;
% First entry is bottom of the deck. Last entry is top of the deck.
global_info.DECK = {...
    'D_A','D_2','D_3','D_4','D_5','D_6', 'D_7', ...
    'D_8','D_9','D_X','D_J','D_Q','D_K', ...
    'C_A','C_2','C_3','C_4','C_5','C_6','C_7', ...
    'C_8','C_9','C_X','C_J','C_Q','C_K', ...
    'H_A','H_2','H_3','H_4','H_5','H_6','H_7', ...
    'H_8','H_9','H_X','H_J','H_Q','H_K', ...
    'S_A','S_2','S_3','S_4','S_5','S_6','S_7', ...
    'S_8','S_9','S_X','S_J','S_Q','S_K'
};
% To which Tableau pile the cards will be dealt. The first entry is to top
% of the deck (See global_info.DECK).
global_info.INITIAL_DEAL_MOVE = {
    '1', '2', '3', '4', '5', '6', '7', ...
    '2', '3', '4', '5', '6', '7', ...
```

```
    '3', '4', '5', '6', '7', ...
    '4', '5', '6', '7', ...
    '5', '6', '7', ...
    '6', '7', ...
    '7'};

global_info.SUITS.D = {'Diamonds','Red'};
global_info.SUITS.C = {'Clubs','Black'};
global_info.SUITS.H = {'Hearts','Red'};
global_info.SUITS.S = {'Spades','Black'};
global_info.CARDVALUE_MAP = containers.Map(...
    {'A','2','3','4','5','6','7','8','9','X','J','Q','K'}, ...
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] ...
    );
global_info.FP_PILES = {'FPC','FPD','FPH','FPS'};
global_info.TP_PILES = {'TP1','TP2','TP3','TP4','TP5','TP6','TP7'};
global_info.FP_TP_PILES = [global_info.FP_PILES, global_info.TP_PILES];

%%%% GLOBAL PARAMETERS %%%%
global_info.SCORE = 0;
global_info.TP_Move_Multiple = 0;
global_info.TP_Move_Multiple_Count = 0;
global_info.DP_Flip_Pile_Running = false;
global_info.CARDS_DEALT = 0;
global_info.INITIAL_DEAL_MOVE_LENGTH = length(global_info.INITIAL_DEAL_MOVE);
global_info.INITIAL_DECK_LENGTH = length(global_info.DECK);
global_info.BOT_ACTIONS_NEW_CMD = 1;
global_info.BOT_LAST_CMD = '';
global_info.BOT_NEXT_CMD = '';

%%%% COMPOSE STATIC GRAPH %%%%%%%
pn_struct = {
    'module_connector_pdf'; % Module connector ...
    'draw_pile_pdf'; % Game pile ...
    'foundation_pile_clubs_pdf'; % Foundation pile: Clubs ...
    'foundation_pile_diamonds_pdf' % Foundation pile: Diamonds ...
    'foundation_pile_hearts_pdf' % Foundation pile: Hearts ...
    'foundation_pile_spades_pdf' % Foundation pile: Spades ...
    'tableau_pile_1_pdf' % Tableau pile 1 ...
    'tableau_pile_2_pdf' % Tableau pile 2 ...
    'tableau_pile_3_pdf' % Tableau pile 3 ...
    'tableau_pile_4_pdf' % Tableau pile 4 ...
    'tableau_pile_5_pdf' % Tableau pile 5 ...
    'tableau_pile_6_pdf' % Tableau pile 6 ...
    'tableau_pile_7_pdf' % Tableau pile 7 ...
    };


if global_info.GUI_ENABLED,
    pn_struct{length(pn_struct) + 1} = 'player_pdf';
end;
if global_info.BOT_ENABLED,
    pn_struct{length(pn_struct) + 1} = 'player_bot_pdf';
end;
pns = pnstruct(pn_struct);
%%%% DYNAMIC DETAILS %%%%
% Only one resource in the PN. Used to symbolize that there is an ongoing
% action, so that a new one can not be started. This assures the atomicity
% and correctness of the system.
dyn.re = {'playerAction', 1, inf};

% Initial tokens.
dyn.m0 = {'pDP_Dealer', global_info.INITIAL_DECK_LENGTH, 'pDP_Turn', ...
    length(global_info.INITIAL_DEAL_MOVE), 'pDP_Move_Init', ...
    length(global_info.INITIAL_DEAL_MOVE)};

% Set priority on the initial move transition to be higher than all other
% transition in order to have a more natural ordering of cards.
dyn.ip = {'tDPe_Out', 10};
% Need to have some time to be able to fetch tokens based on time. (Which
% arrived earliest or latest).
dyn.ft = {'allothers', 0.01};
if global_info.BOT_ENABLED,
    %dyn.ft = [dyn.ft, 'tPBi_Gen', 1];
end;

if global_info.GUI_ENABLED,
    player_GUI;
```

```
end

%%%% SIMULATE %%%%%
pni = initialdynamics(pns, dyn);
sim = gpensim(pni);

%prnss(sim);
prnfinalcolors(sim)
```

B.13 module_connector_pdf.m

```
function [png] = module_connector_pdf()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File: module_connector_pdf.m : Handles the connections of the modules.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

png.PN_name = 'Module_connector';

png.set_of_Ps = {'pMC_Out_Buffer', 'pMC_DP_Turn', 'pMC_DP_Move', ...
    'pMC_TP_Turn', 'pMC_TP_Move', 'pMC_FP_Move'};
png.set_of_Ts = {'tMC_Out_Buffer_Siphon', 'tMC_DP_Move_Siphon', ...
    'tMC_TP_Turn_Siphon', 'tMC_TP_Move_Siphon', 'tMC_FP_Move_Siphon'};
png.set_of_As = {
    'tDPe_Out', 'pMC_Out_Buffer', 1, ...
    'pMC_DP_Turn', 'tDPe_Turn', 1, ...
    'pMC_DP_Move', 'tDPe_Move', 1, ...
    'pMC_Out_Buffer', 'tMC_Out_Buffer_Siphon', 1, ...
    'pMC_DP_Move', 'tMC_DP_Move_Siphon', 1, ...
    'pMC_TP_Turn', 'tMC_TP_Turn_Siphon', 1, ...
    'pMC_TP_Move', 'tMC_TP_Move_Siphon', 1, ...
    'pMC_FP_Move', 'tMC_FP_Move_Siphon', 1, ...
    };
% Add connections to all 7 tableau piles %
for i = 1:7
    num = num2str(i);
    png.set_of_As = [png.set_of_As, {strcat('tTPe_',num,'_Out'),'pMC_Out_Buffer', 1}];
    png.set_of_As = [png.set_of_As, {'pMC_Out_Buffer',strcat('tTPe_',num,'_Add_FaceDown'), 1}];
    png.set_of_As = [png.set_of_As, {'pMC_Out_Buffer',strcat('tTPe_',num,'_Add_FaceUp'), 1}]; % Moving cards from one TP to another
    png.set_of_As = [png.set_of_As, {'pMC_TP_Move',strcat('tTPe_',num,'_Move'), 1}];
    png.set_of_As = [png.set_of_As, {'pMC_TP_Turn',strcat('tTPe_',num,'_Turn'), 1}];
end;

% Add connections to all 4 foundation piles %
foundationpiles = {'Spades','Hearts','Diamonds','Clubs'};
for i = 1:4
    fp = foundationpiles(i);
    png.set_of_As = [png.set_of_As, {strcat('tFPe_',fp{1},'_Out'),'pMC_Out_Buffer', 1}];
    png.set_of_As = [png.set_of_As, {'pMC_Out_Buffer',strcat('tFPe_',fp{1},'_Add'), 1}];
    png.set_of_As = [png.set_of_As, {'pMC_FP_Move',strcat('tFPe_',fp{1},'_Move'), 1}];
end;
```

B.14 player_bot_pdf.m

```
function [png] = player_bot_pdf()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File: player_bot_pdf.m : Bot for simulating user actions.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

png.PN_name = 'Player_Bot_module';

png.set_of_Ps = {'pPB_Cmd'};
```

```
png.set_of_Ts = {'tPBi_Gen', 'tPBe_DP_Turn', 'tPBe_DP_Move', 'tPBe_TP_Turn',
    ...
    'tPBe_TP_Move', 'tPBe_FP_Move', 'tPBi_Siphon'};
png.set_of_As = {
    'tPBi_Gen', 'pPB_Cmd', 1, ...
    'pPB_Cmd', 'tPBi_Siphon', 1, ...
    'pPB_Cmd', 'tPBe_DP_Turn', 1, ...
    'pPB_Cmd', 'tPBe_DP_Move', 1, ...
    'pPB_Cmd', 'tPBe_FP_Move', 1, ...
    'pPB_Cmd', 'tPBe_TP_Turn', 1, ...
    'pPB_Cmd', 'tPBe_TP_Move', 1, ...
    'tPBe_DP_Turn', 'pMC_DP_Turn', 1, ...
    'tPBe_DP_Move', 'pMC_DP_Move', 1, ...
    'tPBe_FP_Move', 'pMC_FP_Move', 1, ...
    'tPBe_TP_Turn', 'pMC_TP_Turn', 1, ...
    'tPBe_TP_Move', 'pMC_TP_Move', 1, ...
    };

% Add connections to all 7 tableau piles %
% for i = 1:7
%     num = num2str(i);
%     png.set_of_As = [png.set_of_As, {strcat('tPe_TP_',num,'_Turn'),'
    pMC_TP_Turn', 1}];
%     png.set_of_As = [png.set_of_As, {strcat('tPe_TP_',num,'_Move'),'
    pMC_TP_Move', 1}];
% end;
%
% % Add connections to all 4 foundation piles %
% foundationpiles = {'Clubs','Diamonds','Hearts','Spades'};
% for i = 1:4
%     fp = foundationpiles(i);
%     png.set_of_As = [png.set_of_As, {strcat('tPe_FP_',fp{1},'_Move'),'
    pMC_FP_Move', 1}];
% end;
```

B.15 player_GUI.m

```
function varargout = player_GUI(varargin)

% PLAYER_GUI MATLAB code for player_GUI.fig
%      PLAYER_GUI, by itself, creates a new PLAYER_GUI or raises the existing
%      singleton*.
%
%      H = PLAYER_GUI returns the handle to a new PLAYER_GUI or the handle to
%      the existing singleton*.
%
%      PLAYER_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PLAYER_GUI.M with the given input arguments.
%
%      PLAYER_GUI('Property','Value',...) creates a new PLAYER_GUI or raises
    the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before player_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to player_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help player_GUI

% Last Modified by GUIDE v2.5 07-Nov-2017 18:47:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @player_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @player_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
```

```
36  if nargin && ischar(varargin{1})
37      gui_State.gui_Callback = str2func(varargin{1});
38  end
39
40  if nargout
41      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
42  else
43      gui_mainfcn(gui_State, varargin{:});
44  end
45  % End initialization code - DO NOT EDIT
46
47
48  % --- Executes just before player_GUI is made visible.
49  function player_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
50  % This function has no output args, see OutputFcn.
51  % hObject    handle to figure
52  % eventdata  reserved - to be defined in a future version of MATLAB
53  % handles    structure with handles and user data (see GUIDATA)
54  % varargin   command line arguments to player_GUI (see VARARGIN)
55
56  % Choose default command line output for player_GUI
57  handles.output = hObject;
58  global global_info;
59  global_info.handles = handles;
60
61  % Define default states for all click buttons
62  global_info.DP_Turn_Btn = false;
63  global_info.DP_Move_Btn = false;
64
65  global_info.TP_1_Turn_Btn = false;
66  global_info.TP_1_Move_Btn = false;
67  global_info.TP_2_Turn_Btn = false;
68  global_info.TP_2_Move_Btn = false;
69  global_info.TP_3_Turn_Btn = false;
70  global_info.TP_3_Move_Btn = false;
71  global_info.TP_4_Turn_Btn = false;
72  global_info.TP_4_Move_Btn = false;
73  global_info.TP_5_Turn_Btn = false;
74  global_info.TP_5_Move_Btn = false;
75  global_info.TP_6_Turn_Btn = false;
76  global_info.TP_6_Move_Btn = false;
77  global_info.TP_7_Turn_Btn = false;
78  global_info.TP_7_Move_Btn = false;
79
80  global_info.FP_C_Move_Btn = false;
81  global_info.FP_D_Move_Btn = false;
82  global_info.FP_H_Move_Btn = false;
83  global_info.FP_S_Move_Btn = false;
84  global_info.Initial_Deal_Btn = false;
85
86  % Update handles structure
87  guidata(hObject, handles);
88
89  % UIWAIT makes player_GUI wait for user response (see UIRESUME)
90  % uiwait(handles.figure1);
91
92  % --- Outputs from this function are returned to the command line.
93  function varargout = player_GUI_OutputFcn(hObject, eventdata, handles)
94  varargout{1} = handles.output;
95
96  % --- Executes on button press in STOPSIM.
97  function STOPSIM_Callback(hObject, eventdata, handles)
98  global global_info;
99  global_info.STOP_SIMULATION = 1;
100
101 % --- Executes on button press in TOGGLEBOT.
102 function TOGGLEBOT_Callback(hObject, eventdata, handles)
103 global global_info;
104 global_info.BOT_ENABLED = ~global_info.BOT_ENABLED;
105
106 % --- Executes on button press in DP_Turn_Btn.
107 function DP_Turn_Btn_Callback(hObject, eventdata, handles)
108 global global_info;
109 if global_info.DP_Turn_Btn == false,
110     global_info.DP_Turn_Btn = true;
111 end
112
113 % --- Executes on button press in DP_Move_Btn.
```

```
114 function DP_Move_Btn_Callback(hObject, eventdata, handles)
115 global global_info;
116 if global_info.DP_Move_Btn == false,
117     global_info.DP_Move_Btn = true;
118 end
119
120 % --- Executes on button press in FP_C_Move_Btn.
121 function FP_C_Move_Btn_Callback(hObject, eventdata, handles)
122 global global_info;
123 if global_info.FP_C_Move_Btn == false,
124     global_info.FP_C_Move_Btn = true;
125 end
126
127 % --- Executes on button press in FP_D_Move_Btn.
128 function FP_D_Move_Btn_Callback(hObject, eventdata, handles)
129 global global_info;
130 if global_info.FP_D_Move_Btn == false,
131     global_info.FP_D_Move_Btn = true;
132 end
133
134 % --- Executes on button press in FP_H_Move_Btn.
135 function FP_H_Move_Btn_Callback(hObject, eventdata, handles)
136 global global_info;
137 if global_info.FP_H_Move_Btn == false,
138     global_info.FP_H_Move_Btn = true;
139 end
140
141 % --- Executes on button press in FP_S_Move_Btn.
142 function FP_S_Move_Btn_Callback(hObject, eventdata, handles)
143 global global_info;
144 if global_info.FP_S_Move_Btn == false,
145     global_info.FP_S_Move_Btn = true;
146 end
147
148 % --- Executes on button press in TP_1_Turn_Btn.
149 function TP_1_Turn_Btn_Callback(hObject, eventdata, handles)
150 global global_info;
151 if global_info.TP_1_Turn_Btn == false,
152     global_info.TP_1_Turn_Btn = true;
153 end
154
155 % --- Executes on button press in TP_1_Move_Btn.
156 function TP_1_Move_Btn_Callback(hObject, eventdata, handles)
157 global global_info;
158 if global_info.TP_1_Move_Btn == false,
159     global_info.TP_1_Move_Btn = true;
160 end
161
162 % --- Executes on button press in TP_2_Turn_Btn.
163 function TP_2_Turn_Btn_Callback(hObject, eventdata, handles)
164 global global_info;
165 if global_info.TP_2_Turn_Btn == false,
166     global_info.TP_2_Turn_Btn = true;
167 end
168
169 % --- Executes on button press in TP_2_Move_Btn.
170 function TP_2_Move_Btn_Callback(hObject, eventdata, handles)
171 global global_info;
172 if global_info.TP_2_Move_Btn == false,
173     global_info.TP_2_Move_Btn = true;
174 end
175
176 % --- Executes on button press in TP_3_Turn_Btn.
177 function TP_3_Turn_Btn_Callback(hObject, eventdata, handles)
178 global global_info;
179 if global_info.TP_3_Turn_Btn == false,
180     global_info.TP_3_Turn_Btn = true;
181 end
182
183 % --- Executes on button press in TP_3_Move_Btn.
184 function TP_3_Move_Btn_Callback(hObject, eventdata, handles)
185 global global_info;
186 if global_info.TP_3_Move_Btn == false,
187     global_info.TP_3_Move_Btn = true;
188 end
189
190 % --- Executes on button press in TP_4_Turn_Btn.
191 function TP_4_Turn_Btn_Callback(hObject, eventdata, handles)
```

```
global global_info;
if global_info.TP_4_Turn_Btn == false,
    global_info.TP_4_Turn_Btn = true;
end

% --- Executes on button press in TP_4_Move_Btn.
function TP_4_Move_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_4_Move_Btn == false,
    global_info.TP_4_Move_Btn = true;
end

% --- Executes on button press in TP_5_Turn_Btn.
function TP_5_Turn_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_5_Turn_Btn == false,
    global_info.TP_5_Turn_Btn = true;
end

% --- Executes on button press in TP_5_Move_Btn.
function TP_5_Move_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_5_Move_Btn == false,
    global_info.TP_5_Move_Btn = true;
end

% --- Executes on button press in TP_6_Turn_Btn.
function TP_6_Turn_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_6_Turn_Btn == false,
    global_info.TP_6_Turn_Btn = true;
end

% --- Executes on button press in TP_6_Move_Btn.
function TP_6_Move_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_6_Move_Btn == false,
    global_info.TP_6_Move_Btn = true;
end

% --- Executes on button press in TP_7_Turn_Btn.
function TP_7_Turn_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_7_Turn_Btn == false,
    global_info.TP_7_Turn_Btn = true;
end

% --- Executes on button press in TP_7_Move_Btn.
function TP_7_Move_Btn_Callback(hObject, eventdata, handles)
global global_info;
if global_info.TP_7_Move_Btn == false,
    global_info.TP_7_Move_Btn = true;
end
```

B.16 player_pdf.m

```
function [png] = player_pdf()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File: player_pdf.m : Handles inputs from the player.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

png.PN_name = 'Player module';

png.set_of_Ps = {};
png.set_of_Ts = {'tPe_DP_Turn', 'tPe_DP_Move', 'tPe_TP_1_Turn', 'tPe_TP_1_Move',...
    'tPe_TP_2_Turn', 'tPe_TP_2_Move', 'tPe_TP_3_Turn', 'tPe_TP_3_Move', ...
    'tPe_TP_4_Turn', 'tPe_TP_4_Move', 'tPe_TP_5_Turn', 'tPe_TP_5_Move', ...
    'tPe_TP_6_Turn', 'tPe_TP_6_Move', 'tPe_TP_7_Turn', 'tPe_TP_7_Move', ...
    'tPe_FP_Clubs_Move', 'tPe_FP_Diamonds_Move', 'tPe_FP_Hearts_Move', 'tPe_FP_Spades_Move'};
png.set_of_As = {
    'tPe_DP_Turn', 'pMC_DP_Turn', 1, ... % Player module
```

```
16      'tPe_DP_Move', 'pMC_DP_Move', 1, ... % Player module
17      };
18
19  % Add connections to all 7 tableau piles %
20  for i = 1:7
21      num = num2str(i);
22      png.set_of_As = [png.set_of_As, {strcat('tPe_TP_',num,'_Turn'),'
            pMC_TP_Turn', 1}];
23      png.set_of_As = [png.set_of_As, {strcat('tPe_TP_',num,'_Move'),'
            pMC_TP_Move', 1}];
24  end;
25
26  % Add connections to all 4 foundation piles %
27  foundationpiles = {'Clubs','Diamonds','Hearts','Spades'};
28  for i = 1:4
29      fp = foundationpiles(i);
30      png.set_of_As = [png.set_of_As, {strcat('tPe_FP_',fp{1},'_Move'),'
            pMC_FP_Move', 1}];
31  end;
```

B.17 player_update_GUI.m

```
1   function [] = player_update_GUI()
2   global global_info;
3
4   %% Clear initial game status.
5   if global_info.CARDS_DEALT == global_info.INITIAL_DEAL_MOVE_LENGTH,
6       set_handle('GameStatus', 'String', '');
7   end;
8
9   %% Update Score
10  set_handle('Score', 'String', strcat('Score:',{'␣'},num2str(global_info.SCORE)
        ));
11
12  %% Draw Pile
13  vistoken = tokenArrivedLate('pDP_Draw_FaceUp_Pile',1);
14  topcard = '';
15  if vistoken,
16      topcard = get_color('pDP_Draw_FaceUp_Pile',vistoken);
17  end;
18  set(global_info.handles.DP_StatusMsg,'String',strcat('#FD:␣',num2str(length(
        tokIDs('pDP_Draw_FaceDown_Pile'))), ...
19      ',␣#FU:␣',num2str(length(tokIDs('pDP_Draw_FaceUp_Pile'))),',␣FU_Top:',{'␣'
            },topcard));
20
21
22  %% Foundation Piles
23  foundationpiles = {'Clubs','Diamonds','Hearts','Spades'};
24  for i = 1:4
25      fp = foundationpiles(i);
26      pile = fp{1};
27      vistoken = tokenArrivedLate(strcat('pFP_',pile,'_Pile'),1);
28      topcard = '';
29      if vistoken,
30          topcard = get_color(strcat('pFP_',pile,'_Pile'),vistoken);
31      end;
32      statusHandle = strcat('FP_',pile(1),'_StatusMsg');
33      set(global_info.handles.(statusHandle),'String',strcat('#:␣',num2str(
            length(tokIDs(strcat('pFP_',pile,'_Pile')))),',␣Top:',{'␣'},topcard));
34  end;
35
36  %% Tableau Piles
37  for i = 1:7
38      num = num2str(i);
39      numtokens = length(tokIDs(strcat('pTP_',num,'_FaceUp_Pile')));
40      vistoken = tokenArrivedLate(strcat('pTP_',num,'_FaceUp_Pile'),numtokens);
41      statusmsg = 'Face␣Up:';
42      if vistoken,
43          for i = numtokens:-1:1,
44              tokencolors = get_color(strcat('pTP_',num,'_FaceUp_Pile'),vistoken
                    (i));
45              statusmsg = sprintf('%s\n%s',statusmsg,tokencolors{1});
46          end
```

```
47      end;
48      faceup_handle = strcat('TP_',num,'_FaceUpMsg');
49      facedown_handle = strcat('TP_',num,'_FaceDownMsg');
50      set(global_info.handles.(faceup_handle),'String',statusmsg);
51      set(global_info.handles.(facedown_handle),'String',strcat('#FD:␣',num2str(
            length(tokIDs(strcat('pTP_',num,'_FaceDown_Pile'))))));
52  end;
```

B.18 post_tTPe_Add_FaceUp.m

```
1   function [] = post_tTPe_Add_FaceUp(transition)
2
3   global global_info;
4   global_info.CARDS_DEALT = global_info.CARDS_DEALT + 1;
5
6   if global_info.TP_Move_Multiple_Count <= 1,
7       if isfield(global_info,'last_command_source'),
8           release(global_info.last_command_source);
9       end
10  else,
11      global_info.TP_Move_Multiple_Count = global_info.TP_Move_Multiple_Count -
            1;
12  end;
```

B.19 pre_tFPe_Add.m

```
1   function [fire, transition] = pre_tFPe_Add(transition)
2
3   global global_info;
4   fire = 0;
5   moveToken = tokenArrivedEarly('pMC_Out_Buffer',1);
6   tokenColor = get_color('pMC_Out_Buffer',moveToken);
7   if(length(tokenColor) ~= 2),
8       return;
9   end;
10  [~, suit, handle_err] = get_suit_from_transname(transition.name);
11  [doCommand, cmdDest, card, cmdSource] = ...
12      checkCommand_Move(tokenColor,suit,'',handle_err);
13  if(doCommand),
14      transition.selected_tokens = moveToken;
15      transition.new_color = card;
16      transition.override = 1;
17      fire = 1;
18      global_info.SCORE = global_info.SCORE + 10;
19      if(global_info.DISP_CHANGES),
20          disp(strcat('Moved␣card',{'␣'},card,{'␣'},'from',{'␣'},cmdSource,...
21              {'␣'},'to',{'␣'},cmdDest));
22      end;
23  end
```

B.20 pre_tFPe_Move.m

```
1   function [fire, transition] = pre_tFPe_Move(transition)
2
3   fire = 0;
4   moveToken = tokenArrivedLate('pMC_FP_Move',1);
5   [suit_abbr, suit, ~] = get_suit_from_transname(transition.name);
6   [moveCmd, ~] = splitCommand(get_color('pMC_FP_Move',moveToken));
7   if(length(moveCmd) >= 3 && strcmp(moveCmd{3},strcat('FP',suit_abbr))),
8       transition.selected_tokens = moveToken;
9       fire = 1;
10  end
```

B.21 pre_tFPe_Out.m

```
function [fire, transition] = pre_tFPe_Out(transition)

% Explicitly sure to get the card at the top of the stack.
[~, suit, ~] = get_suit_from_transname(transition.name);

moveToken = tokenArrivedEarly(strcat('pFP_',suit,'_Move'), 1);
cardToken = tokenArrivedLate(strcat('pFP_',suit,'_Pile'), 1);

transition.selected_tokens = [moveToken cardToken];
fire = 1;
```

B.22 pre_tPe_FP_Move.m

```
function [fire, transition] = pre_tPe_FP_Move(transition)

global global_info;
fire = 0;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;

[suit_abbr, suit, handle_err, move_btn, handle_move_loc] =
     get_suit_from_transname(transition.name);
[playerAction] = request(transition.name, {'playerAction', 1});
if global_info.(move_btn) ~= false && playerAction,
    %global_info = setfield(global_info,move_btn,false);
    global_info.(move_btn) = false;
    dest = get_handle(handle_move_loc,'String');
    command = strcat('Move:',dest,':',strcat('FP',suit_abbr));
    vistoken = tokenArrivedLate(strcat('pFP_',suit,'_Pile'),1);
    if vistoken,
        color = get_color(strcat('pFP_',suit,'_Pile'),vistoken);
        color = color{1};
        if checkCommand_Move({command;color},'',transition.name,handle_err),
            transition.new_color = command;
            fire = 1;
        end;
    end;
end
```

B.23 pre_tPe_TP_Move.m

```
function [fire, transition] = pre_tPe_TP_Move(transition)

global global_info;
fire = 0;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;

[tableau, handle_err, move_btn, turn_btn, handle_move_loc, handle_move_amount
    ]...
    = get_tableau_from_transname(transition.name);
[playerAction] = request(transition.name, {'playerAction', 1});
if global_info.(move_btn) ~= false && playerAction,
    global_info.(move_btn) = false;
    dest = get_handle(handle_move_loc,'String');

    % Is amount numeric and equal or less that current cards in FaceUp?
    if ismember(dest, global_info.FP_PILES),
        amount = 1;
    else,
        amount = str2double(get_handle(handle_move_amount,'String'));
        if isnan(amount) || amount < 1,
            amount = 1;
```

```
23          end;
24          if amount > length(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile'))),
25              set_handle(handle_err,'String','INVALID_AMOUNT');
26              return;
27          end;
28      end;
29
30      command = strcat('Move:',dest,':TP',tableau,':',num2str(amount));
31
32      vistoken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'),amount);
33      vistoken = vistoken(amount);
34      if vistoken,
35          color = get_color(strcat('pTP_',tableau,'_FaceUp_Pile'),vistoken);
36          color = color{1};
37          if checkCommand_Move({command;color},'',transition.name,handle_err),
38              % Need some sort of perpetual fireing.
39              transition.new_color = command;
40              fire = 1;
41          end;
42      end;
43  end
```

B.24 pre_tPe_TP_Turn.m

```
1   function [fire, transition] = pre_tPe_TP_Turn(transition)
2
3   global global_info;
4   fire = 0;
5   if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
6       return;
7   end;
8
9   [tableau, handle_err, move_btn, turn_btn, handle_move_loc, handle_move_amount
        ]...
10      = get_tableau_from_transname(transition.name);
11  [playerAction] = request(transition.name, {'playerAction', 1});
12  if global_info.(turn_btn) ~= false && playerAction,
13      global_info.(turn_btn) = false;
14      if ~isempty(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile'))),
15          set_handle(handle_err,'String','FaceUp_Pile_must_be_empty');
16          return;
17      elseif isempty(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))),
18          set_handle(handle_err,'String','FaceDown_Pile_is_empty');
19          return;
20      end;
21      set_handle(handle_err,'String','');
22      global_info.last_command_source = transition.name;
23      transition.new_color = strcat('Turn:TP',tableau);
24      fire = 1;
25  end;
```

B.25 pre_tTPe_Add_FaceDown.m

```
1   function [fire, transition] = pre_tTPe_Add_FaceDown(transition)
2
3   global global_info;
4   fire = 0;
5   [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
6   % Can only add FaceDown cards during the initial dealing.
7   if global_info.CARDS_DEALT >= global_info.INITIAL_DEAL_MOVE_LENGTH ...
8       || length(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))) + 1 ...
9       == str2double(tableau),
10      return;
11  end;
12  moveToken = tokenArrivedEarly('pMC_Out_Buffer',1);
13  [moveCmd, card] = splitCommand(get_color('pMC_Out_Buffer',moveToken));
14  if length(moveCmd) >= 2 && strcmp(moveCmd{2},strcat('TP',tableau)),
15      transition.selected_tokens = moveToken;
16      transition.new_color = card;
```

```
    transition.override = 1;
    fire = 1;
    if(global_info.DISP_CHANGES),
        disp(strcat('Moved_card',{'_'},card,{'_'},'from_DP',{'_'},...
            'to',{'_'},moveCmd{2},{'_'},'(FD)'));
    end;
end
```

B.26 pre_tTPe_Add_FaceUp.m

```
function [fire, transition] = pre_tTPe_Add_FaceUp(transition)

global global_info;
[tableau, handle_err, ~, ~, ~, ~] = get_tableau_from_transname(transition.name
    );
fire = 0;

% Can only add FaceUp cards once the initial dealing is complete.
isFDFull = length(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))) + 1 ...
    == str2double(tableau);
isDealingInProgress = global_info.CARDS_DEALT < ...
    global_info.INITIAL_DEAL_MOVE_LENGTH;
if isDealingInProgress && ~isFDFull,
    return;
end;

moveToken = tokenArrivedEarly('pMC_Out_Buffer',1);
tokenColor = get_color('pMC_Out_Buffer',moveToken);
if(length(tokenColor) ~= 2),
    return;
end;

if isDealingInProgress && isFDFull,
    doCommand = true;
    [moveCmd, card] = splitCommand(tokenColor);
    cmdDest = moveCmd{2};
    source = 'DP';
else,
    [doCommand, cmdDest, card, cmdSource] = ...
        checkCommand_Move(tokenColor,tableau,'',handle_err);
    source = cmdSource;
end

if(doCommand && strcmp(cmdDest,strcat('TP',tableau))),
    transition.selected_tokens = moveToken;
    transition.new_color = card;
    transition.override = 1;
    fire = 1;

    if ~isDealingInProgress,
        if strcmp(source,'DP'),
            % 10 Points when moving from Draw Pile to Tableau
            global_info.SCORE = global_info.SCORE + 5;
        elseif ismember(source,global_info.FP_PILES),
            % Lose 15 points when moving from a Foundation Pile to Tableau
            global_info.SCORE = max(global_info.SCORE - 15, 0);
        end;
    end
    if(global_info.DISP_CHANGES),
        disp(strcat('Moved_card',{'_'},card,{'_'},'from',{'_'},source,...
            {'_'},'to',{'_'},cmdDest,{'_'},'(FU)'));
    end;
end
```

B.27 pre_tTPe_Move.m

```
function [fire, transition] = pre_tTPe_Move(transition)

global global_info;
```

```
fire = 0;
moveToken = tokenArrivedLate('pMC_TP_Move',1);
[tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);

moveColor = get_color('pMC_TP_Move',moveToken);
[moveCmd, ~] = splitCommand(moveColor);

if(length(moveCmd) >= 4 && strcmp(moveCmd{3},strcat('TP',tableau))),
    amount = str2double(moveCmd{4});
    global_info.TP_Move_Multiple_Count = amount;
    global_info.TP_Move_Multi_Gen_Tokens = amount - 1;
    global_info.TP_Move_LastCmd = moveColor;

    transition.selected_tokens = moveToken;
    fire = 1;
end
```

B.28 pre_tTPe_Out.m

```
function [fire, transition] = pre_tTPe_Out(transition)

global global_info;
fire = 0;

if global_info.TP_Move_Multi_Gen_Tokens == 0,
    [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);

    moveToken = tokenArrivedEarly(strcat('pTP_',tableau,'_Move'),1);
    lenMoveTokens = length(tokIDs(strcat('pTP_',tableau,'_Move')));
    cardToken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'), ...
        lenMoveTokens);
    cardToken = cardToken(lenMoveTokens);

    transition.selected_tokens = [moveToken cardToken];
    fire = 1;
end;
```

B.29 pre_tTPe_Turn.m

```
function [fire, transition] = pre_tTPe_Turn(transition)

global global_info;
fire = 0;
moveToken = tokenArrivedLate('pMC_TP_Turn',1);
[tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
moveCmd = get_color('pMC_TP_Turn',moveToken);

if(length(moveCmd) >= 1 && strcmp(moveCmd{1},strcat('Turn:TP',tableau))),
    if ~isempty(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile'))) || ...
            isempty(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))),
        return;
    end;

    topCard = tokenArrivedLate(strcat('pTP_',tableau,'_FaceDown_Pile'),1);
    transition.selected_tokens = topCard;
    color = get_color(strcat('pTP_',tableau,'_FaceDown_Pile'), topCard);
    transition.new_color = color{1};
    transition.override = 1;
    fire = 1;
    global_info.SCORE = global_info.SCORE + 5;
    if(global_info.DISP_CHANGES),
        disp(strcat('Turned_card',{'_'},color{1},{'_'},'at',{'_TP'},tableau));
    end;
end
```

B.30 pre_tTPi_Move_Multiple.m

```
function [fire, transition] = pre_tTPi_Move_Multiple(transition)

global global_info;
fire = 0;

if global_info.TP_Move_Multi_Gen_Tokens > 0,
    transition.new_color = global_info.TP_Move_LastCmd;
    transition.override = 1;
    fire = 1;
end;
```

B.31 set_handle.m

```
function [] = set_handle(Handle, PropertyName, PropertyValue)
    % Extend Matlab SET command to first check if GUI is enabled.
    % SET(H,'PropertyName',PropertyValue)
    global global_info;
    if global_info.GUI_ENABLED,
        set(global_info.handles.(Handle),PropertyName,PropertyValue);
    end;
end
```

B.32 splitCommand.m

```
function [command, card] = splitCommand( tokenColors )

color_1 = tokenColors{1};
if length(tokenColors) == 2,
    color_2 = tokenColors{2};
else,
    color_2 = '0';
end;
if ~isempty(strfind(color_1,'Move:')),
    command = strsplit(color_1,':');
    card = color_2;
else,
    command = strsplit(color_2,':');
    card = color_1;
end;
```

B.33 tableau_pile_1_pdf.m

```
function [png] = tableau_pile_1_pdf()
modname = '1';
png.PN_name = strcat('Tableau_Pile_',{'_'},modname);

png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
    _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
    '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
    strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
        tTPi_',modname,'_Move_Multiple')};
png.set_of_As = {
    strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
        ),1, ...
    strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
        _FaceDown_Pile'),1, ...
    strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
        ...
    strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
        1, ...
```

```
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.34 tableau_pile_2_pdf.m

```
1   function [png] = tableau_pile_2_pdf()
2   modname = '2';
3   png.PN_name = strcat('Tableau Pile -',{' '},modname);
4
5   png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
        _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
6   png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
        '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
7       strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
            tTPi_',modname,'_Move_Multiple')};
8   png.set_of_As = {
9       strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
            ),1, ...
10      strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
            _FaceDown_Pile'),1, ...
11      strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
            ...
12      strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
            1, ...
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.35 tableau_pile_3_pdf.m

```
1   function [png] = tableau_pile_3_pdf()
2   modname = '3';
3   png.PN_name = strcat('Tableau Pile -',{' '},modname);
4
5   png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
        _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
6   png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
        '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
7       strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
            tTPi_',modname,'_Move_Multiple')};
8   png.set_of_As = {
9       strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
            ),1, ...
10      strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
            _FaceDown_Pile'),1, ...
11      strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
            ...
12      strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
            1, ...
```

```
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.36 tableau_pile_4_pdf.m

```
1   function [png] = tableau_pile_4_pdf()
2   modname = '4';
3   png.PN_name = strcat('Tableau_Pile_-',{'_'},modname);
4
5   png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
        _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
6   png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
        '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
7       strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
            tTPi_',modname,'_Move_Multiple')};
8   png.set_of_As = {
9       strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
            ),1, ...
10      strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
            _FaceDown_Pile'),1, ...
11      strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
            ...
12      strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
            1, ...
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.37 tableau_pile_5_pdf.m

```
1   function [png] = tableau_pile_5_pdf()
2   modname = '5';
3   png.PN_name = strcat('Tableau_Pile_-',{'_'},modname);
4
5   png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
        _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
6   png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
        '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
7       strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
            tTPi_',modname,'_Move_Multiple')};
8   png.set_of_As = {
9       strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
            ),1, ...
10      strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
            _FaceDown_Pile'),1, ...
11      strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
            ...
12      strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
            1, ...
```

```
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.38 tableau_pile_6_pdf.m

```
1   function [png] = tableau_pile_6_pdf()
2   modname = '6';
3   png.PN_name = strcat('Tableau_Pile_-',{'_'},modname);
4
5   png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
        _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
6   png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
        '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
7       strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
            tTPi_',modname,'_Move_Multiple')};
8   png.set_of_As = {
9       strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
            ),1, ...
10      strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
            _FaceDown_Pile'),1, ...
11      strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
            ...
12      strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
            1, ...
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.39 tableau_pile_7_pdf.m

```
1   function [png] = tableau_pile_7_pdf()
2   modname = '7';
3   png.PN_name = strcat('Tableau_Pile_-',{'_'},modname);
4
5   png.set_of_Ps = {strcat('pTP_',modname,'_FaceUp_Pile'),strcat('pTP_',modname,'
        _FaceDown_Pile'),strcat('pTP_',modname,'_Move')};
6   png.set_of_Ts = {strcat('tTPe_',modname,'_Add_FaceUp'),strcat('tTPe_',modname,
        '_Add_FaceDown'),strcat('tTPe_',modname,'_Move'), ...
7       strcat('tTPe_',modname,'_Turn'),strcat('tTPe_',modname,'_Out'),strcat('
            tTPi_',modname,'_Move_Multiple')};
8   png.set_of_As = {
9       strcat('tTPe_',modname,'_Add_FaceUp'),strcat('pTP_',modname,'_FaceUp_Pile'
            ),1, ...
10      strcat('tTPe_',modname,'_Add_FaceDown'),strcat('pTP_',modname,'
            _FaceDown_Pile'),1, ...
11      strcat('tTPe_',modname,'_Turn'),strcat('pTP_',modname,'_FaceUp_Pile'),1,
            ...
12      strcat('pTP_',modname,'_FaceDown_Pile'), strcat('tTPe_',modname,'_Turn'),
            1, ...
```

```
13      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
            ...
14      strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
15      strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
16      strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
            2, ...
17      strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
            1
18      };
19  png.set_of_Is = {
20      strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
21      };
```

B.40 tDPe_Move_pre.m

```
1   function [fire, transition] = tDPe_Move_pre(transition)
2
3   fire = 0;
4   if ~isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
5       fire = 1;
6   end
```

B.41 tDPe_Out_pre.m

```
1   function [fire, transition] = tDPe_Out_pre(transition)
2
3   % Want to make sure that we get the earliest move-token, and the latest
4   % card. This is so that we can have a natural ordering of the cards during
5   % the initial dealing.
6   moveToken = tokenArrivedEarly('pDP_Move_Out', 1);
7   % Explicitly sure to get the card at the top of the stack.
8   cardToken = tokenArrivedLate('pDP_Draw_FaceUp_Pile', 1);
9
10  transition.selected_tokens = [moveToken cardToken];
11  fire = 1;
```

B.42 tDPi_Dealer_pre.m

```
1   function [fire, transition] = tDPi_Dealer_pre(transition)
2
3   global global_info;
4   fire = 0;
5   if ~isempty(global_info.DECK),
6       if global_info.RANDOM_DECK,
7           card = randi([1,length(global_info.DECK)]);
8       else,
9           card = 1;
10      end;
11      transition.new_color = global_info.DECK(card);
12      global_info.DECK(card) = []; % Remove this card from the array
13      fire = 1;
14  end;
```

B.43 tDPi_Enable_FP_Trans_post.m

```
1   function [] = tDPi_Enable_FP_Trans_post(transition)
2
3   global global_info;
4   if ~isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
5       global_info.DP_Flip_Pile_Running = true;
6   else,
7       % Release playerAction resource to allow for another player action.
8       release(global_info.last_command_source);
9   end;
```

B.44 tDPi_Flip_Pile_post.m

```
function [] = tDPi_Flip_Pile_post(transition)

global global_info;
if isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
    global_info.DP_Flip_Pile_Running = false;
    global_info.SCORE = max(global_info.SCORE - 100, 0);
    % Release playerAction resource to allow for another player action.
    release(global_info.last_command_source);
end;
```

B.45 tDPi_Flip_Pile_pre.m

```
function [fire, transition] = tDPi_Flip_Pile_pre(transition)

global global_info;
fire = 0;
if global_info.DP_Flip_Pile_Running == true,
    transition.selected_tokens = tokenArrivedLate('pDP_Draw_FaceUp_Pile',1);
    fire = 1;
end
```

B.46 tDPi_Move_Init_pre.m

```
function [fire, transition] = tDPi_Move_Init_pre(transition)

global global_info;

fire = 0;
if ~isempty(global_info.INITIAL_DEAL_MOVE),
    transition.new_color = strcat('Move:TP',num2str(global_info.
        INITIAL_DEAL_MOVE{1}),':DP');
    global_info.INITIAL_DEAL_MOVE(1) = [];
    fire = 1;
end;
```

B.47 tDPi_Turn_post.m

```
function [] = tDPi_Turn_post(transition)

global global_info;

% Release playerAction resource to allow for another player action.
if isfield(global_info,'last_command_source'),
    release(global_info.last_command_source);
end;
```

B.48 tDPi_Turn_pre.m

```
function [fire, transition] = tDPi_Turn_pre(transition)

global global_info;
fire = 0;
dealer_trans = get_trans('tDPi_Dealer');
% Make sure the dealer transition has fired enough times. Simply having an
% inhibitor arc is not enough as this transition seems to fire before all
% tokens are in the face-down pile.
```

```
if dealer_trans.times_fired == global_info.INITIAL_DECK_LENGTH,
    topFD = tokenArrivedLate('pDP_Draw_FaceDown_Pile',1);
    transition.selected_tokens = topFD;
    fire = 1;
    if(global_info.DISP_CHANGES && global_info.CARDS_DEALT >= global_info.
        INITIAL_DEAL_MOVE_LENGTH),
        color = get_color('pDP_Draw_FaceDown_Pile', topFD);
        disp(strcat('Turned_card',{'_'},color{1},{'_'},'at',{'_DP'}));
    end;
end;
```

B.49 tMC_DP_Move_Siphon_pre.m

```
function [fire, transition] = tMC_DP_Move_Siphon_pre(transition)

% Siphon for the DP Move command. Should not attempt to move if there are
% no tokens in the FaceUp pile.
fire = 0;
if isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
    fire = 1;
end
```

B.50 tMC_FP_Move_Siphon_pre.m

```
function [fire, transition] = tMC_FP_Move_Siphon_pre(transition)

% Siphon for the FP Move command. Checks the length of the command, and if
% the length is valid, it will check if the destination is valid.

global global_info;

fire = 0;
moveToken = tokenArrivedLate('pMC_FP_Move',1);
[moveCmd, ~] = splitCommand(get_color('pMC_FP_Move',moveToken));

if(length(moveCmd) < 3 || ~ismember(moveCmd{3}, global_info.FP_PILES)),
    transition.selected_tokens = moveToken;
    fire = 1;
end
```

B.51 tMC_Out_Buffer_Siphon_pre.m

```
function [fire, transition] = tMC_Out_Buffer_Siphon_pre(transition)

% Siphon for the Out command from all modules. Will first check if the
% length of the command is correct, and then if the destination is valid.

global global_info;

fire = 0;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;

moveToken = tokenArrivedEarly('pMC_Out_Buffer',1);
tokenColor = get_color('pMC_Out_Buffer',moveToken);
[command, ~] = splitCommand(tokenColor);

if length(tokenColor) ~= 2 || ~ismember(command{2}, global_info.FP_TP_PILES),
    transition.selected_tokens = moveToken;
    fire = 1;
    return;
end;
```

B.52 tMC_TP_Move_Siphon_pre.m

```
function [fire, transition] = tMC_TP_Move_Siphon_pre(transition)

% Siphon for the TP Move command. Checks the length of the command, and if
% the length is valid, it will check if the destination is valid.

global global_info;

fire = 0;
moveToken = tokenArrivedLate('pMC_TP_Move',1);
moveColor = get_color('pMC_TP_Move',moveToken);
[moveCmd, ~] = splitCommand(moveColor);

if length(moveCmd) < 4 || ~ismember(moveCmd{3},global_info.TP_PILES),
    transition.selected_tokens = moveToken;
    fire = 1;
end;
```

B.53 tMC_TP_Turn_Siphon_pre.m

```
function [fire, transition] = tMC_TP_Turn_Siphon_pre(transition)

% Siphon for the TP Turn command. Checks the length of the command, and if
% the length is valid, it will check if the destination is valid.

global global_info;

fire = 0;
moveToken = tokenArrivedLate('pMC_TP_Turn',1);
moveCmd = get_color('pMC_TP_Turn',moveToken);

if length(moveCmd) < 1,
    fire = 1;
else,
    cmdSplit = strsplit(moveCmd{1},':');
    if length(cmdSplit) ~= 2 || ~ismember(cmdSplit{2}, global_info.TP_PILES),
        transition.selected_tokens = moveToken;
        fire = 1;
    end;
end;
```

B.54 tPBe_DP_Move_pre.m

```
function [fire, transition] = tPBe_DP_Move_pre(transition)

global global_info;
fire = 0;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;
moveToken = tokenArrivedLate('pPB_Cmd', 1);
if isempty(moveToken),
    return;
end;
moveColor = get_color('pPB_Cmd', moveToken);

[playerAction] = request(transition.name, {'playerAction', 1});
if strcmp(moveColor,'DP_Move') && playerAction,
    movesLeft = length(global_info.BOT_DP_MOVES);
    if movesLeft == 0,
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end

    vistoken = tokenArrivedLate('pDP_Draw_FaceUp_Pile',1);
    if ~vistoken,
```

```
24          global_info.BOT_ACTIONS_NEW_CMD = 1;
25          return;
26      end;
27      moveTo = randi(movesLeft);
28      dest = global_info.BOT_DP_MOVES{moveTo};
29      command = strcat('Move:',dest,':DP');
30
31      color = get_color('pDP_Draw_FaceUp_Pile',vistoken);
32      color = color{1};
33
34      if checkCommand_Move({command;color},'',transition.name,'DP_ErrorMsg'),
35          transition.selected_tokens = moveToken;
36          transition.new_color = command;
37          transition.override = 1;
38          fire = 1;
39          return;
40      end;
41      global_info.BOT_DP_MOVES(moveTo) = [];
42  end;
```

B.55 tPBe_DP_Turn_pre.m

```
1   function [fire, transition] = tPBe_DP_Turn_pre(transition)
2
3   global global_info;
4   fire = 0;
5   if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
6       return;
7   end;
8   moveToken = tokenArrivedLate('pPB_Cmd', 1);
9   if isempty(moveToken),
10      return;
11  end;
12  moveColor = get_color('pPB_Cmd', moveToken);
13
14  [playerAction] = request(transition.name, {'playerAction', 1});
15  if strcmp(moveColor,'DP_Turn') && playerAction,
16      if isempty(tokIDs('pDP_Draw_FaceDown_Pile')) && ...
17              isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
18          global_info.BOT_ACTIONS_NEW_CMD = 1;
19          return;
20      end;
21      global_info.last_command_source = transition.name;
22      transition.selected_tokens = moveToken;
23      transition.override = 1;
24      fire = 1;
25  end;
```

B.56 tPBe_FP_Move_pre.m

```
1   function [fire, transition] = tPBe_FP_Move_pre(transition)
2
3   global global_info;
4   fire = 0;
5   if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
6       return;
7   end;
8   moveToken = tokenArrivedLate('pPB_Cmd', 1);
9   if isempty(moveToken),
10      return;
11  end;
12  moveColor = get_color('pPB_Cmd', moveToken);
13  moveColor = moveColor{1};
14  [playerAction] = request(transition.name, {'playerAction', 1});
15  if ~isempty(strfind(moveColor,'FP_Move')) && playerAction,
16      movesLeft = length(global_info.BOT_FP_MOVES);
17      if movesLeft == 0,
18          global_info.BOT_ACTIONS_NEW_CMD = 1;
19          return;
```

```
20      end
21      cmd_split = strsplit(moveColor,':');
22      suit_abbr = cmd_split{2};
23      global_suit = global_info.SUITS.(suit_abbr);
24      vistoken = tokenArrivedLate(strcat('pFP_',global_suit{1},'_Pile'),1);
25      if ~vistoken,
26          global_info.BOT_ACTIONS_NEW_CMD = 1;
27          return;
28      end;
29      moveTo = randi(movesLeft);
30      dest = global_info.BOT_FP_MOVES{moveTo};
31
32      command = strcat('Move:',dest,strcat(':FP',suit_abbr));
33
34      color = get_color(strcat('pFP_',global_suit{1},'_Pile'),vistoken);
35      color = color{1};
36      if checkCommand_Move({command;color},'',transition.name,strcat('FP_',
            suit_abbr,'_ErrorMsg')),
37          transition.selected_tokens = moveToken;
38          transition.new_color = command;
39          transition.override = 1;
40          fire = 1;
41          return;
42      end;
43      global_info.BOT_FP_MOVES(moveTo) = [];
44  end;
```

B.57 tPBe_TP_Move_pre.m

```
1   function [fire, transition] = tPBe_TP_Move_pre(transition)
2
3   global global_info;
4   fire = 0;
5   if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
6       return;
7   end;
8   moveToken = tokenArrivedLate('pPB_Cmd', 1);
9   if isempty(moveToken),
10      return;
11  end;
12  moveColor = get_color('pPB_Cmd', moveToken);
13  moveColor = moveColor{1};
14  [playerAction] = request(transition.name, {'playerAction', 1});
15  if ~isempty(strfind(moveColor,'TP_Move')) && playerAction,
16      movesLeft = length(global_info.BOT_TP_MOVES);
17      if movesLeft == 0,
18          global_info.BOT_ACTIONS_NEW_CMD = 1;
19          return;
20      end
21      cmd_split = strsplit(moveColor,':');
22      tableau = cmd_split{2};
23      lenTokens = length(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile')));
24
25      if lenTokens == 0,
26          if ~isempty(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))),
27              global_info.BOT_NEXT_CMD = strcat('TP_Turn:',tableau);
28          end;
29          global_info.BOT_ACTIONS_NEW_CMD = 1;
30          return;
31      end;
32      moveTo = randi(movesLeft);
33      dest = global_info.BOT_TP_MOVES{moveTo};
34      if strcmp(dest,tableau),
35          global_info.BOT_TP_MOVES(moveTo) = [];
36          moveTo = randi(movesLeft);
37          dest = global_info.BOT_TP_MOVES{moveTo};
38      end;
39
40
41      % 20% of the time the bot will attempt to move partial amount of cards.
42      % 80% of the time it will attempt to move all.
43      if lenTokens > 1 && ismember(dest, global_info.TP_PILES),
44          if randi(100) <= global_info.BOT_ACTIONS_TP_FULL_PARTIAL_MOVE,
```

```
45              amount = lenTokens;
46          else,
47              amount = randi(lenTokens-1);
48          end
49      else,
50          amount = 1;
51      end;
52      command = strcat('Move:',dest,':TP',tableau,':',num2str(amount));
53
54      % The top card to be moved is used to check validity of the command.
55      vistoken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'), amount)
            ;
56      vistoken = vistoken(amount);
57      color = get_color(strcat('pTP_',tableau,'_FaceUp_Pile'),vistoken);
58      color = color{1};
59      if checkCommand_Move({command;color},'',transition.name,strcat('TP_',
            tableau,'_ErrorMsg')),
60          transition.selected_tokens = moveToken;
61          transition.new_color = command;
62          transition.override = 1;
63          fire = 1;
64          return;
65      end;
66      global_info.BOT_TP_MOVES(moveTo) = [];
67  end;
```

B.58 tPBe_TP_Turn_pre.m

```
1   function [fire, transition] = tPBe_TP_Turn_pre(transition)
2
3   global global_info;
4   fire = 0;
5   if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
6       return;
7   end;
8   moveToken = tokenArrivedLate('pPB_Cmd', 1);
9   if isempty(moveToken),
10      return;
11  end;
12
13  moveColor = get_color('pPB_Cmd', moveToken);
14  moveColor = moveColor{1};
15
16  [playerAction] = request(transition.name, {'playerAction', 1});
17  if ~isempty(strfind(moveColor,'TP_Turn')) && playerAction,
18
19      cmd_split = strsplit(moveColor,':');
20      tableau = cmd_split{2};
21      if ~isempty(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile'))) || ...
22              isempty(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))),
23
24          global_info.BOT_ACTIONS_NEW_CMD = 1;
25          return;
26      end;
27
28      command = strcat('Turn:TP',tableau);
29
30      global_info.last_command_source = transition.name;
31      transition.selected_tokens = moveToken;
32      transition.new_color = command;
33      transition.override = 1;
34      fire = 1;
35      return;
36  end;
```

B.59 tPBi_Gen_pre.m

```
function [fire, transition] = tPBi_Gen_pre(transition)

fire = 0;
global global_info;
global PN;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;
% Only one resource used, thus we can check directly in the internal
% data structure of the resource.
if global_info.BOT_ENABLED && global_info.BOT_ACTIONS_NEW_CMD && ...
        PN.system_resources.instance_usage(1,1) == 0,
    if isempty(global_info.BOT_NEXT_CMD),
        rndNum = randi(100);
        source = '';
        if rndNum <= global_info.BOT_ACTIONS(1) && ...
                ~strcmp(global_info.BOT_LAST_CMD,'DP_Turn'),
            action = 'DP_Turn';
        elseif rndNum <= global_info.BOT_ACTIONS(2) && ...
                ~strcmp(global_info.BOT_LAST_CMD,'DP_Move'),
            action = 'DP_Move';

            if randi(100) <= global_info.BOT_ACTIONS_TP_FP,
                global_info.BOT_DP_MOVES = global_info.TP_PILES;
            else,
                global_info.BOT_DP_MOVES = global_info.FP_PILES;
            end
        elseif rndNum <= global_info.BOT_ACTIONS(3) && ...
                ~strcmp(global_info.BOT_LAST_CMD,'FP_Move'),
            action = 'FP_Move';
            suits = {':C', ':D', ':H', ':S'};
            source = suits(randi(length(suits)));
            global_info.BOT_FP_MOVES = global_info.TP_PILES;
        elseif rndNum <= global_info.BOT_ACTIONS(4) && ...
                ~strcmp(global_info.BOT_LAST_CMD,'TP_Turn'),
            action = 'TP_Turn';
            source = strcat(':',num2str(randi(7)));
        else
            action = 'TP_Move';
            source = strcat(':',num2str(randi(7)));
            if randi(100) <= global_info.BOT_ACTIONS_TP_FP,
                global_info.BOT_TP_MOVES = global_info.TP_PILES;
            else,
                global_info.BOT_TP_MOVES = global_info.FP_PILES;
            end
        end;
        transition.new_color = strcat(action,source);
    else,
        transition.new_color = global_info.BOT_NEXT_CMD;
    end;

    % Reset ongoing commands.
    global_info.BOT_ACTIONS_NEW_CMD = 0;
    global_info.BOT_NEXT_CMD = '';

    transition.override = 1;
    fire = 1;
end;
```

B.60 tPBi_Gen_Stop_pre.m

```
function [fire, transition] = tPBi_Gen_pre(transition)

global global_info;
fire = 0;
if ~global_info.BOT_ENABLED,
    fire = 1;
end;
```

B.61 tPBi_Siphon_pre.m

```
function [fire, transition] = tPBi_Siphon_pre(transition)

% Remove unused commands from pPB_Cmd.
global global_info;
fire = 0;
if length(tokIDs('pPB_Cmd')) > 1 || ~global_info.BOT_ENABLED,
    transition.selected_tokens = tokenArrivedEarly('pPB_Cmd', 1);
    fire = 1;
end;
```

B.62 tPe_DP_Move_pre.m

```
function [fire, transition] = tPe_DP_Move_pre(transition)


global global_info;
fire = 0;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;

[playerAction] = request(transition.name, {'playerAction', 1});
if global_info.DP_Move_Btn ~= false && playerAction,
    global_info.DP_Move_Btn = false;
    dest = get_handle('DP_Move_Location','String');
    command = strcat('Move:',dest,':DP');
    vistoken = tokenArrivedLate('pDP_Draw_FaceUp_Pile',1);
    if vistoken,
        color = get_color('pDP_Draw_FaceUp_Pile',vistoken);
        color = color{1};

        if checkCommand_Move({command;color},'',transition.name,'DP_ErrorMsg')
            ,
            transition.new_color = command;
            fire = 1;
        end;
    end;
end
```

B.63 tPe_DP_Turn_pre.m

```
function [fire, transition] = tPe_DP_Turn_pre(transition)

global global_info;
pause(0.01); % Halts execution in the main loop to allow to check for events.
fire = 0;
if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
    return;
end;

[playerAction] = request(transition.name, {'playerAction', 1});
if global_info.DP_Turn_Btn ~= false && playerAction,
    global_info.DP_Turn_Btn = false;
    global_info.last_command_source = transition.name;
    fire = 1;
end;
```