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.
destroyer A transition which only has inputs. Opposite of generator.

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

Klondike, or Solitaire as it has been called in North America, is one of the most popular patience 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.

Solitaire is played with a standard deck of 52 cards and no Jokers. We will describe the layout and rules of the game

At the beginning of the game, the deck is shuffled and the cards are laid out in seven tableau piles from left to right. Each pile contains one additional card compared to the previous one. There is one upturned card at the beginning of each pile. The first pile to the left contains one card facing up, the second contains one card facing down and one up, the third pile contains two cards facing down and one up, continuing to the seventh pile respectively, which contains six cards facing down and one facing up. Tableau piles can be built down by alternating colors. Partial or complete piles can be moved on top of other piles as long as long as the constraint of alternating card color is respected. Empty piles can be filled with a King or a pile of cards that starts with a King.

We build up the piles by stacking cards of the same suit starting from Ace and finishing with King. The aim of the game is to build up a stack of cards of the same suit. When this is accomplished, the goal is to move this stack to foundation where the Ace of the suit had previously been placed.

The remainder of the deck is dealt by turning one card at a time from the draw pile. When all the cards from the draw pile have been turned, the draw pile can be reset.

The layout of the piles is illustrated in figure 1.

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`

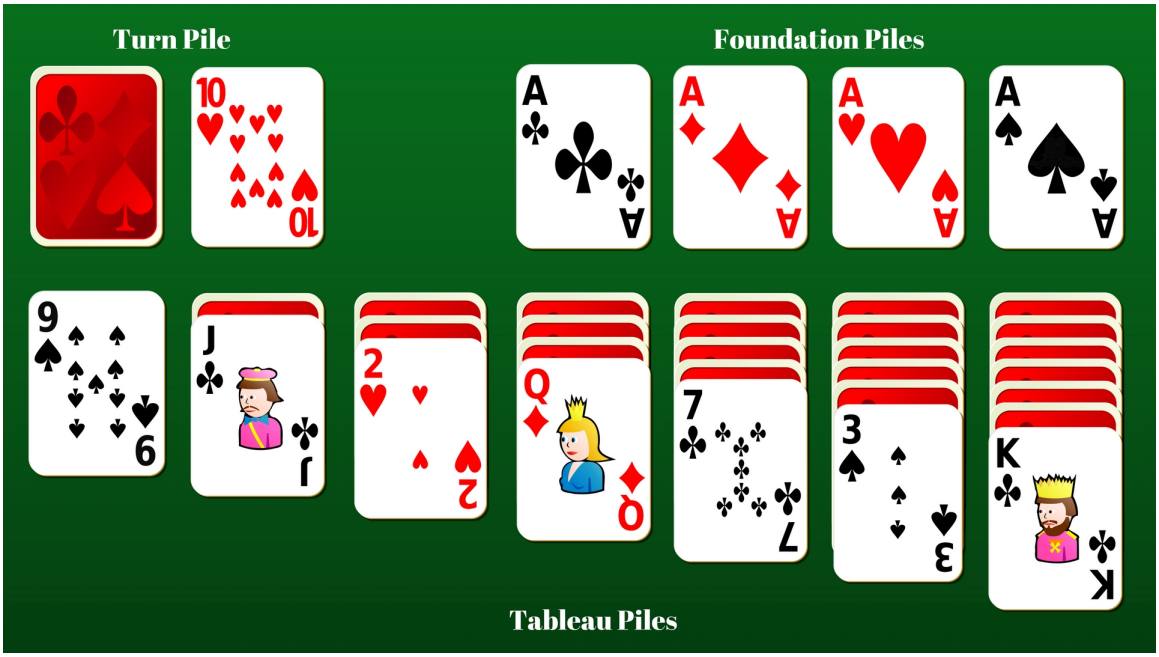

Fig. 1. Illustration of card layout in Solitaire

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
```

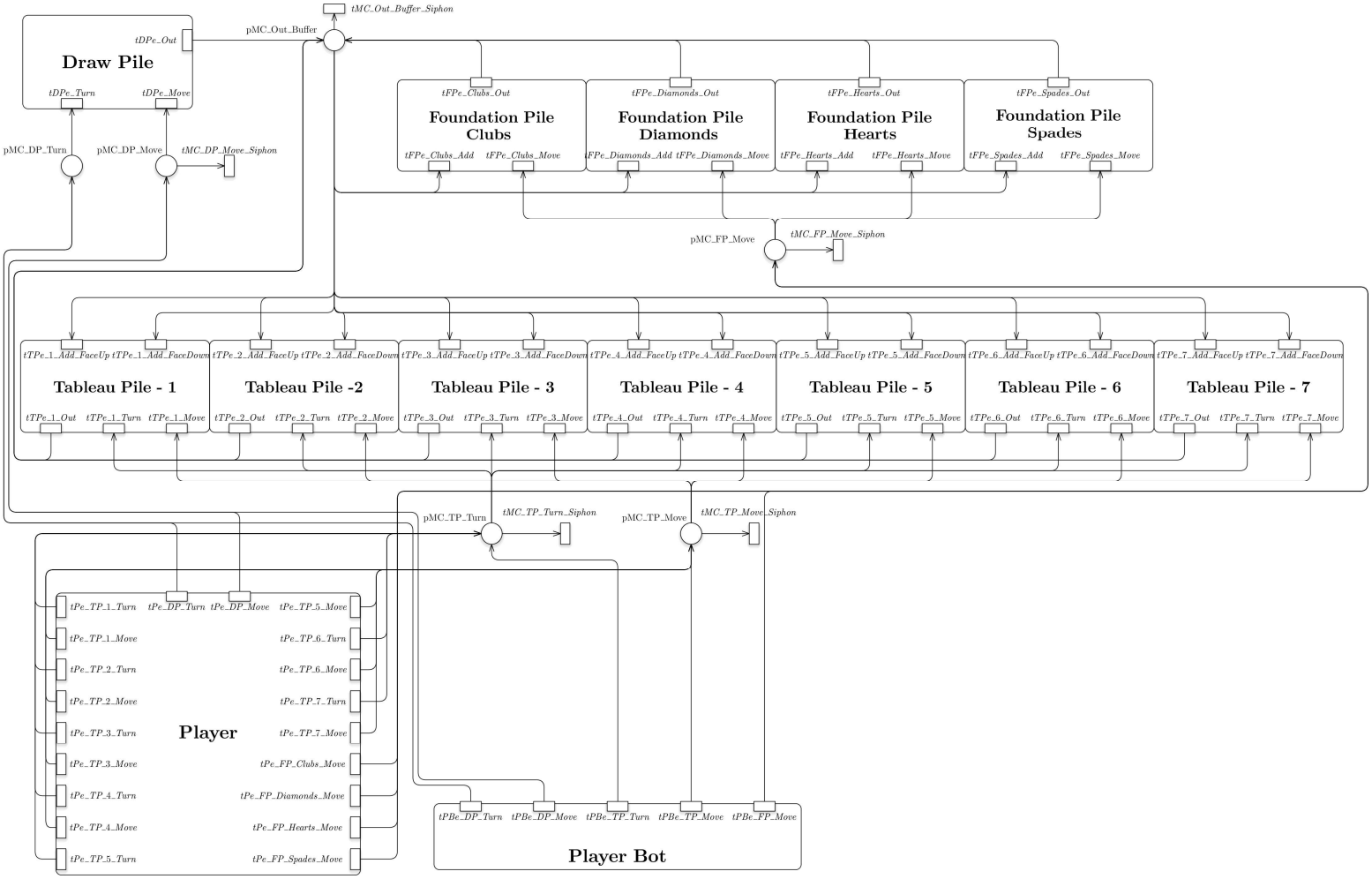

Fig. 2. The complete model - Without the internal components of the modules.

The model developed is pretty large, and contains 94 transition and 42 places. It is developed using the modular 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 3, 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`.

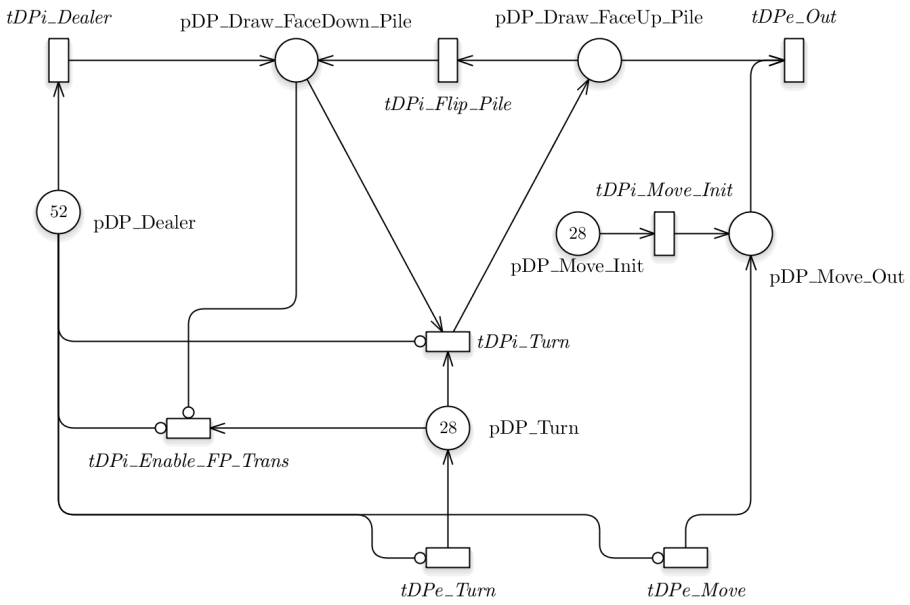

Fig. 3. Draw Pile Module

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 a destroyer, 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 destroyer, 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 4. 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.

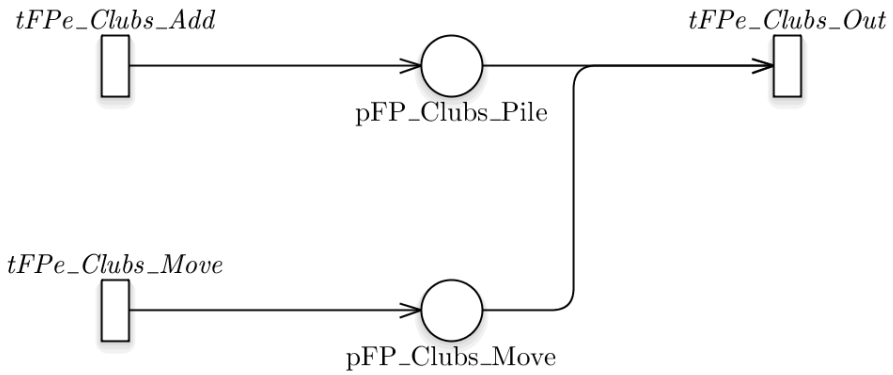

Fig. 4. Foundation Pile Module

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 36-45

```
% 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 5. 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 4.4 and 2.4. All the pre- and post-processor files are shared between all piles.

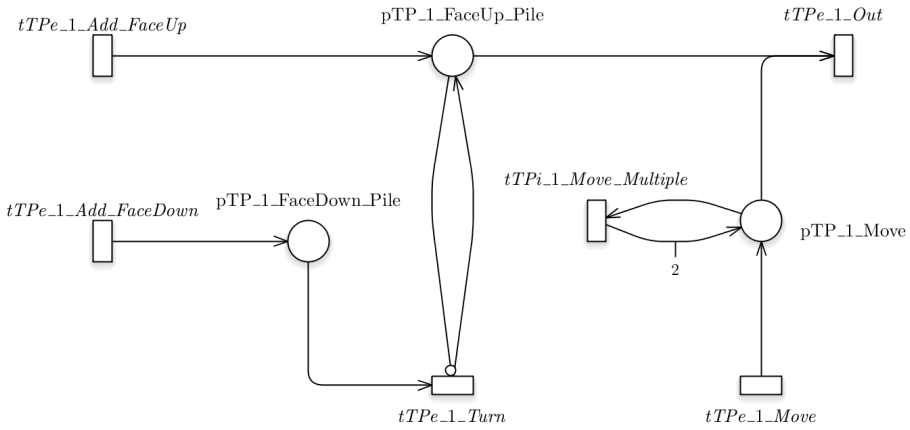

Fig. 5. Tableau Pile Module

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 4.4.

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

```
[tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
% Can only add FaceDown cards during the initial dealing.
if global_info.CARDS_DEALT >= global_info.INITIAL_DEAL_MOVE_LENGTH ...
    || length(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))) + 1 ...
    == str2double(tableau),
    return;
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. The main job of the pre-processor is to check whether the game is still in the initial phase, and if so, how many cards have been added to `pTP_1_FaceDown_Pile`. While the game is in the initial phase, there are no checks on the the suit or rank of the cards being added, but once it enters the normal phase, the pre-processor will rely on the `checkCommand_Move` function to check the validity against the actual Solitaire rules. More information about the `checkCommand_Move` function is found in chapter 3.3.1.

Listing 1.13. pre_tTPe_Add_FaceUp.m lines 7-31

```
% 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
```

Both the `tTPe_1_Add_FaceDown` and `tTPe_1_Add_FaceUp` also have post-processors which will increment the variable `global_info.CARDS_DEALT`. The post-processor for `tTPe_1_Add_FaceUp` also decrements the variable `global_info.TP_Move_Multiple_Count` which is used when multiple cards are to be moved from the pile. Listing 1.14 shows how the variable is decremented, and once the last card has been received, the `playerAction` resource will be released. Full code for the post-processor can be found in B.18. More details on moving multiple cards will follow in the next paragraphs.

Listing 1.14. post_tTPe_Add_FaceUp.m lines 5-12

```
1
2  if global_info.TP_Move_Multiple_Count <= 1,
3      if isfield(global_info,'last_command_source'),
4          release(global_info.last_command_source);
5      end
6  else,
7      global_info.TP_Move_Multiple_Count = global_info.TP_Move_Multiple_Count -
           1;
8  end;
```

The module encompasses two external inputs, there first of which is `tTPe_1_Turn`. As can be seen in figure 5, the transition is only enabled if there is at least one token in `pTP_1_FaceDown_Pile` and no tokens in `pTP_1_FaceUp_Pile`. The transitions pre-processor will check that the command in `pMC_TP_Turn` is meant for this particular pile. This is done through extracting the unique identifier using the function `get_tableau_from_transname`.

The next external input is `tTPe_1_Move`, which works similarly to the movement inputs of DP and FP, but with one big difference; it supports moving multiple cards at a time. This is achieved in conjunction with `tTPi_1_Move_Multiple` using two global variables shown in listing 1.15. Note that it is only possible to move multiple cards to other TP.

Listing 1.15. pre_tTPe_Move.m lines 11-19

```
1  if(length(moveCmd) >= 4 && strcmp(moveCmd{3},strcat('TP',tableau))),
2      amount = str2double(moveCmd{4});
3      global_info.TP_Move_Multiple_Count = amount;
4      global_info.TP_Move_Multi_Gen_Tokens = amount - 1;
5
6      transition.selected_tokens = moveToken;
7      fire = 1;
8  end
```

The same functionality could be achieved using only one variable, however by using two we make sure that the `playerAction` resource remains taken until all the cards have reached their destination. When `tTPi_1_Move_Multiple` fire it will take the token in `pTP_1_Move`, and return two new ones, essentially duplicating the token. By doing this one time less than the amount of cards to be moved, we end up with the desired number of tokens in `pTP_1_Move`. Listing 1.16 shows the pre-processor of `tTPi_1_Move_Multiple`, and listing 1.17 shows how `global_info.TP_Move_Multi_Gen_Tokens` is decremented in the `COMMON_POST` file.

Listing 1.16. pre_tTPi_Move_Multiple.m lines 6-13

```
1  if global_info.TP_Move_Multi_Gen_Tokens > 0,
2      [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
3      moveToken = tokenArrivedEarly(strcat('pTP_',tableau,'_Move'),1);
4      transition.selected_token = moveToken;
5      transition.new_color = get_color(strcat('pTP_',tableau,'_Move'),moveToken)
           ;
6      transition.override = 1;
7      fire = 1;
8  end;
```

Listing 1.17. COMMON_POST.m lines 28-34

```
1  elseif ismember(transition.name, {'tTPi_1_Move_Multiple',...
2          'tTPi_2_Move_Multiple', 'tTPi_3_Move_Multiple', ...
```

```
3           'tTPi_4_Move_Multiple', 'tTPi_5_Move_Multiple', ...
4           'tTPi_6_Move_Multiple', 'tTPi_7_Move_Multiple'}),
5       global_info.TP_Move_Multi_Gen_Tokens = ...
6           global_info.TP_Move_Multi_Gen_Tokens - 1;
7   end;
```

The only external output is `tTPe_1_Out`, which will only fire once enough tokens are generated by `tTPi_1_Move_Multiple`. In order ensure that the correct card is moved, the pre-processor will fetch the **n**-th latest arrived card. Where **n** is the number of tokens in `pTP_1_Move`. This is repeated for every token in `pTP_1_Move`. By doing this, correctness is preserved by always moving the highest ranking card first. Listing 1.18 shows the main logic of the pre-processor, the whole file is found in at B.28.

Listing 1.18. pre_tTPe_Out.m lines 6-19

```
1   if global_info.TP_Move_Multi_Gen_Tokens == 0,
2       [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
3       moveToken = tokenArrivedEarly(strcat('pTP_',tableau,'_Move'),1);
4
5       % Get the n-th latest arrived card from the face-up pile in order to
6       % move the correct card first.
7       lenMoveTokens = length(tokIDs(strcat('pTP_',tableau,'_Move')));
8       cardToken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'), ...
9           lenMoveTokens);
10      cardToken = cardToken(lenMoveTokens);
11
12      transition.selected_tokens = [moveToken cardToken];
13      fire = 1;
14  end;
```

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

	Name	Description
1	pTP_1_FaceDown_Pile	Holds the face-down cards. These are not visible to the player.
2	pTP_1_FaceUp_Pile	Holds the face-up cards. All cards are visible to the player.
3	pTP_1_Move	Buffer for move-commands
4	tTPe_1_Add_FaceDown	External input for adding cards to the face-up pile.
5	tTPe_1_Add_FaceUp	External input for adding cards to the face-down pile during the initial phase.
6	tTPe_1_Move	External input for the move-command.
7	tTPe_1_Out	External output
8	tTPe_1_Turn	External input for the turn-command.
9	tTPi_1_Move_Multiple	Internal transition which duplicates the move-command when moving multiple cards.

2.7 Player Module

The Player module is a set of 20 transitions, tightly knit with the GUI and the Module Connector. This module differs from the DP, FP and TP modules in that the arcs to the Module Connector are done in the Player module rather than in the Module Connector. By doing this, it is possible to toggle this module on/off in the variable `global_info.GUI_ENABLED`.

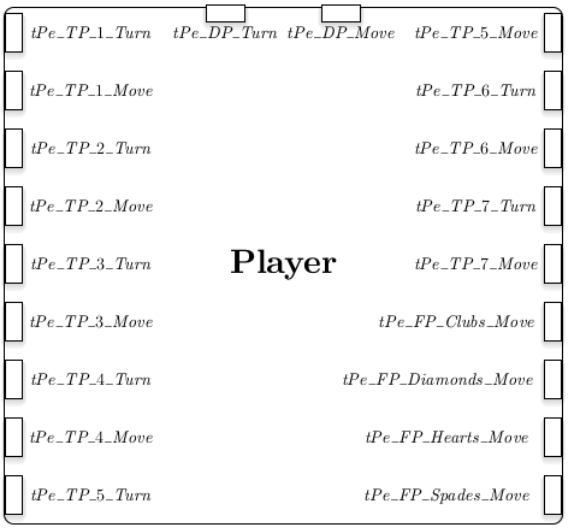

Fig. 6. Player module

An important aspect of the Player module is the acquisition of the `playerAction` resource in the pre-processor file of every transition. By doing this, the atomicity of each action is preserved, as it's not possible to start a new action before the previous one finishes executing. Resources are discussed in more detail in chapter 3.5.

In order to make the buttons of the GUI be responsive, it was necessary to introduce the `pause` command in one of the pre-processors. For this, the `tPe_DP_Turn_pre` pre-processor was chosen. It's a bit unclear what exactly happens when the `pause` command executes, but I suspect Matlab has a seperate event queue for all GUI elements. This queue is likely checked every time the main program execution is halted with `pause`. There might be a more elegant way of doing this, but none was located. Having the `pause` command execute in the pre-processor executes might not be well-suited for real-time simulations as it might introduce bias in the timings. In this project however, this solution worked very well, with no problems or unresponsiveness at all.

Listing 1.19. 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;
```

As can be seen in listing 1.19, to check if a button is pressed is actually a matter of checking whether the global variable corresponding to that button is set. In the listing above, it is accomplished by checking if `global_info.DP_Turn_Btn` is set, and if so set it back to false. A consequence of doing this, is that it's possible to queue up actions, for example by hitting the DP Turn-button when the game is in process of moving cards from one pile to another.

When issuing a new move command, its destination and possibly amount of cards to be moved is fetched directly from the GUI elements. This is done with the function `get_handle`, which is a simple extension of `get`, with the difference that GUI elements will only be fetched if `global_info.GUI_ENABLED` is set to `true`.

Table 4. Transitions used in Player

	Name	Description
1	tPe_DP_Move	Sends a move command to the DP module.
2	tPe_DP_Turn	Sends a turn command to the DP module.
3	tPe_FP_Clubs_Move	Sends a move command to the FP Clubs module.
4	tPe_FP_Diamonds_Move	Sends a move command to the FP Diamonds module.
5	tPe_FP_Hearts_Move	Sends a move command to the FP Hearts module.
6	tPe_FP_Spades_Move	Sends a move command to the FP Spades module.
7	tPe_TP_1_Move	Sends a move command to the TP 1 module.
8	tPe_TP_1_Turn	Sends a turn command to the TP 1 module.
9	tPe_TP_2_Move	Sends a move command to the TP 2 module.
10	tPe_TP_2_Turn	Sends a turn command to the TP 2 module.
11	tPe_TP_3_Move	Sends a move command to the TP 3 module.
12	tPe_TP_3_Turn	Sends a turn command to the TP 3 module.
13	tPe_TP_4_Move	Sends a move command to the TP 4 module.
14	tPe_TP_4_Turn	Sends a turn command to the TP 4 module.
15	tPe_TP_5_Move	Sends a move command to the TP 5 module.
16	tPe_TP_5_Turn	Sends a turn command to the TP 5 module.
17	tPe_TP_6_Move	Sends a move command to the TP 6 module.
18	tPe_TP_6_Turn	Sends a turn command to the TP 6 module.
19	tPe_TP_7_Move	Sends a move command to the TP 7 module.
20	tPe_TP_7_Turn	Sends a turn command to the TP 7 module.

2.8 Player Bot Module

As with the Player module, the connections to the Module Connector is done within this module. This makes it possible to toggle this module on/off with the variable `global_info.BOT_ENABLED`. The module is depicted in figure 7. In earlier iterations of the module it also had an additional place before `tPBi_Gen`, an arc going from `tPBi_Gen` to this place and a transition connected to this place. This worked fine, however it was found desirable to be able to toggle the bot on/off when doing testing and/or analysis. Therefore these where removed, and `tPBi_Gen` simply serves as a generator when its conditions in the pre-processor are met.

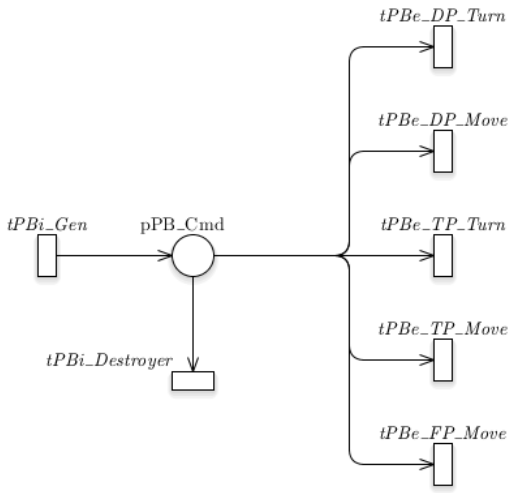

Fig. 7. Player Bot module

In addition to the requirement of `global_info.BOT_ENABLED` must be set to true, the pre-processor of `tPBi_Gen` will check that there are no active actions ongoing, and that the previously executed command has exhausted all its parameters. To check whether there are any actions currently ongoing, the global `PN` structure is checked. In this structure, all resources and their current usage can be found. Since the model only uses a single resource, it's enough to check if the first resource is taken by any transition. Listing 1.20 shows how this is done by calling `PN.system resources.instance usage(1,1)`.

Listing 1.20. tPBi_Gen_pre.m lines 9-28

```
1 % Only one resource used, thus we can check directly in the internal
2 % data structure of the resource.
3 if global_info.BOT_ENABLED && global_info.BOT_ACTIONS_NEW_CMD && ...
4         PN.system_resources.instance_usage(1,1) == 0,
5     if isempty(global_info.BOT_NEXT_CMD),
```

```
6           rndNum = randi(100);
7           source = '';
8           if rndNum <= global_info.BOT_ACTIONS(1) && ...
9                   ~strcmp(global_info.BOT_LAST_CMD, 'DP_Turn'),
10              action = 'DP_Turn';
11          elseif rndNum <= global_info.BOT_ACTIONS(2) && ...
12                  ~strcmp(global_info.BOT_LAST_CMD, 'DP_Move'),
13              action = 'DP_Move';
14
15              if randi(100) <= global_info.BOT_ACTIONS_TP_FP,
16                  global_info.BOT_DP_MOVES = global_info.TP_PILES;
17              else,
18                  global_info.BOT_DP_MOVES = global_info.FP_PILES;
19              end
20          elseif rndNum <= global_info.BOT_ACTIONS(3) && ...
```

The listing above also shows parts of how the module generates the commands. As this project is not really focused on AI programming, the logic for generating commands is very simple and serves more as a proof of concept. The bot possesses no ability to view the state of the game before making a decision about which command is most appropriate, rather it will simply pick a random number between 1 and 100, and select the command according to the distribution found in `global_info.BOT_ACTIONS`. Listing 1.21 shows the three variables which control the distribution of the bots actions.

Listing 1.21. main_simulation_file.m lines 19-29

```
1  % The bot generates a number from 1-100, this is number is used with the
2  % array below to determine which action is to be taken. The cutoffs are,
3  % DP_Turn, DP_Move, FP_Move, TP_Turn, TP_Move. Given array [20, 50, 70,
4  % 80], a number between 1-20 would attempt a DP_Turn action, 21-50 DP_Move,
5  % and so on.
6  global_info.BOT_ACTIONS = [10, 42, 44, 60];
7  % The probability of moving to a tableau pile versus a foundation pile.
8  global_info.BOT_ACTIONS_TP_FP = 7;
9  % The probability that the bot will attempt to move the full stack versus a
10 % part of it.
11 global_info.BOT_ACTIONS_TP_FULL_PARTIAL_MOVE = 35;
```

`global_info.BOT_ACTIONS` is a row vector with 4 elements numbered from 1 to 100. The bot has 5 possible commands; `DP_Turn`, `DP_Move`, `FP_Move`, `TP_Turn`, and `TP_Move`. Given a random numbers generated from 1-100, the probability distribution for the commands is as follows; `DP_Turn` between 1 and the first element of the vector, `DP_Turn` between the first and second element, `FP_Move` between the second and third element, `TP_Turn` between the third and fourth element, and `TP_Move` between the forth element and 100. `global_info.BOT_ACTIONS_TP_TP` defines how often (on average) the bot will attempt to move a card from a TP pile to another TP pile versus a FP pile. Listing 1.21 shows this value set to 15, which means that the bot will opt to move to another TP pile only 15 percent of the time.
Finally, the `global_info.BOT_ACTIONS_TP_FULL_PARTIAL_MOVE` works in a similar way, as it defines how often the TP pile will attempt to move all the cards in the face-up pile versus just a subset of them.
By manipulation these values, the likelihood of solving the game will be drastically altered. Of course having a smarter bot which took actions after considering the game state, and perhaps learned from its mistake would be must better. However, this was outside the scope of this project. Future work is discussed in more

detail in chapter 3.9.

Given that the bot continuously generates commands, even though they might be invalid for the current state of the game, I added the destroyer `tPBi_Destroyer`. This is transition is not strictly necessary for the model to work, but was still deemed advantageous to keep the number of active tokens in the Petri Net as low as possible. Listing 1.22 shows parts of the code for the destroyer.

Listing 1.22. tPBi_Destroyer_pre.m lines 3-9

```
% 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;
```

The external outputs all have pre-processors which will make them fire only if the token in `pPB_Cmd` has the correct color. In addition, some pre-processors have additional logic which makes them exhaust the different movement parameters before attempting a new command. An example of this can be seen in listing 1.23, where `global_info.BOT_ACTIONS_NEW_CMD` is set to 1 once there is no more piles to attempt to move to. `global_info.BOT_ACTIONS_NEW_CMD` is also set to 1 in the post-processor, if the command is successfully issued.

Listing 1.23. tPBe_DP_Move_pre.m lines 16-29

```
    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,
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end;
    moveTo = randi(movesLeft);
    dest = global_info.BOT_DP_MOVES{moveTo};
    command = strcat('Move:',dest,':DP');
```

Listing 1.24. COMMON_POST.m lines 24-27

```
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;
```

Table 5. Places and transitions used in Player Bot

	Name	Description
1	pPB_Cmd	Holds the next command to be executed.
2	tPBe_DP_Move	Will fetch tokens with color: DP_Move
3	tPBe_DP_Turn	Will fetch tokens with color: DP_Turn
4	tPBe_FP_Move	Will fetch tokens with color that contains: FP_Move
5	tPBe_TP_Move	Will fetch tokens with color that contains: TP_Move
6	tPBe_TP_Turn	Will fetch tokens with color that contains: TP_Turn
7	tPBi_Gen	Issues new commands once the conditions of the pre-processor are met.
8	tPBi_Destroyer	Makes sure at most 1 token is present at `pPB_Cmd` at all times.

2.9 Module Connector

The module connector is what makes all the different modules communicate. A figure of the module connector can be seen in figure 2 or for a full vertical view, in chapter A in the appendix.

By having all the external inputs and output of every module be only transitions, we achieve a clean-cut interface. This also makes testing each individual modules easier, as the desired colored input can be programmed in the preprocessor during testing.

The only pre- og post-processors of the module connector can be found in the destroyer transitions. These are not strictly necessary in the current form of the model, as the Player and Player Bot modules do a good job of keep tokens with incorrect colors out. However, it might be a good idea for future work to remove this responsibility from non-essential modules. Listing 1.25 shows the destroyer of FP move commands.

Listing 1.25. tMC_FP_Move_Destroyer_pre.m

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

% Destroyer 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
```

Table 6. Places and transitions used in Module Connector

	Name	Description
1	pMC_DP_Move	Buffer for DP move commands.
2	pMC_DP_Turn	Buffer for DP turn commands.
3	pMC_FP_Move	Buffer for FP move commands.
4	pMC_Out_Buffer	Outgoing buffer for all modules.
5	pMC_TP_Move	Buffer for TP move commands.
6	pMC_TP_Turn	Buffer for TP turn commands.
7	tMC_DP_Move_Destroyer	Destroyer for DP move commands.
8	tMC_FP_Move_Destroyer	Destroyer for FP move commands.
9	tMC_Out_Buffer_Destroyer	Destroyer the outgoing buffer.
10	tMC_TP_Move_Destroyer	Destroyer for TP move commands.
11	tMC_TP_Turn_Destroyer	Destroyer for TP turn commands.

3 Implementation

3.1 GUI

The interface developed for this project is similar to the interface of the Solitaire game. Sorting and arranging the cards is described using the same concepts that we have defined in the Solitaire Rules section.

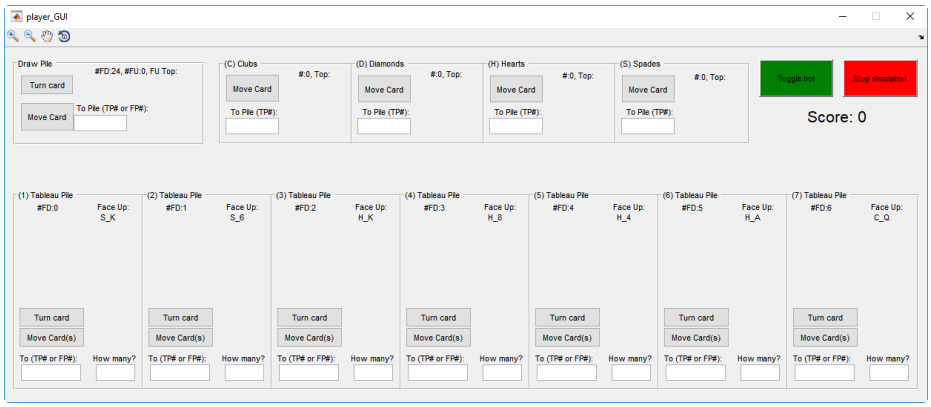

Fig. 8. GUI after inital dealing cards

Each pile is indexed in order to make it easier to simulate moving cards between piles. We can move cards between pile by writing the amount of cards to be moved in the "To Pile" box - see Figure **??**.
Should the "How many?" field be left blank, only one card will be moved. We have defined the four suits in the deck of cards:

Suit	Cards index name
Clubs	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
Diamonds	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
Hearts	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
Spades	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

The Index name of the card will show in the "Face Up" field for the Tableau piles, in the "FU Top" field for the Draw pile and "Top" for the foundation piles.

There are four foundation piles corresponding to the fours suits in the deck of cards:

Index name	Pile name
FPC	Foundation Pile Clubs
FPD	Foundation Pile Diamonds
FPH	Foundation Pile Hearts
FPS	Foundation Pile Spades

We have also given index names to the seven Tableau Piles:

Index name	Pile name
TP1	Tableau Pile 1
TP2	Tableau Pile 2
TP3	Tableau Pile 3
TP4	Tableau Pile 4
TP5	Tableau Pile 5
TP6	Tableau Pile 6
TP7	Tableau Pile 7

The Draw Pile does not have an index, as we are not able to move cards back to this pile. The "Turn card" button allows us to shuffle to the next card in the Draw Pile. This action cannot be reversed. Here we can also see how many cards in the Draw Pile are facing up (#FU), how many are facing down (#FD) as well as the index name of the card facing up that is on top.

In accordance with the game rules that we have described in the introduction, seven of the cards are revealed initially, one on top of each tableau pile. The Draw Pile will contain 24 cards that will be revealed one at a time by clicking "Turn card". When an Ace be revealed in the initial position of the dealt cards or by shuffling through the Draw Pile, it can be moved to its corresponding foundation pile (FP#). When the card that is on top of a Tableau pile is moved, the facing down card remaining on top of the pile can be turned.
We can easily move all cards from one pile to another, as illustrated in figures 9 and 10.

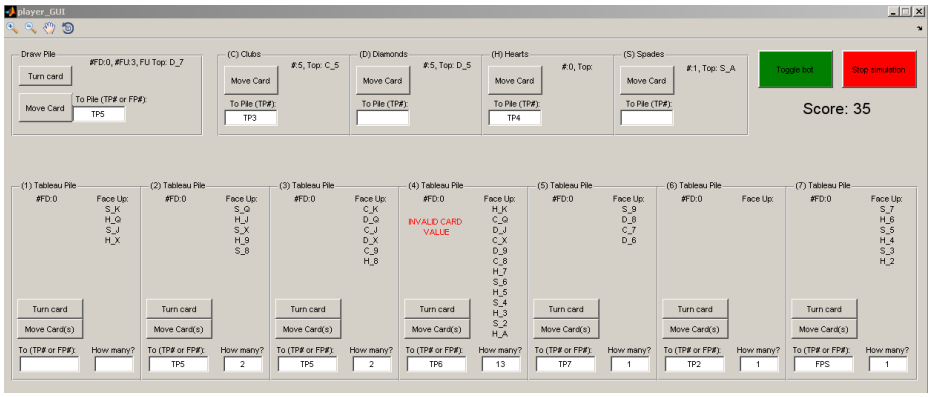

Fig. 9. A suite of 13 cards on Tableau pile 4 (TP4)

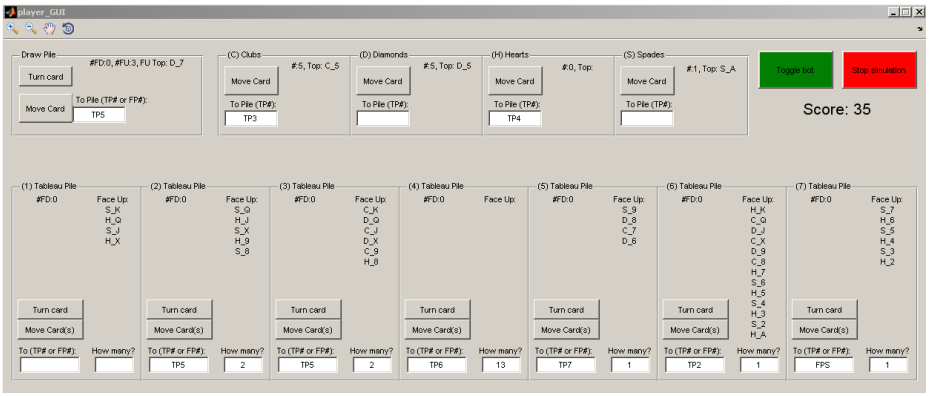

Fig. 10. The cards from TP4 are moved to TP6

Invalid moves such as moving a card from a different suit on the wrong foundation pile, or attempting to build down a tableau pile with two consecutve cards of the same colour are marked with the error message "Invalid move". In the event that there are no valid moves left, the game becomes unwinnable as illustrated in figure 11.

Fig. 11. There are no valid moves left

In the event a solution is found and the foundation piles are complete, the game ends and the final amount of points displayed as shown in figure 12.

Fig. 12. There are no valid moves left

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 13 shows a flowchart of the logic in the function.

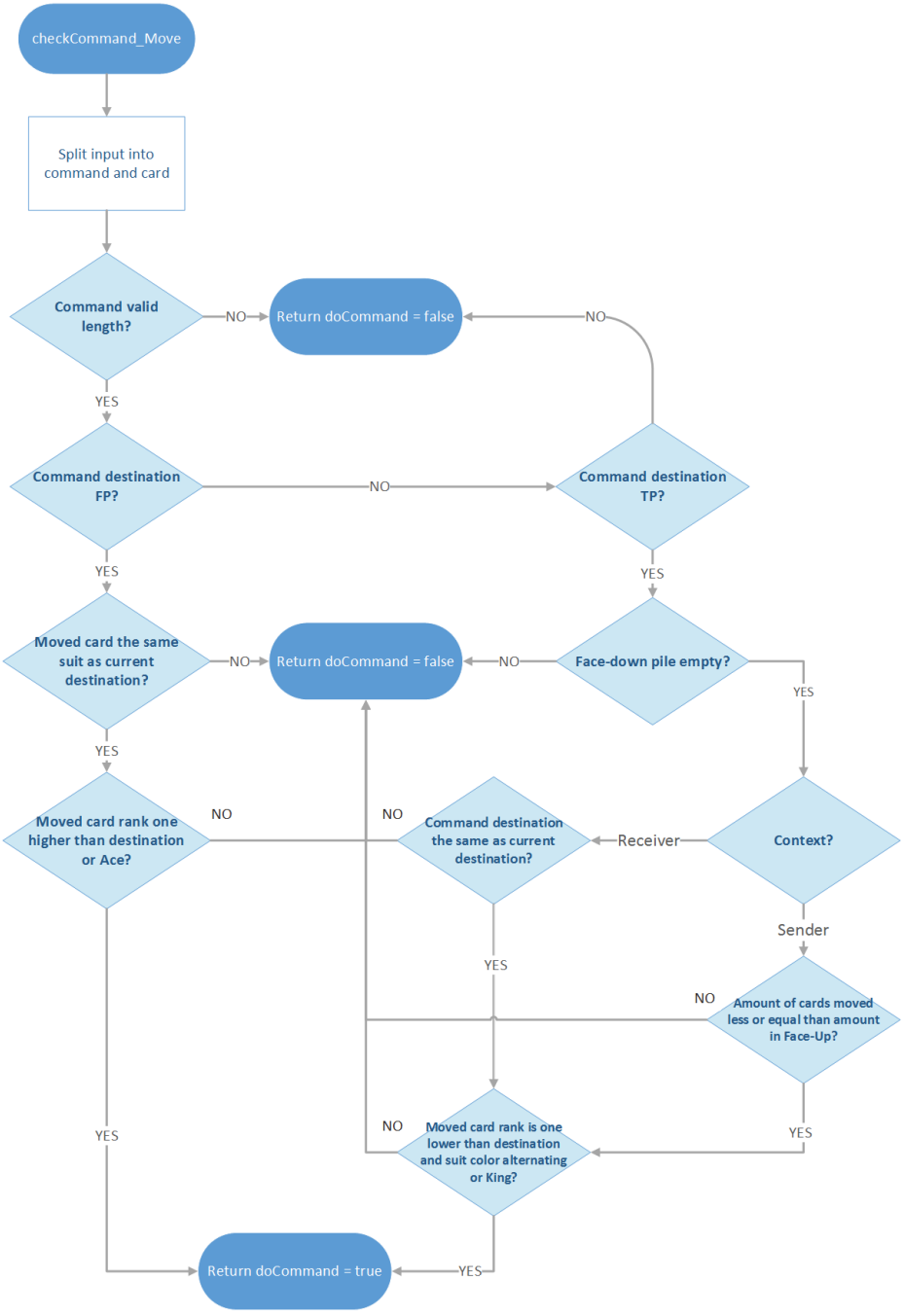

Fig. 13. Flowchart - checkCommand_Move

3.4 Initial Dealing

3.5 Resources

3.6 Moving Multiple Cards

3.7 Scoring

The standard scoring scheme in the Windows Solitaire game defines the amount of points that are awarded for moving cards between piles and is described as follows:

Move	Points
Draw pile to tableau pile	5
Draw pile to foundation pile	10
Tableau pile to foundation pile	10
Turn over tableau card	5
Foundation pile to tableau pile	-15
Reset draw pile	-100 (minimum score is 0)

3.8 Possible improvments

A major drawback of the destroyer `tMC_Out_Buffer_Destroyer` 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_Destroyer`, 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.

Lastly, a performance increase might be gained by reducing the number of transitions in the Player module, or at least introducing a new place and transition. This new place would serve as an input to all of the 20 transitions of the

Player module, whilst the new transition would serve as an input the new place. By doing this, the number enabled transitions, and pre-processors which would have to be executed would drastically drop.

3.9 Future work

Smarter bot

4 Testing, Analysis and Results

4.1 Matlab version

The project has been developed and tested in versions R2013a 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 Structural Invariants

4.3.1 Siphons

4.4 Initial Dealing

4.5 Resources

4.6 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 - Vertical view

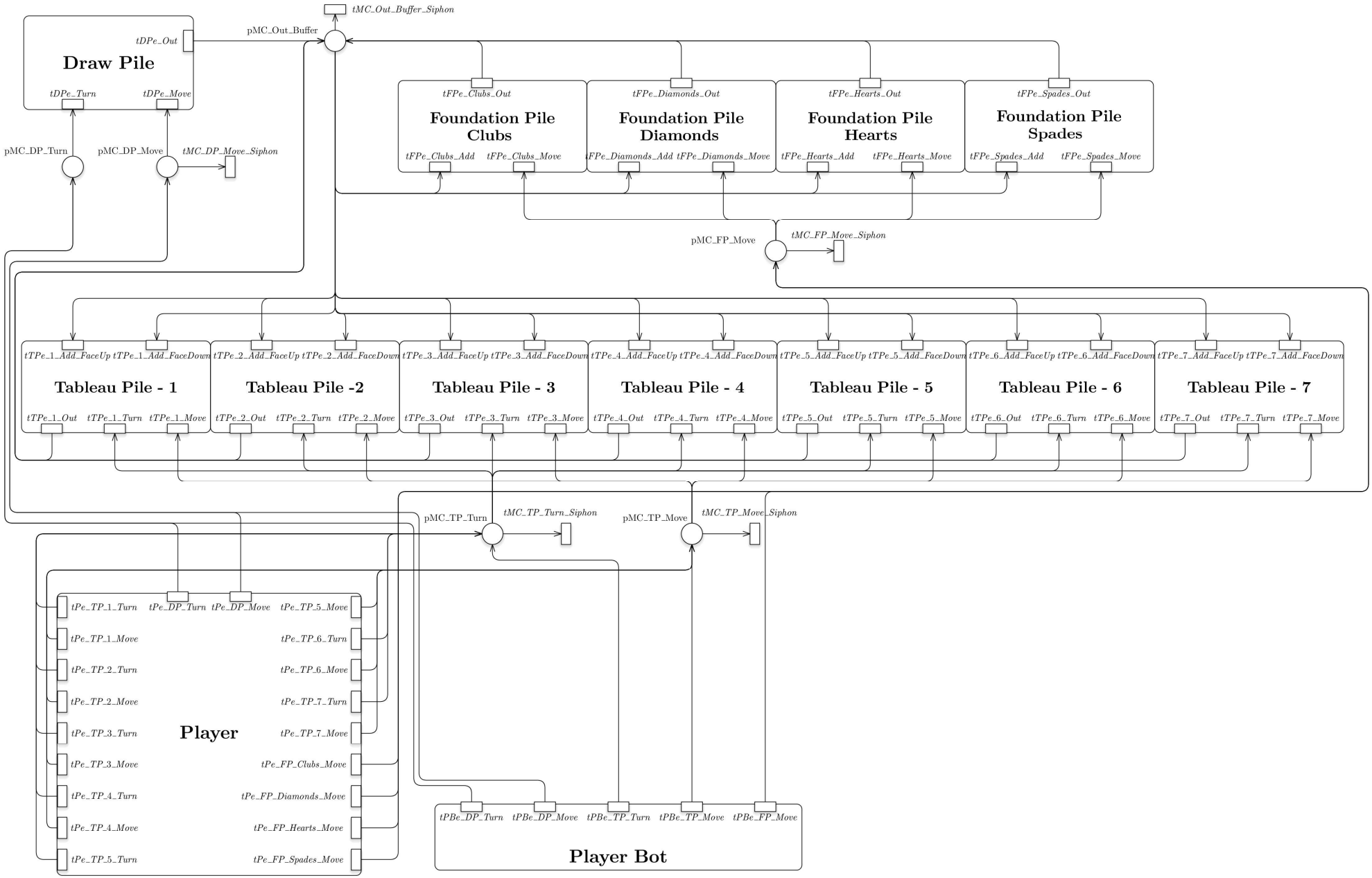

Fig. 14. The complete model in horizontal view

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
```

```
5  % Release playerAction resource to allow for another player action.
6  if ismember(transition.name, {'tTPe_1_Add_FaceDown', ...
7          'tTPe_2_Add_FaceDown', 'tTPe_3_Add_FaceDown', ...
8          'tTPe_4_Add_FaceDown', 'tTPe_5_Add_FaceDown', ...
9          'tTPe_6_Add_FaceDown', 'tTPe_7_Add_FaceDown'}),
10     global_info.CARDS_DEALT = global_info.CARDS_DEALT + 1;
11 elseif ismember(transition.name, {'tTPe_1_Add_FaceUp', ...
12         'tTPe_2_Add_FaceUp', 'tTPe_3_Add_FaceUp', ...
13         'tTPe_4_Add_FaceUp', 'tTPe_5_Add_FaceUp', ...
14         'tTPe_6_Add_FaceUp', 'tTPe_7_Add_FaceUp'}),
15     post_tTPe_Add_FaceUp(transition);
16 elseif ismember(transition.name, {
17         'tFPe_Clubs_Add','tFPe_Diamonds_Add', 'tFPe_Hearts_Add', ...
18         'tFPe_Spades_Add', 'tTPe_1_Turn', 'tTPe_2_Turn', 'tTPe_3_Turn', ...
19         'tTPe_4_Turn', 'tTPe_5_Turn', 'tTPe_6_Turn', 'tTPe_7_Turn', ...
20         'tMC_DP_Move_Destroyer', 'tMC_FP_Move_Destroyer', ...
21         'tMC_Out_Buffer_Destroyer', 'tMC_TP_Move_Destroyer', ...
22         'tMC_TP_Turn_Destroyer'}),
23     release(global_info.last_command_source);
24 elseif ismember(transition.name, {
25         'tPBe_DP_Move','tPBe_DP_Turn', 'tPBe_FP_Move','tPBe_TP_Move', ...
26         'tPBe_TP_Turn',}),
27     global_info.BOT_ACTIONS_NEW_CMD = 1;
28 elseif ismember(transition.name, {'tTPi_1_Move_Multiple',...
29         'tTPi_2_Move_Multiple', 'tTPi_3_Move_Multiple', ...
30         'tTPi_4_Move_Multiple', 'tTPi_5_Move_Multiple', ...
31         'tTPi_6_Move_Multiple', 'tTPi_7_Move_Multiple'}),
32     global_info.TP_Move_Multi_Gen_Tokens = ...
33         global_info.TP_Move_Multi_Gen_Tokens - 1;
34 end;
35
36 % Check if game is won. Win condition: 13 tokens on each of the foundation
37 % piles.
38 if(length(tokIDs('pFP_Clubs_Pile')) == 13 && ...
39         length(tokIDs('pFP_Diamonds_Pile')) == 13 && ...
40         length(tokIDs('pFP_Hearts_Pile')) == 13 && ...
41         length(tokIDs('pFP_Spades_Pile')) == 13),
42     set_handle('GameStatus', 'String', 'GAME_WON!');
43     disp('GAME_WON!');
44     global_info.STOP_SIMULATION = 1;
45 end
46
47 if global_info.CARDS_DEALT >= global_info.INITIAL_DEAL_MOVE_LENGTH,
48     if global_info.GUI_ENABLED,
49         player_update_GUI();
50     end
51 end
```

B.3 COMMON_PRE.m

```
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);
6  elseif ismember(transition.name, {'tFPe_Clubs_Move', 'tFPe_Diamonds_Move', ...
7          'tFPe_Hearts_Move', 'tFPe_Spades_Move'}),
8      [fire, transition] = pre_tFPe_Move(transition);
9  elseif ismember(transition.name, {'tPe_FP_Clubs_Move', 'tPe_FP_Diamonds_Move',
       ...
10         'tPe_FP_Hearts_Move', 'tPe_FP_Spades_Move'}),
11     [fire, transition] = pre_tPe_FP_Move(transition);
12 elseif ismember(transition.name, {'tPe_FP_Clubs_Out', 'tPe_FP_Diamonds_Out',
       ...
13         'tPe_FP_Hearts_Out', 'tPe_FP_Spades_Out'}),
14     [fire, transition] = pre_tFPe_Out(transition);
15 elseif ismember(transition.name, {'tTPe_1_Add_FaceDown', 'tTPe_2_Add_FaceDown'
       , ...
16         'tTPe_3_Add_FaceDown', 'tTPe_4_Add_FaceDown', 'tTPe_5_Add_FaceDown',
               ...
17         '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, ...
29      'pDP_Draw_FaceDown_Pile', 'tDPi_Enable_FP_Trans', 1
30      };
```

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;

if ~global_info.BOT_ENABLED,
    global_info.MAX_LOOP = 9999999;
end



%%%% 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 = 7;
% 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)

% cotree(pni, 0, 1)
```

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_Destroyer', 'tMC_DP_Move_Destroyer', ...
    'tMC_TP_Turn_Destroyer', 'tMC_TP_Move_Destroyer', 'tMC_FP_Move_Destroyer'};
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_Destroyer', 1, ...
    'pMC_DP_Move', 'tMC_DP_Move_Destroyer', 1, ...
    'pMC_TP_Turn', 'tMC_TP_Turn_Destroyer', 1, ...
    'pMC_TP_Move', 'tMC_TP_Move_Destroyer', 1, ...
    'pMC_FP_Move', 'tMC_FP_Move_Destroyer', 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_Destroyer'};
png.set_of_As = {
    'tPBi_Gen', 'pPB_Cmd', 1, ...
    'pPB_Cmd', 'tPBi_Destroyer', 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, ...
```

```
19      'tPBe_DP_Turn', 'pMC_DP_Turn', 1, ...
20      'tPBe_DP_Move', 'pMC_DP_Move', 1, ...
21      'tPBe_FP_Move', 'pMC_FP_Move', 1, ...
22      'tPBe_TP_Turn', 'pMC_TP_Turn', 1, ...
23      'tPBe_TP_Move', 'pMC_TP_Move', 1, ...
24      };
25
26  % Add connections to all 7 tableau piles %
27  % for i = 1:7
28  %     num = num2str(i);
29  %     png.set_of_As = [png.set_of_As, {strcat('tPe_TP_',num,'_Turn'),'
        pMC_TP_Turn', 1}];
30  %     png.set_of_As = [png.set_of_As, {strcat('tPe_TP_',num,'_Move'),'
        pMC_TP_Move', 1}];
31  % end;
32  %
33  % % Add connections to all 4 foundation piles %
34  % foundationpiles = {'Clubs','Diamonds','Hearts','Spades'};
35  % for i = 1:4
36  %     fp = foundationpiles(i);
37  %     png.set_of_As = [png.set_of_As, {strcat('tPe_FP_',fp{1},'_Move'),'
        pMC_FP_Move', 1}];
38  % end;
```

B.15 player_GUI.m

```
1   function varargout = player_GUI(varargin)
2   % Last Modified by GUIDE v2.5 07-Nov-2017 18:47:57
3
4   % Begin initialization code - DO NOT EDIT
5   gui_Singleton = 1;
6   gui_State = struct('gui_Name',       mfilename, ...
7                      'gui_Singleton',  gui_Singleton, ...
8                      'gui_OpeningFcn', @player_GUI_OpeningFcn, ...
9                      'gui_OutputFcn',  @player_GUI_OutputFcn, ...
10                     'gui_LayoutFcn',  [] , ...
11                     'gui_Callback',   []);
12  if nargin && ischar(varargin{1})
13      gui_State.gui_Callback = str2func(varargin{1});
14  end
15
16  if nargout
17      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
18  else
19      gui_mainfcn(gui_State, varargin{:});
20  end
21  % End initialization code - DO NOT EDIT
22
23
24  % --- Executes just before player_GUI is made visible.
25  function player_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
26  % This function has no output args, see OutputFcn.
27  % hObject    handle to figure
28  % eventdata  reserved - to be defined in a future version of MATLAB
29  % handles    structure with handles and user data (see GUIDATA)
30  % varargin   command line arguments to player_GUI (see VARARGIN)
31
32  % Choose default command line output for player_GUI
33  handles.output = hObject;
34  global global_info;
35  global_info.handles = handles;
36
37  % Define default states for all click buttons
38  global_info.DP_Turn_Btn = false;
39  global_info.DP_Move_Btn = false;
40
41  global_info.TP_1_Turn_Btn = false;
42  global_info.TP_1_Move_Btn = false;
43  global_info.TP_2_Turn_Btn = false;
44  global_info.TP_2_Move_Btn = false;
45  global_info.TP_3_Turn_Btn = false;
46  global_info.TP_3_Move_Btn = false;
47  global_info.TP_4_Turn_Btn = false;
```

```
48  global_info.TP_4_Move_Btn = false;
49  global_info.TP_5_Turn_Btn = false;
50  global_info.TP_5_Move_Btn = false;
51  global_info.TP_6_Turn_Btn = false;
52  global_info.TP_6_Move_Btn = false;
53  global_info.TP_7_Turn_Btn = false;
54  global_info.TP_7_Move_Btn = false;
55
56  global_info.FP_C_Move_Btn = false;
57  global_info.FP_D_Move_Btn = false;
58  global_info.FP_H_Move_Btn = false;
59  global_info.FP_S_Move_Btn = false;
60
61  % Update handles structure
62  guidata(hObject, handles);
63
64  % UIWAIT makes player_GUI wait for user response (see UIRESUME)
65  % uiwait(handles.figure1);
66
67  % --- Outputs from this function are returned to the command line.
68  function varargout = player_GUI_OutputFcn(hObject, eventdata, handles)
69  varargout{1} = handles.output;
70
71  % --- Executes on button press in STOPSIM.
72  function STOPSIM_Callback(hObject, eventdata, handles)
73  global global_info;
74  global_info.STOP_SIMULATION = 1;
75
76  % --- Executes on button press in TOGGLEBOT.
77  function TOGGLEBOT_Callback(hObject, eventdata, handles)
78  global global_info;
79  global_info.BOT_ENABLED = ~global_info.BOT_ENABLED;
80
81  % --- Executes on button press in DP_Turn_Btn.
82  function DP_Turn_Btn_Callback(hObject, eventdata, handles)
83  global global_info;
84  if global_info.DP_Turn_Btn == false,
85      global_info.DP_Turn_Btn = true;
86  end
87
88  % --- Executes on button press in DP_Move_Btn.
89  function DP_Move_Btn_Callback(hObject, eventdata, handles)
90  global global_info;
91  if global_info.DP_Move_Btn == false,
92      global_info.DP_Move_Btn = true;
93  end
94
95  % --- Executes on button press in FP_C_Move_Btn.
96  function FP_C_Move_Btn_Callback(hObject, eventdata, handles)
97  global global_info;
98  if global_info.FP_C_Move_Btn == false,
99      global_info.FP_C_Move_Btn = true;
100 end
101
102 % --- Executes on button press in FP_D_Move_Btn.
103 function FP_D_Move_Btn_Callback(hObject, eventdata, handles)
104 global global_info;
105 if global_info.FP_D_Move_Btn == false,
106     global_info.FP_D_Move_Btn = true;
107 end
108
109 % --- Executes on button press in FP_H_Move_Btn.
110 function FP_H_Move_Btn_Callback(hObject, eventdata, handles)
111 global global_info;
112 if global_info.FP_H_Move_Btn == false,
113     global_info.FP_H_Move_Btn = true;
114 end
115
116 % --- Executes on button press in FP_S_Move_Btn.
117 function FP_S_Move_Btn_Callback(hObject, eventdata, handles)
118 global global_info;
119 if global_info.FP_S_Move_Btn == false,
120     global_info.FP_S_Move_Btn = true;
121 end
122
123 % --- Executes on button press in TP_1_Turn_Btn.
124 function TP_1_Turn_Btn_Callback(hObject, eventdata, handles)
125 global global_info;
```

```
126 if global_info.TP_1_Turn_Btn == false,
127     global_info.TP_1_Turn_Btn = true;
128 end
129
130 % --- Executes on button press in TP_1_Move_Btn.
131 function TP_1_Move_Btn_Callback(hObject, eventdata, handles)
132 global global_info;
133 if global_info.TP_1_Move_Btn == false,
134     global_info.TP_1_Move_Btn = true;
135 end
136
137 % --- Executes on button press in TP_2_Turn_Btn.
138 function TP_2_Turn_Btn_Callback(hObject, eventdata, handles)
139 global global_info;
140 if global_info.TP_2_Turn_Btn == false,
141     global_info.TP_2_Turn_Btn = true;
142 end
143
144 % --- Executes on button press in TP_2_Move_Btn.
145 function TP_2_Move_Btn_Callback(hObject, eventdata, handles)
146 global global_info;
147 if global_info.TP_2_Move_Btn == false,
148     global_info.TP_2_Move_Btn = true;
149 end
150
151 % --- Executes on button press in TP_3_Turn_Btn.
152 function TP_3_Turn_Btn_Callback(hObject, eventdata, handles)
153 global global_info;
154 if global_info.TP_3_Turn_Btn == false,
155     global_info.TP_3_Turn_Btn = true;
156 end
157
158 % --- Executes on button press in TP_3_Move_Btn.
159 function TP_3_Move_Btn_Callback(hObject, eventdata, handles)
160 global global_info;
161 if global_info.TP_3_Move_Btn == false,
162     global_info.TP_3_Move_Btn = true;
163 end
164
165 % --- Executes on button press in TP_4_Turn_Btn.
166 function TP_4_Turn_Btn_Callback(hObject, eventdata, handles)
167 global global_info;
168 if global_info.TP_4_Turn_Btn == false,
169     global_info.TP_4_Turn_Btn = true;
170 end
171
172 % --- Executes on button press in TP_4_Move_Btn.
173 function TP_4_Move_Btn_Callback(hObject, eventdata, handles)
174 global global_info;
175 if global_info.TP_4_Move_Btn == false,
176     global_info.TP_4_Move_Btn = true;
177 end
178
179 % --- Executes on button press in TP_5_Turn_Btn.
180 function TP_5_Turn_Btn_Callback(hObject, eventdata, handles)
181 global global_info;
182 if global_info.TP_5_Turn_Btn == false,
183     global_info.TP_5_Turn_Btn = true;
184 end
185
186 % --- Executes on button press in TP_5_Move_Btn.
187 function TP_5_Move_Btn_Callback(hObject, eventdata, handles)
188 global global_info;
189 if global_info.TP_5_Move_Btn == false,
190     global_info.TP_5_Move_Btn = true;
191 end
192
193 % --- Executes on button press in TP_6_Turn_Btn.
194 function TP_6_Turn_Btn_Callback(hObject, eventdata, handles)
195 global global_info;
196 if global_info.TP_6_Turn_Btn == false,
197     global_info.TP_6_Turn_Btn = true;
198 end
199
200 % --- Executes on button press in TP_6_Move_Btn.
201 function TP_6_Move_Btn_Callback(hObject, eventdata, handles)
202 global global_info;
203 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
    'tPe_DP_Move', 'pMC_DP_Move', 1, ... % Player module
    };

% 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.17 player_update_GUI.m

```
function [] = player_update_GUI()
global global_info;

%% Clear initial game status.
if global_info.CARDS_DEALT == global_info.INITIAL_DEAL_MOVE_LENGTH,
    set_handle('GameStatus', 'String', '');
end;

%% Update Score
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;
```

```
fire = 0;
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;
    global_info.SCORE = global_info.SCORE + 10;
    if(global_info.DISP_CHANGES),
        disp(strcat('Moved_card',{'_'},card,{'_'},'from',{'_'},cmdSource,...
            {'_'},'to',{'_'},cmdDest));
    end;
end
```

B.20 pre_tFPe_Move.m

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

fire = 0;
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
```

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 = upper(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, ~, 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 = upper(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;
        end;
        if amount > length(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile'))),
            set_handle(handle_err,'String','INVALID_AMOUNT');
            return;
        end;
    end;

    command = strcat('Move:',dest,':TP',tableau,':',num2str(amount));

    vistoken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'),amount);
    vistoken = vistoken(amount);
    if vistoken,
        color = get_color(strcat('pTP_',tableau,'_FaceUp_Pile'),vistoken);
        color = color{1};
        if checkCommand_Move({command;color},'',transition.name,handle_err),
            % Need some sort of perpetual fireing.
            transition.new_color = command;
            fire = 1;
        end;
    end;
end
```

B.24 pre_tPe_TP_Turn.m

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

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

[tableau, handle_err, ~, turn_btn, ~, ~] ...
```

```
    = get_tableau_from_transname(transition.name);
[playerAction] = request(transition.name, {'playerAction', 1});
if global_info.(turn_btn) ~= false && playerAction,
    global_info.(turn_btn) = false;
    if ~isempty(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile'))),
        set_handle(handle_err,'String','FaceUp_Pile_must_be_empty');
        return;
    elseif isempty(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))),
        set_handle(handle_err,'String','FaceDown_Pile_is_empty');
        return;
    end;
    set_handle(handle_err,'String','');
    global_info.last_command_source = transition.name;
    transition.new_color = strcat('Turn:TP',tableau);
    fire = 1;
end;
```

B.25 pre_tTPe_Add_FaceDown.m

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

global global_info;
fire = 0;
[tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
% Can only add FaceDown cards during the initial dealing.
if global_info.CARDS_DEALT >= global_info.INITIAL_DEAL_MOVE_LENGTH ...
    || length(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))) + 1 ...
    == str2double(tableau),
    return;
end;
moveToken = tokenArrivedEarly('pMC_Out_Buffer',1);
[moveCmd, card] = splitCommand(get_color('pMC_Out_Buffer',moveToken));
if length(moveCmd) >= 2 && strcmp(moveCmd{2},strcat('TP',tableau)),
    transition.selected_tokens = moveToken;
    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;

    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);

    % Get the n-th latest arrived card from the face-up pile in order to
    % move the correct card first.
    lenMoveTokens = length(tokIDs(strcat('pTP_',tableau,'_Move')));
    cardToken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'), ...
        lenMoveTokens);
    cardToken = cardToken(lenMoveTokens);

```

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

B.29 pre_tTPe_Turn.m

```
1   function [fire, transition] = pre_tTPe_Turn(transition)
2
3   global global_info;
4   fire = 0;
5   moveToken = tokenArrivedLate('pMC_TP_Turn',1);
6   [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
7   turnCmd = get_color('pMC_TP_Turn',moveToken);
8
9   if(length(turnCmd) >= 1 && strcmp(turnCmd{1},strcat('Turn:TP',tableau))),
10      topCard = tokenArrivedLate(strcat('pTP_',tableau,'_FaceDown_Pile'),1);
11      transition.selected_tokens = topCard;
12      color = get_color(strcat('pTP_',tableau,'_FaceDown_Pile'), topCard);
13      transition.new_color = color{1};
14      transition.override = 1;
15      fire = 1;
16      global_info.SCORE = global_info.SCORE + 5;
17      if(global_info.DISP_CHANGES),
18          disp(strcat('Turned_card',{'_'},color{1},{'_'},'at',{'_TP'},tableau));
19      end;
20  end
```

B.30 pre_tTPi_Move_Multiple.m

```
1   function [fire, transition] = pre_tTPi_Move_Multiple(transition)
2
3   global global_info;
4   fire = 0;
5
6   if global_info.TP_Move_Multi_Gen_Tokens > 0,
7       [tableau, ~, ~, ~, ~, ~] = get_tableau_from_transname(transition.name);
8       moveToken = tokenArrivedEarly(strcat('pTP_',tableau,'_Move'),1);
9       transition.selected_token = moveToken;
10      transition.new_color = get_color(strcat('pTP_',tableau,'_Move'),moveToken)
            ;
11      transition.override = 1;
12      fire = 1;
13  end;
```

B.31 set_handle.m

```
1   function [] = set_handle(Handle, PropertyName, PropertyValue)
2       % Extend Matlab SET command to first check if GUI is enabled.
3       % SET(H,'PropertyName',PropertyValue)
4       global global_info;
5       if global_info.GUI_ENABLED,
6           set(global_info.handles.(Handle),PropertyName,PropertyValue);
7       end;
8   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, ...
    strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Out'), 1,
        ...
    strcat('tTPe_',modname,'_Move'), strcat('pTP_',modname,'_Move'), 1 ...
    strcat('pTP_',modname,'_Move'), strcat('tTPe_',modname,'_Out'), 1, ...
    strcat('tTPi_',modname,'_Move_Multiple'), strcat('pTP_',modname,'_Move'),
        2, ...
    strcat('pTP_',modname,'_Move'), strcat('tTPi_',modname,'_Move_Multiple'),
        1
    };
png.set_of_Is = {
    strcat('pTP_',modname,'_FaceUp_Pile'), strcat('tTPe_',modname,'_Turn'), 1
    };
```

B.34 tableau_pile_2_pdf.m

```
function [png] = tableau_pile_2_pdf()
modname = '2';
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, ...
```

```
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

```
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;
```

B.42 tDPi_Dealer_pre.m

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

global global_info;
fire = 0;
if ~isempty(global_info.DECK),
    if global_info.RANDOM_DECK,
        card = randi([1,length(global_info.DECK)]);
    else,
        card = 1;
    end;
    transition.new_color = global_info.DECK(card);
    global_info.DECK(card) = []; % Remove this card from the array
    fire = 1;
end;
```

B.43 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;
```

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_Destroyer_pre.m

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

% Destroyer 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_Destroyer_pre.m

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

% Destroyer 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_Destroyer_pre.m

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

% Destroyer 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_Destroyer_pre.m

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

% Destroyer 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_Destroyer_pre.m

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

% Destroyer 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,
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end;
    moveTo = randi(movesLeft);
    dest = global_info.BOT_DP_MOVES{moveTo};
    command = strcat('Move:',dest,':DP');

    color = get_color('pDP_Draw_FaceUp_Pile',vistoken);
    color = color{1};

    if checkCommand_Move({command;color},'',transition.name,'DP_ErrorMsg'),
        transition.selected_tokens = moveToken;
        transition.new_color = command;
        transition.override = 1;
        fire = 1;
        return;
    end;
    global_info.BOT_DP_MOVES(moveTo) = [];
end;
```

B.55 tPBe_DP_Turn_pre.m

```
function [fire, transition] = tPBe_DP_Turn_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_Turn') && playerAction,
    if isempty(tokIDs('pDP_Draw_FaceDown_Pile')) && ...
            isempty(tokIDs('pDP_Draw_FaceUp_Pile')),
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end;
    global_info.last_command_source = transition.name;
    transition.selected_tokens = moveToken;
    transition.override = 1;
    fire = 1;
end;
```

B.56 tPBe_FP_Move_pre.m

```
function [fire, transition] = tPBe_FP_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);
moveColor = moveColor{1};
[playerAction] = request(transition.name, {'playerAction', 1});
if ~isempty(strfind(moveColor,'FP_Move')) && playerAction,
    movesLeft = length(global_info.BOT_FP_MOVES);
    if movesLeft == 0,
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end
    cmd_split = strsplit(moveColor,':');
    suit_abbr = cmd_split{2};
    global_suit = global_info.SUITS.(suit_abbr);
    vistoken = tokenArrivedLate(strcat('pFP_',global_suit{1},'_Pile'),1);
    if ~vistoken,
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end;
    moveTo = randi(movesLeft);
    dest = global_info.BOT_FP_MOVES{moveTo};

    command = strcat('Move:',dest,strcat(':FP',suit_abbr));

    color = get_color(strcat('pFP_',global_suit{1},'_Pile'),vistoken);
    color = color{1};
    if checkCommand_Move({command;color},'',transition.name,strcat('FP_',
        suit_abbr,'_ErrorMsg')),
        transition.selected_tokens = moveToken;
        transition.new_color = command;
        transition.override = 1;
        fire = 1;
        return;
```

```
    end;
    global_info.BOT_FP_MOVES(moveTo) = [];
end;
```

B.57 tPBe_TP_Move_pre.m

```
function [fire, transition] = tPBe_TP_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);
moveColor = moveColor{1};
[playerAction] = request(transition.name, {'playerAction', 1});
if ~isempty(strfind(moveColor,'TP_Move')) && playerAction,
    movesLeft = length(global_info.BOT_TP_MOVES);
    if movesLeft == 0,
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end
    cmd_split = strsplit(moveColor,':');
    tableau = cmd_split{2};
    lenTokens = length(tokIDs(strcat('pTP_',tableau,'_FaceUp_Pile')));

    if lenTokens == 0,
        if ~isempty(tokIDs(strcat('pTP_',tableau,'_FaceDown_Pile'))),
            global_info.BOT_NEXT_CMD = strcat('TP_Turn:',tableau);
        end;
        global_info.BOT_ACTIONS_NEW_CMD = 1;
        return;
    end;
    moveTo = randi(movesLeft);
    dest = global_info.BOT_TP_MOVES{moveTo};
    if strcmp(dest,tableau),
        global_info.BOT_TP_MOVES(moveTo) = [];
        moveTo = randi(movesLeft);
        dest = global_info.BOT_TP_MOVES{moveTo};
    end;


    % 20% of the time the bot will attempt to move partial amount of cards.
    % 80% of the time it will attempt to move all.
    if lenTokens > 1 && ismember(dest, global_info.TP_PILES),
        if randi(100) <= global_info.BOT_ACTIONS_TP_FULL_PARTIAL_MOVE,
            amount = lenTokens;
        else,
            amount = randi(lenTokens-1);
        end
    else,
        amount = 1;
    end;
    command = strcat('Move:',dest,':TP',tableau,':',num2str(amount));

    % The top card to be moved is used to check validity of the command.
    vistoken = tokenArrivedLate(strcat('pTP_',tableau,'_FaceUp_Pile'), amount)
        ;
    vistoken = vistoken(amount);
    color = get_color(strcat('pTP_',tableau,'_FaceUp_Pile'),vistoken);
    color = color{1};
    if checkCommand_Move({command;color},'',transition.name,strcat('TP_',
        tableau,'_ErrorMsg')),
        transition.selected_tokens = moveToken;
        transition.new_color = command;
        transition.override = 1;
        fire = 1;
        return;
    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

```
1   function [fire, transition] = tPBi_Gen_pre(transition)
2
3   fire = 0;
4   global global_info;
5   global PN;
6   if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
7       return;
8   end;
9   % Only one resource used, thus we can check directly in the internal
10  % data structure of the resource.
11  if global_info.BOT_ENABLED && global_info.BOT_ACTIONS_NEW_CMD && ...
12          PN.system_resources.instance_usage(1,1) == 0,
13      if isempty(global_info.BOT_NEXT_CMD),
14          rndNum = randi(100);
15          source = '';
16          if rndNum <= global_info.BOT_ACTIONS(1) && ...
17                  ~strcmp(global_info.BOT_LAST_CMD,'DP_Turn'),
18              action = 'DP_Turn';
19          elseif rndNum <= global_info.BOT_ACTIONS(2) && ...
20                  ~strcmp(global_info.BOT_LAST_CMD,'DP_Move'),
21              action = 'DP_Move';
22
23              if randi(100) <= global_info.BOT_ACTIONS_TP_FP,
24                  global_info.BOT_DP_MOVES = global_info.TP_PILES;
25              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_Destroyer_pre.m

```
function [fire, transition] = tPBi_Destroyer_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.61 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 = upper(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')
            ,
```

```
21                 transition.new_color = command;
22                 fire = 1;
23             end;
24         end;
25 end
```

B.62 tPe_DP_Turn_pre.m

```
1  function [fire, transition] = tPe_DP_Turn_pre(transition)
2
3  global global_info;
4  pause(0.01); % Halts execution in the main loop to allow to check for events.
5  fire = 0;
6  if global_info.CARDS_DEALT < global_info.INITIAL_DEAL_MOVE_LENGTH,
7      return;
8  end;
9
10 [playerAction] = request(transition.name, {'playerAction', 1});
11 if global_info.DP_Turn_Btn ~= false && playerAction,
12      global_info.DP_Turn_Btn = false;
13      global_info.last_command_source = transition.name;
14      fire = 1;
15 end;
```