

Design Document – Freecell

Siv Lu, Zhuyun Xiao, Manman Lu

Apr 13, 2014

1 Rules and Logic

1.1 General Rules of Freecell

FreeCell has been a popular card game for Microsoft Windows for more than two decades. Our version is a text-based FreeCell Game which also has a board that mimics the graphic-based board.

On the board, you will see four free cells on the top left side of the board, and four foundations on the top right side of the board.

Initially, they are all empty. Your goal is to eventually move all the cards on to the foundations so that each of the four piles has cards of identical suit ordered in the same sequence in ascending suit sequence from bottom to top.

There are also eight piles of cards (originally 52 cards) facing up which you can rearrange by following the legal moves:

- The card at the end of each list is free to be moved to the free cell. However, it is better to keep the free cells empty if possible.
- One card can be moved from the end of one pile and placed on top of another pile as long as the moved card is one lower in number than the top card (the last card on the pile) of this other pile, and is of different color than that top card. For example, you can move "Heart A"(red) on top of "Spade 2?(black).
- In the case of more than one card are piled up in descending order on the top of a list: for instance, Heart 5(red), Spade 4(black), Diamond 3(red); you may move all three cards on top of a pile with top card Club 6 (black).
- Empty piles can be occupied by any card of your choice, yet do not waste the empty spaces because they can be powerful later, and sometimes crucial to your success.

- Cards from the FreeCell and piles can be moved to the foundation as long as the cards on the foundation are in ascending sequential order starting from Ace, and are of the same suit.
- The final goal of the game is either to have four piles of cards on four piles with each pile arranged in descending order from King to Ace, or the cards are placed on four foundations based on suit in ascending order. You may shuffle the cards if you are at a deadlock.

1.2 Control and Logic of Game

- If the user wants to move one card to the foundation or the free cell, the user only needs to specify one card. The program automatically checks if this card can be moved to the foundation first, and then checks if this card can be moved to free cell.
- If the user wants to move a card or a series cards to a pile, the user only need to specify the card or the first card of that series and then the pile number. The program will find the card first, and checks if it is movable, given the pile number.

2 Files and Function Prototypes

2.1 freecell_main.c

This source file contains the main function, which abstracts the logic of our game. This file also includes all the files it depends on to plat the game.

Source code:

```
int main(){
    srand(time(0));
    int num;
    char card[3];
    int success_move;

    setup_board();
    while(!check_win()){
        print_board();
        success_move = 0;
        while (!success_move){
            get_parse_input(card, &num);
            success_move = 0;
            if (num == 0){
```

```

        if (move_one_card(card)) success_move = 1;
    }
    else{
        if (move_to_position(card, num-1)) success_move = 1;
    }
}
}
printf("Congrats! You win!\n")
cleanup_board();
return 0;
}

```

2.2 freecell_board.c freecell_board.h

These two files contains functions that set up the board, shuffle the board, and clean up the board.

Function Prototypes:

```

/*This function setups the board by distributing cards in piles*/
void setup_board();

/*clean up the board by freeing allocated space*/
void cleanup_board();

/*This function is called when the player gets stuck and wants to shuffle the board.
This function shuttles the card in the piles*/
void shuffle_pile();

```

2.3 freecell_io.c freecell_io.h

These two files contains I/O operations. It gets the user's input and parse it. If the user inputs a valid card move, the function will return with setting the card and move as the user specified. If the user wants something else, the function will call other functions to perform the task. Also this file depends partially on freecell_board

Function Prototypes:

```

/*This function prints out rules of this game*/
void print_rules();

```

```

/*This function print out the board*/
void print_board();

/*This function takes user input and store it in card and num.
If you user didn't provide a num, then store it as 0
This function keeps taking input if user types invalid input*/
void get_parse_input(char* card, int* num);

```

2.4 freecell_move.c freecell_move.h

These files contain the functions that are used for moving a card, or a series of cards.

Function Prototypes:

```

/*This function checks the piles to see if the user has won
It returns 0 if not winning, 1 if winning*/
int check_win();

/*This function moves one card, according to the rules*/
int move_one_card(char* card);

/*This function moves one or more cards to a specified position,
according to the rules*/
int move_to_position(char* card, int num);

```

2.5 card.c card.h

These files contain an important data structure, "Card", of this game, and the operations on a linked list. A Card is a node in a linked list.

Function Prototypes:

```

struct card_node{
    char name[3];
    struct card_node* next;
};

typedef struct card_node Card;

/*add at the end*/
void add_node(Card **head_of_pile, char* cardname);

```

```

/*delete at the end of a pile*/
void delete_node(Card **head_of_pile);

/*try find the card with cardname in the given pile*/
Card* search_node(Card* head_of_pile, char *cardname);

/*move a series of cards from a pile to another pile*/
int append_nodes(Card* prev, Card* from_head, Card** to_pile);

/*check to see if the given card is at the end of the given pile*/
int is_end(Card* head_of_pile, char* cardname);

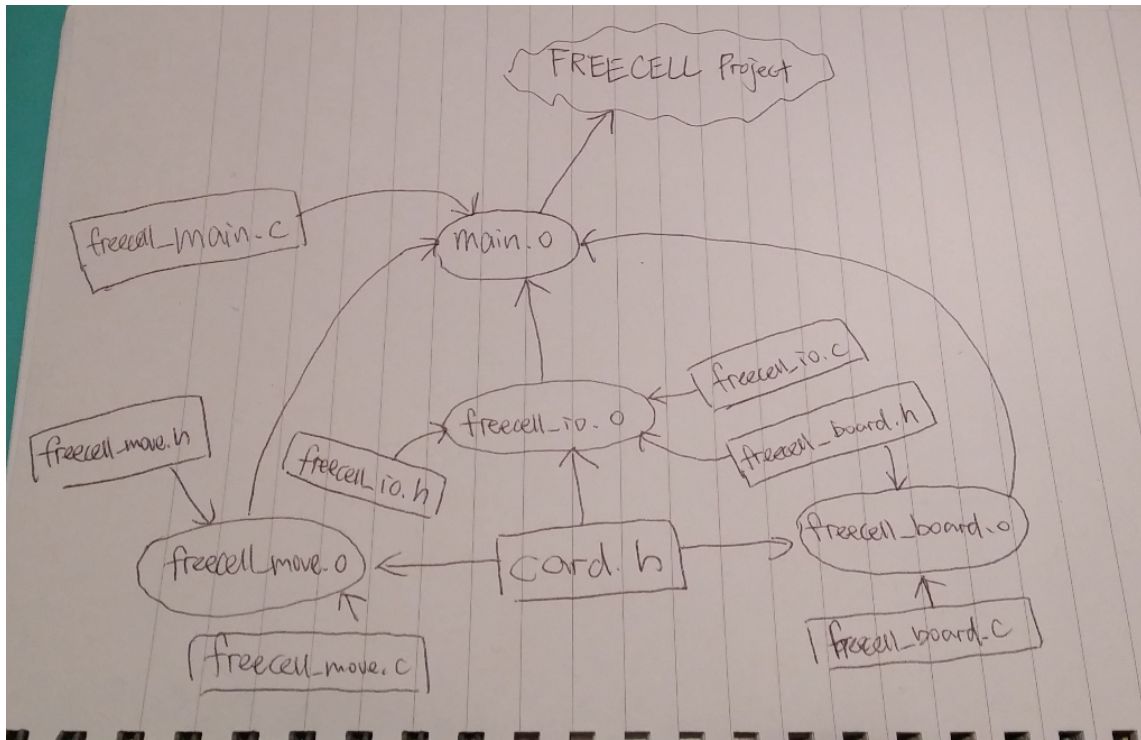
/*return the last node of a given pile*/
Card* get_last_node(Card *head_of_pile);

/*return the previous node of the given card in a given pile*/
Card* get_prev_node(char* target, Card *head_of_pile);

/*free a given list*/
void free_list(Card* head_of_pile);

```

3 Dependency Graph



NOTE: arrow means "contributes to"; rectangle means it is a header file or source code; circle means it is a object file.