Hacettepe University Department of Computer Engineering BBM203 Software Laboratory I Assignment I

Berke Bayraktar - 21992847

16.11.2020

Information About Assignment

Subject: Arrays

Deadline: 19.11.2020

Programming Language: C++

Sections

1 Problem Definition

In this ezperiment we were expected to create the game solitaire. While doing we were only allowed to use the static data structure “array”. From a more techincal perspective the problem requires that we should be able to move cards from one place to another and not vioalating game rules while doing so.

2 Solution

2.1 My Approach

My solution for this problem was creating a class called board that would be responsible for all game operations and keeping track the state of the game. And creating a second class that is responsible from displaying this board. By doing so I tried to achieve seperation of concerns. **For me most important part of the problem was** having to apply many operations on arrays and since they do not have their own methods and properties **this affected my design in such a way that** I have created low-level functions for manipulating the arrays and more higher level functions which use those low-level functions to perform the given game operation. By doing so I tried to avoid code duplication. Lastly since board is a complicated classes and has to be initialized with cards in place for starting the game, default initialization was not enough therefore I used builder design pattern and created a builder classes that would construct the board.

Inside Board class:

Properties for keeping state of the game:

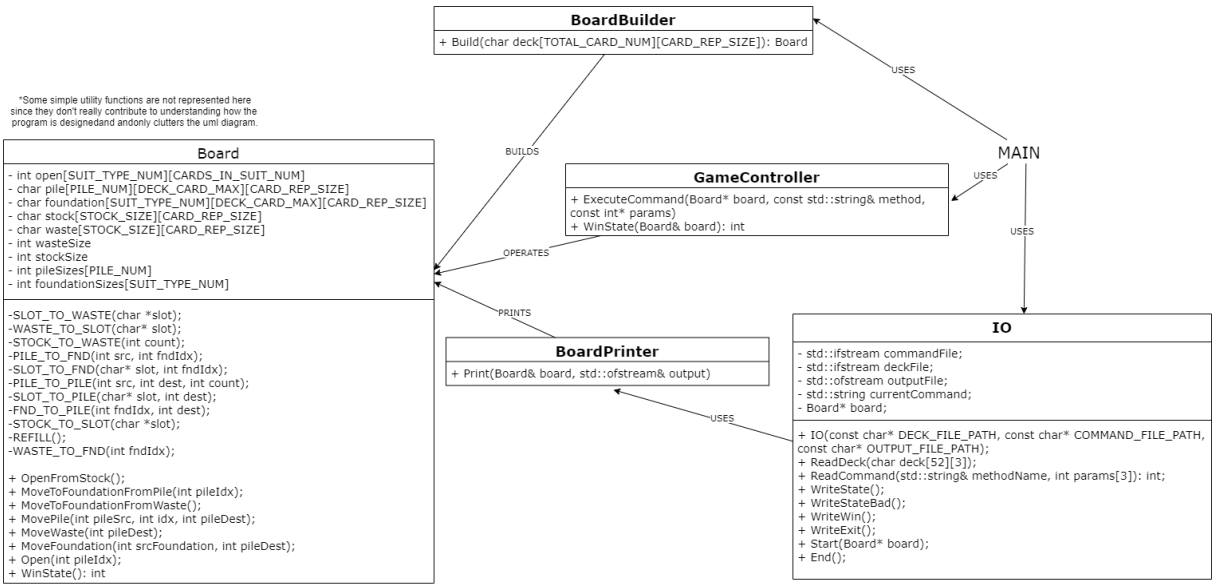
```
int open[SUIT_TYPE_NUM][CARDS_IN_SUIT_NUM];
char pile[PILE_NUM][DECK_CARD_MAX][CARD_REP_SIZE];
char foundation[SUIT_TYPE_NUM][DECK_CARD_MAX][CARD_REP_SIZE];
char stock[STOCK_SIZE][CARD_REP_SIZE];
char waste[STOCK_SIZE][CARD_REP_SIZE];

int wasteSize = 0;
int stockSize = STOCK_SIZE;
int pileSizes[PILE_NUM] = {1, 2, 3, 4, 5, 6, 7};
int foundationSizes[SUIT_TYPE_NUM] = {0,0,0,0};
```

Low level and High level methods for carrying game operations:

```
void SLOT_TO_WASTE(char *slot);
void WASTE_TO_SLOT(char* slot);
void STOCK_TO_WASTE(int count);
void PILE_TO_FND(int src, int fndIdx);
void SLOT_TO_FND(char* slot, int fndIdx);
void PILE_TO_PILE(int src, int dest, int count);
void SLOT_TO_PILE(char* slot, int dest);
void FND_TO_PILE(int fndIdx, int dest);
void STOCK_TO_SLOT(char *slot);
void REFILL();
void WASTE_TO_FND(int fndIdx);
```

```
void OpenFromStock();
void MoveToFoundationFromPile(int pileIdx);
void MoveToFoundationFromWaste();
void MovePile(int pileSrc, int idx, int pileDest);
void MoveWaste(int pileDest);
void MoveFoundation(int srcFoundation, int pileDest);
void Open(int pileIdx);
int WinState();
```

2.2 Class Diagram and Class Explanations:



Board: Responsible for carrying out game operations.

BoardBuilder: Responsible for initializing the board.

BoardPrinter: Responsible for displaying the board.

Note: Both BoardBuilder and BoardPrinter has to be a friend of class board because they need access to private properites of the board class.

GameController: Responsible for routing the command from IO to right operation on board. Information flows as: IO->Main->GameController->Board

IO: Responsible for reading from file and writing to output.

2.2.3 Arrays

I have used array in almost everywhere in my program few usages are: storing the parameters parsed from command string, storing the deck imported from deck.txt. But main usage of arrays were for representing the game state in board class. This class holds multi-dimensional character and integer arrays. Character arrays are essentialy for representing a “card” in the game e.g H12. Integer arrays are for keeping the size of their corresponding character arrays. As an example:

```
char pile[PILE_NUM][DECK_CARD_MAX][CARD_REP_SIZE]
int pileSizes[PILE_NUM]
```

consider these lines.

1 - pile[0][2] would return a char* to third card in the first pile.

2 - pile[3] would return how many cards are there in 4th pile.