Casino – C++

TECHNICAL MANUAL

October 24, 2018

Abish Jha
Ramapo College of New Jersey

Contents

1. Bug Report
2. Description of data structures/classes
3. How to Run
4. Screen Shots

Bug Report

There are no known bugs.

Description of data structures/classes

The data structures used in the game are mostly standard C++ data structures which include vector, set and string. The game makes use of one custom data structure which is Move.

Move is a class with three member variables:

- The first is the action to be performed which can be any of the five actions possible i.e. build, multi build, extend build, capture, and trail. It is stored as a string.
- The second is the hand card involved in the move. This is a string as well and should be two characters long, the first representing the suit and the second representing the value.
- The third is the loose cards involved in the move. This is a vector of strings where every element is either a build or a loose card involved in the move.

This class also provides member function for accessing or assigning and printing the move results in a human readable way.

Classes:

1. Deck

   This class hold the cards for the game life time. It also has functionality to randomly shuffle the deck when it is auto generated by the class. The deck can also be initialized through a space separated string containing the cards or from a text file where every card in the deck in on a new line.

2. Player

   The game uses two objects of class player, one for the human and the other for the computer. There is a ‘is_human’ flag in the class which differentiates the human user from the computer user. This class also holds the player’s hand cards, captured cards and the tournament score.

3. Game

   This is the main entry point of the game as the name suggests. This class asks the user if they want to start a new game or load a game. Then it proceeds to load a game state as per chosen by the user. It then prints a menu for the user and proceeds with the game. This class creates objects

of the Deck, Player, Logic, Build, and Move classes. Both the round and the tournament is being dealt with through this one class.

4. Move

   This is a basic structure class to hold the Move data structure that has member variables: action, hand card and loose cards.

5. Logic

   This is the brains behind the game. When a user asks for help or the computer needs to calculate its next move, this class provides member functions that check if an input move is valid and also generates the best possible move regardless of who is playing.

6. Build

   One object of this class is declared per round. This object holds the builds in the current game, their owners, their sums and various other member functions that allow creation and maintenance of builds.

How to Run

The program is easy to run in a development environment like Visual Studio. You can just create an empty C++ project, add the header and implementation files and build and run.

If you are on the shell, you can use g++ to compile the code.

Running the program is easy. Go to main, the entry point for C++ programs, and declare an object of 'Game' class. Then call the method play() on it.

```
int main(int argc, char** argv) {
    Game new_game = Game();
    new_game.play();

    return 0;
}
```

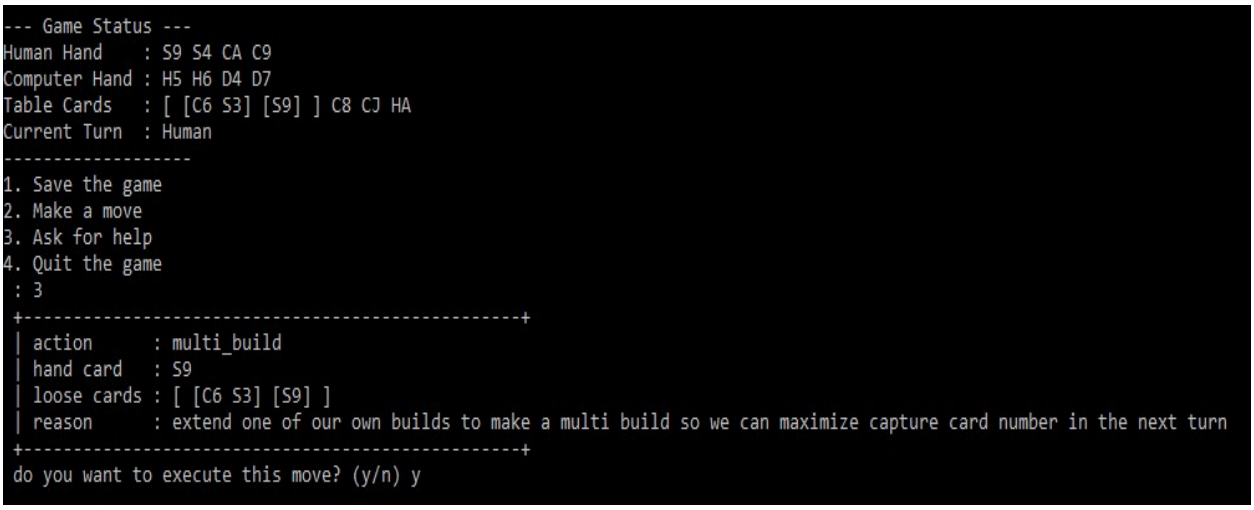
Screen Shots

File input ::

```
--- WELCOME TO CASINO!!! ---

would you like to resume a saved game? (y/n) y
 enter file name with path: ./load.txt
 game loaded from ./load.txt
```

Computer Help ::

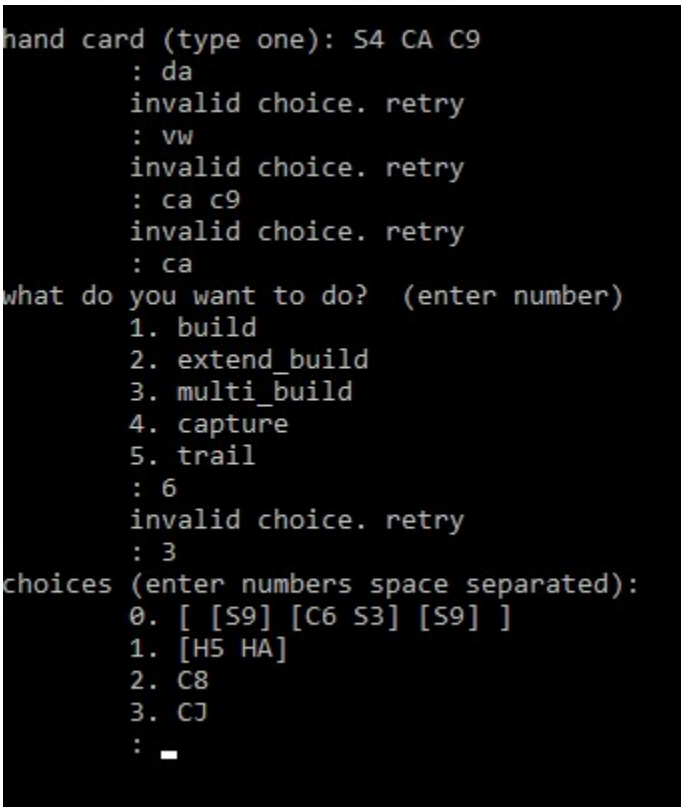


Move Input ::

```
--- Game Status ---
Human Hand    : S4 CA C9
Computer Hand : H6 D4 D7
Table Cards   : [ [S9] [C6 S3] [S9] ] [H5 HA] C8 CJ
Current Turn  : Human
------------------
1. Save the game
2. Make a move
3. Ask for help
4. Quit the game
 : 2
hand card (type one): S4 CA C9
        : s4
what do you want to do?  (enter number)
        1. build
        2. extend_build
        3. multi_build
        4. capture
        5. trail
        : 4
choices (enter numbers space separated):
        0. [ [S9] [C6 S3] [S9] ]
        1. [H5 HA]
        2. C8
        3. CJ
        : 0 1 2
        invalid move. retry

hand card (type one): S4 CA C9
        :
```

Move Input Validation ::


hand card (type one): S4 CA C9
: da
invalid choice. retry
: vw
invalid choice. retry
: ca c9
invalid choice. retry
: ca
what do you want to do? (enter number)
1. build
2. extend_build
3. multi_build
4. capture
5. trail
: 6
invalid choice. retry
: 3
choices (enter numbers space separated):
0. [[S9] [C6 S3] [S9]]
1. [H5 HA]
2. C8
3. CJ
: