Computer Programming 3

Final project
Memory Console Game

December 30th, 2019

Author: Karol Latos

Abstract

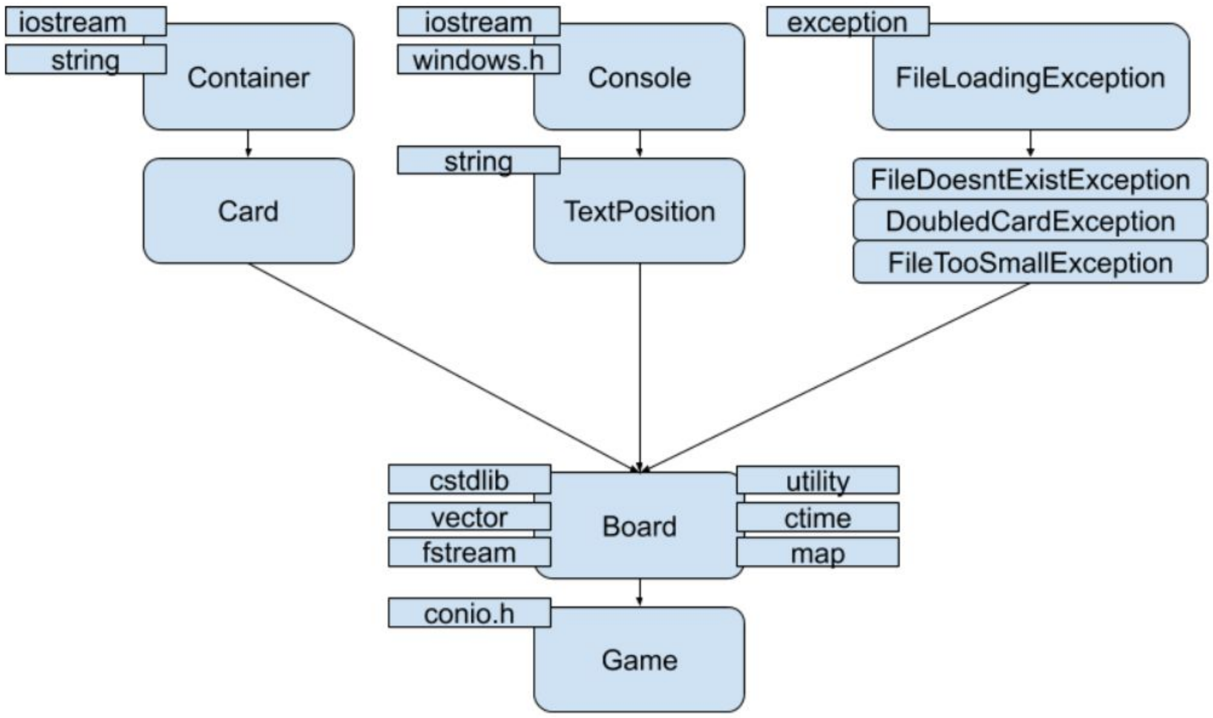
Computer Programming in the 3rd semester is a final part of C/C++ programming course on Macrofaculty. The project summarizing all knowledge gained should be a concise encapsulation of good programming practices, extensive language use, utilization of common algorithms and optimized code. One proposal to epitomize these factors as well as I'm able to is a console game based on the player's ability to memorize cards. Three gameplay difficulties are available: easy, medium and hard, each presenting a set of covered words on board of sizes 4x4, 6x6 and 8x8, respectively. The player's goal is to uncover pairs of words, so that the uncovered words are the same. When the words are matched they stay uncovered, while those which differ return to being covered after being displayed for 2.5 seconds. The game is over when all cards are uncovered, i.e. the player successfully connects all pairs of cards with the same words on them.

Code architecture

The program composes of 10 classes, with 3 composition branches. *Composition* per se is not a convention in the code, but is predominant due to the rule '*Composition over inheritance*'. Then the diagram below describes not inheritance, nor the composition of classes, but rather their dependencies. Each of the branches pertains to different utility of the language and serves a different purpose, and all of the classes can be grouped as such.

1. Displaying a single card - *Container* and *Card*
   a. ***Container***: sets the size of the output buffer for one card and centers the content within it.
   b. ***Card***: stores all the information about the card - whether it's covered or not, if it's out of the game or not and what's the word on this card, together with methods for changing the card state and outputting it to the console. *Card* inherits from *Container*.

2. Modifying the console and text attributes - *Console* and *TextPosition*
    a. ***Console****:* acts as a wrapper for C++ console-oriented functions, so that large portions of messy code between the files can be hidden within the class body. It consists of, among others, methods to set the text color, set or shift the cursor position and disable resizing of the console window.
    b. ***TextPosition***: uses part of the data from *Console* class to output either centered or right-aligned text.
3. File-related exceptions
    a. ***FileLoadingException***: general exception class for errors emerging while trying to open and read cards from the file.
    b. ***FileDoesntExistException***, ***DoubledCardException***, ***FileTooSmallException***



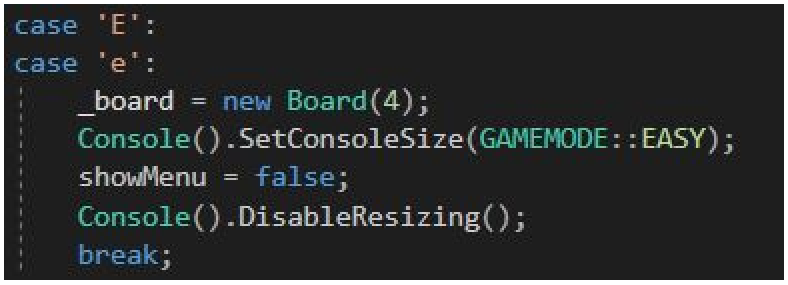
Composed of these three branches are the main operating classes of the code, *Board* and *Game.*

Board: manipulates and stores the current state of the game; 2-dimensional vector of cards, game runtime, cards loading, drawing of the board and processes of revealing the cards, getting card address and removing matched cards from game.

Game: displays the main menu, with a possibility to either quit or select one of three difficulty levels and then initialize the board with appropriate size.

Highlighted code sections

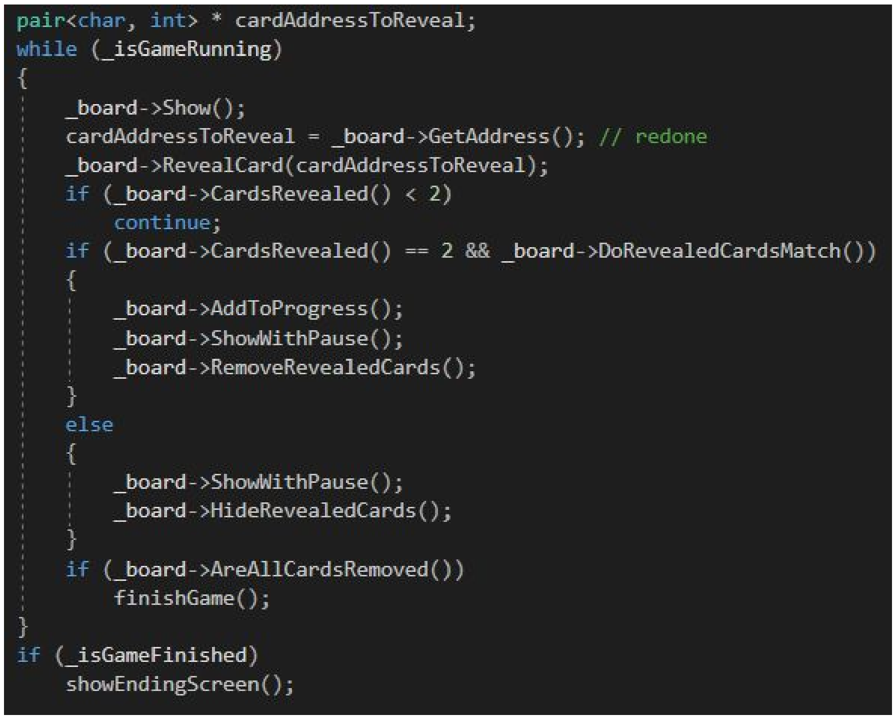
Some portions of the code were selected to present how different components of the program work together.



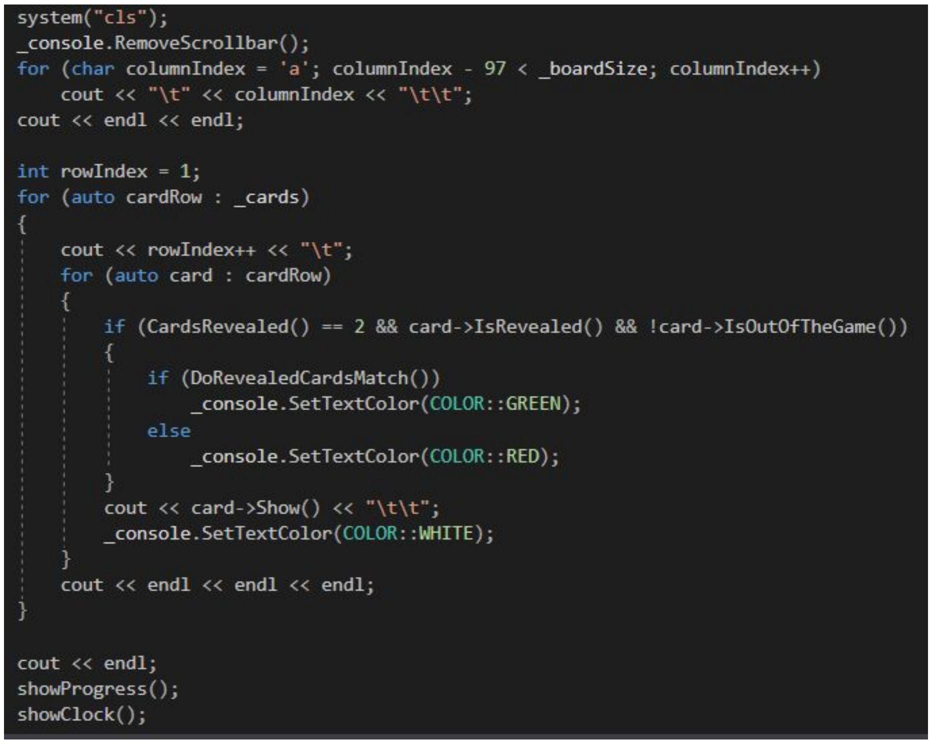
```
case 'E':
case 'e':
    _board = new Board(4);
    Console().SetConsoleSize(GAMEMODE::EASY);
    showMenu = false;
    Console().DisableResizing();
    break;
```

The case of creating 4x4 game board for easy gamemode.

Main loop in *Game::Run()* method, utilizing *Board* public methods to control the flow of the game.



```
pair<char, int> * cardAddressToReveal;
while (_isGameRunning)
{
    _board->Show();
    cardAddressToReveal = _board->GetAddress(); // redone
    _board->RevealCard(cardAddressToReveal);
    if (_board->CardsRevealed() < 2)
        continue;
    if (_board->CardsRevealed() == 2 && _board->DoRevealedCardsMatch())
    {
        _board->AddToProgress();
        _board->ShowWithPause();
        _board->RemoveRevealedCards();
    }
    else
    {
        _board->ShowWithPause();
        _board->HideRevealedCards();
    }
    if (_board->AreAllCardsRemoved())
        finishGame();
}
if (_isGameFinished)
    showEndingScreen();
```



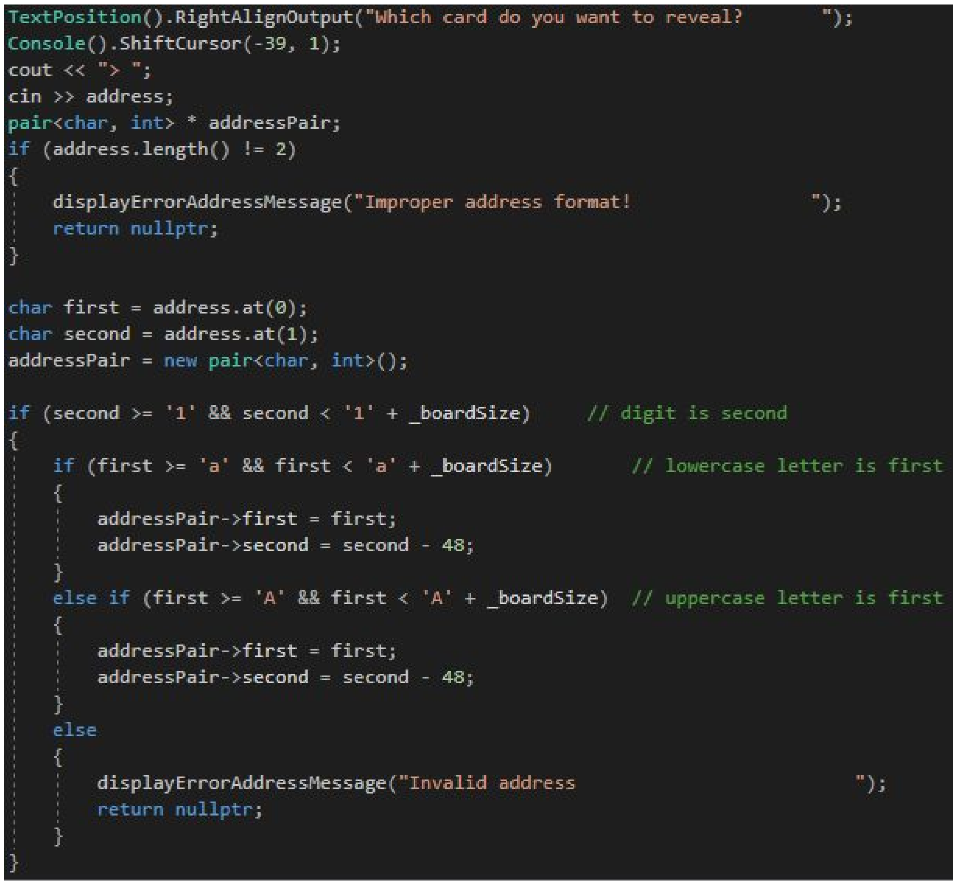
```
system("cls");
_console.RemoveScrollbar();
for (char columnIndex = 'a'; columnIndex - 97 < _boardSize; columnIndex++)
    cout << "\t" << columnIndex << "\t\t";
cout << endl << endl;

int rowIndex = 1;
for (auto cardRow : _cards)
{
    cout << rowIndex++ << "\t";
    for (auto card : cardRow)
    {
        if (CardsRevealed() == 2 && card->IsRevealed() && !card->IsOutOfTheGame())
        {
            if (DoRevealedCardsMatch())
                _console.SetTextColor(COLOR::GREEN);
            else
                _console.SetTextColor(COLOR::RED);
        }
        cout << card->Show() << "\t\t";
        _console.SetTextColor(COLOR::WHITE);
    }
    cout << endl << endl << endl;
}

cout << endl;
showProgress();
showClock();
```

Drawing the board - with address labels and colored cards. The scrollbar has to be removed on each redrawing of the output buffer.

Getting the address of a card to reveal from the player; with a check for proper address format and decoding the input into address *std::pair*.



```
TextPosition().RightAlignOutput("Which card do you want to reveal?       ");
Console().ShiftCursor(-39, 1);
cout << "> ";
cin >> address;
pair<char, int> * addressPair;
if (address.length() != 2)
{
    displayErrorAddressMessage("Improper address format!                ");
    return nullptr;
}

char first = address.at(0);
char second = address.at(1);
addressPair = new pair<char, int>();

if (second >= '1' && second < '1' + _boardSize)     // digit is second
{
    if (first >= 'a' && first < 'a' + _boardSize)       // lowercase letter is first
    {
        addressPair->first = first;
        addressPair->second = second - 48;
    }
    else if (first >= 'A' && first < 'A' + _boardSize)  // uppercase letter is first
    {
        addressPair->first = first;
        addressPair->second = second - 48;
    }
    else
    {
        displayErrorAddressMessage("Invalid address                         ");
        return nullptr;
    }
}
```

Conclusions

The code written for the final project consists of inheritance, polymorphism, exception handling, operations on the output stream and the console, casting and operations on files. It represents the experience gained during the Computer Programming course and outside of the classes. Project files are available on Github: github.com/rol-x/MemoryConsoleGame.