CPT 323 - Object-Oriented Programming Using C++

Assignment 2 - Software Design and Implementation

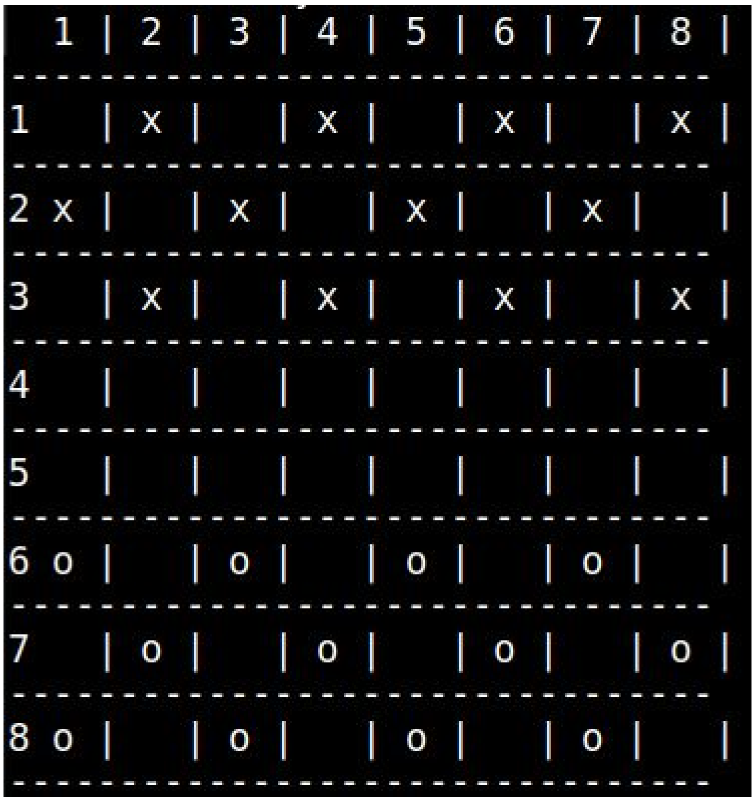
Due Date: 15 Oct, 2017, 11:59pm

In your first assignment, you focused on data structures and performance. In this second assignment we will focus on software design and implementation.

The task in this case is to implement a basic English draughts game. **You may do your assignment by yourself or in a pair.**

Description of the game:

Create a game prototype using modern C++. This is a two player version of the board game, “English draughts”. . The game will start with initialisation of the board, laying out the tokens for the players. The tokens are laid out on the board to begin with as follows:



```
  1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
--------------------------------
1   | x |   | x |   | x |   | x |
--------------------------------
2 x |   | x |   | x |   | x |   |
--------------------------------
3   | x |   | x |   | x |   | x |
--------------------------------
4   |   |   |   |   |   |   |   |
--------------------------------
5   |   |   |   |   |   |   |   |
--------------------------------
6 o |   | o |   | o |   | o |   |
--------------------------------
7   | o |   | o |   | o |   | o |
--------------------------------
8 o |   | o |   | o |   | o |   |
--------------------------------
```

That is, each player has 12 tokens placed in the locations specified. The checker's location may be referred to by two digits in a row – firstly the column and then the row. So "1,1" would refer to the top left cell and "8,1" to the top right, "1,8" to the bottom left, and so on.

You may select any characters you wish to represent the tokens but it makes sense to use an upper and lower case pair to represent normal and king pieces.

Before the game begins, each player can be added to a roster of players available. At the beginning of the game, a player will be selected at random (either human or computer) to take the first move and the players will take turns with their moves until one player can no longer make a legal move.

Pieces can only move in a diagonal direction and normal pieces can only move towards the opposite side from where they started. So, normal "white" pieces can only move up the board and normal "red" pieces can only move down the board - you may choose to distinguish between the pieces in other ways but colour is common (you are not required to create coloured tokens, but you may if you wish). A piece becomes a king by reaching the opposite end of the board from where it started. Once it becomes a king, a piece can move in either direction.

There are two valid moves that can be made – a normal move where a piece can move one place at a time and an attack which is where a piece jumps over an adjacent enemy which results in the jumped piece being taken. You can only jump one piece at a time but once you have made a jump if there is another piece that can also be jumped you can do that immediately – you don't have to wait until your next turn to do so. Any pieces that are jumped are removed from the game. (You might find inheritance handy to represent this relationship).

The rules of the game as you will program it are according to the rules of "English Draughts" at the following url: http://www.mastersgames.com/rules/draughts-rules.htm . Any alternate rules you may have come across do not apply.

Startup Code

Below we will outline the classes we have provided to assist you in your solution to this assignment. **You can use the data structures provided in the sample program. You should feel free to modify this startup code as you see fit.** All design decisions you make should be justified in your report. Please note that we expect that justifications should reference broader reading on software design (such as quotes from reputable web sources). We have done our best to ensure the startup code is relatively bug free but that is not guaranteed. If you find a bug, please raise it on the discussion board so we can respond and ensure your fellow students are informed.

Provided Structures

We have provided source code for a substantial part of a Model-View-Controller application. In a model-view-controller application we have three key logically separated component that clearly separate the responsibilities for the program and allow for relatively easy switching between different implementation strategies. For example I started this project with the intention of giving you a curses interface to work with (a text based windowing system). That ended up being more complicated than I expected so I have switched back to a text based interface.

For your part of the system to work with mine, all we need to consider are the opaque interfaces for the model, the view and the controller.

In particular, in designing the model, you need to consider the functions required in your code and the functions you are able to call (in general the model is called upon rather than calls functions as it is the manager of the data in the system).

The view (ui) class cornerstone of the displaying of data. When you want the view to change we have a few functions available that you are able to call:

· void main_menu(void); - displays the main menu for the system.
· void add_player(void); - creates the view that allows us to add players to the system (just type in their name).
· void play_game(void); - start a new game - this involves displaying a menu of the available registered players that each user can select from.
· void quit_game(void); - quits the current game and displays the main menu.

The controller does all the event handling. Your part of the system does not need to call this part of the system but the controller may set or retrieve data from the model.

Finally we have the model section of the program. As we have said this is the part you which to implement and this part of the system manages the system state - it manages all the data for the system. The functions you need to implement in this part are:

void start_game(int, int); - starts a new game with the specified player ids
char get_token(int,int); - gets the token at the specified 1 based (not 0 based) location.
void make_move(int id, int startx, int starty, int endx, int endy); - makes a move with the current player id, again in 1 based indexes.
void add_player(const std::string&); - adds a player to the system
int get_player_score(int); - gets the score for the player with the specified player id
int get_current_player(void); gets the id of the current player
std::string get_player_name(int); gets the name of a player with the specified player id
std::map<int, std::string> get_player_list(void) const; gets the list of registered players

int get_winner(); - gets the winner of the game if the game has been won. Return -1 if the game is still in play
int get_width(); gets the width of the board
int get_height(); gets the height of the board

Please note that in the design of this application we have made extensive use of **namespaces**. A namespace is an additional level in the class hierarchy that helps to organise the classes or functions inside it. For example, all classes that are part of the application are in the draughts namespace which has three child namespaces being model, nc_controller and ncview. The names here evolved from the fact that I was going to use curses for the view but that has not ended up happening as it was getting far too complicated.

I apologise in advance for the fact I was not able to comment the code properly. The main documentation of the code is provided here but also I indent to update the documentation. If anything is unclear please ask on the discussion board. You may download the startup code from google drive here:
https://drive.google.com/open?id=0BzM8LtxnJauTRGFtWTlOamNiWjQ or I will also maintain a copy on github available here:
https://github.com/muaddib1971/pucpp/tree/master/assignments/a2 which is also mirrored on titan / jupiter / saturn in ~e70949/shared/pucpp/assignments/a2

Allocation of Marks

This section describes in detail all functionality requirements for assignment 2. A rough indication of the number of marks for each requirement is provided.

1. **Design of the model (10 marks)**

   As part of your implementation you are going to have to decide early on what classes will be part of your data model. You should create a class diagram early on of your initial design. You should specify instance data for each class and any public methods they should have.

2. **Correct implementation of the model to link in with the provided view and controller (20 marks)**

   The class design you have been provided with has a partially implemented mvc pattern. Complete this pattern and only use the mvc mechanisms to communicate between components to reduce coupling.

3. **Implementation of the game rules (10 marks)**

Based on the provided rules url above, you should write one or more modules that relate to the game rules for the game. As this is a substantial part of the implementation effort, we are providing specific marks for this.

4. Correct use of inheritance (10 marks)

Use inheritance wherever you can but in particular design an inheritance hierarchy for items. The same is true for input/output classes.

5. Correct use of operator overloading (5 marks)

All your data classes should have operator<< and operator>> overloaded to simplify the reading and writing of data in ascii/binary format.

6. Correct use of exception handling. (10 marks)

All error handling and validation should be done by the use of the exception handling mechanisms rather than returning bool or enum state values. All exceptions which are thrown must be caught. It is optional whether you implement your own exception classes or use c++'s built in exception classes. All functions implemented in the view are wrapped in a try/catch block so any exceptions thrown in your part of the system will be caught in ours.

7. Use of modern C++ features (10 marks)

For this requirement, you should explore some of the modern c++ features taught in this course. For full marks here there should be no use of new unless you really need it (such as with private constructors) and you should explore some of the features of modern c++ such as unique_ptr, lambdas, constexpr, noexcept, override, etc.

8. Demonstration (10 marks)

A demonstration is required for Assignment #2 to show your progress.

It will occur in your scheduled lab classes. The demonstration will be brief, and you must be ready to demonstrate in a two minute period when it is your turn.

This demonstration will occur in the week beginning 2 Oct 2017. You will need to demonstrate a relatively complete model that is correctly connected up to the view.

9. Make / CMake build files (5 marks)

You must write a Makefile or a cmake file to be used with compiling your program. You must compile each object file separately and then link them together. You must pass the appropriate flags into the appropriate phase of compilation. Please note that the requirement of -Wall -pedantic -std=c++14 or -std=c++1z still holds for this assignment. Please ensure your program compiles and runs on titan/jupiter/saturn as that is where we will test your program.

10. Report (10 marks)

You will need a class diagram of your final program. You will also need to justify any design decisions you have made. This means your report needs to provide sound justifications for your decisions. The design design patterns you have used, or other theoretical bases for your final design should be also described and argued for. This report must be submitted as a pdf to the turnitin submission box provided in blackboard by the due date.

Penalties

Marks will be deducted for the following:

- Incorrect compiler flags - 5 mark penalty

- Compile errors and warnings - a 5 marks deduction for compiler warnings, a 10 mark penalty for small compiler errors that your marker can fix quickly. If we cannot get your program to compile, your program will get a maximum of 40% of the marks available for the assignment.

- Unnecessary use of C functions (-5 marks)

- Use of pointers for memory allocation with new / delete - use unique_ptr / shared_ptr when you need to but it is better to allocate objects on the stack when you can. (-5 marks)

- Fatal runtime errors such as segmentation faults, bus errors, infinite loops, etc. Sections affected will get a maximum of 40%. If there are components we cannot run due to fatal runtime errors then those sections will also get a maximum of 40% of the marks available for that section even if the code is correct. It should be runnable and testable.

- Missing files (affected components get zero marks).

- not filling-in the readme file or not providing a readme file - 2 marks penalty.

- Not identifying yourself in all files submitted - this is your work so take ownership! 2 mark deduction for this.

- Late submission: your mark will be reduced by 10 marks for each day late up to a maximum of 5 days. Submissions later than 5 days will not be accepted.

Extensions of Time

You may apply for an extension of time by emailing the instructor at least one week before the due date but please note that I will only give extensions for extenuating circumstances. Common requests for extensions such as “My hard disk crashed and I lost all my work.” won’t be accepted and we advise you to keep an up to date backup of all relevant source files. Likewise, applications for extensions based on working overtime or having a heavy study load will not be granted.

Assignment Submission

Your assignment will be submitted in two parts - your source code which will be compressed into a zip file, and a pdf of your report. The report will be submitted via a turnitin assignment submission and it is your responsibility to ensure that the report shows a low similarity level (you should see a green square).

When/how do I get my marks?

Assignment marks will be made available within 10 working days of the final submission deadline. An announcement will be made on Blackboard when marks are available. An announcement will also be made with regards to what you need to do if there are any mistakes in your marks.

Help!

Please utilise the following with regards to getting help for your assignments:

- For general assignment questions, please use the Blackboard discussion board. That way all students can see all questions once.

- Please do not post large pieces of code to the Blackboard discussion board for plagiarism reasons. If you need help with your code, send me an email.

- You are generally welcome to bring assignment-related questions to class.

- If you are having problems that may prevent you from completing your assignment on time, please talk to someone as early as possible. If you find any problems with the assignment specifications (such as mistakes or bugs), please post your queries to the Blackboard discussion board.