Team Final Project: Implementation Phase

You have specified a scope of work and designed a program, now is the time to implement it. While the project has you design, then implement (then test), in actuality, this will be a cyclic process. You are **encouraged** to revisit your design. What is working, what is not? How do you need to iterate it? Keep track of these changes, they will be important for the final report.

At a minimum, your program must do the following:

1.) Allow one or two people to play the game you specified. You may implement simple (even random) artificial intelligence for one player of a two-player game if you desire. Remember, each move should not violate the rules of the game!
2.) Allow the user to stop play at any time by saving the game state to disk. You must override the `<<` operator for your game object.
3.) Allow the user to restore a previously saved game from the disk at any time (if during play, the current game may be discarded). You must override the `>>` operator for your game object.

At a minimum, your program must adhere to the following restrictions:

1.) Gameplay must take place through a graphical user interface built on FLTK / our graphics libraries. You must use `Button`s with callbacks.
2.) No function may be more than 24 statements long, including `main()`.
3.) Must run either under X windows or MS Windows (this is more lenient than the homeworks have been).

For extra credit, you may also do the following (not an exhaustive list!):

1.) Use images on your graphics objects to have more complex graphics than simple shapes.
2.) Implement scoring and automatically load and update a score file with names and high scores.
3.) Implement a simple A.I. player (a completely random one will **not** earn you extra credit).
4.) Implement parts of your program as a functional library for some subset of grid-based games (make sure to be clear in your report and your program structure if you do this).
5.) Implement multiple versions of the game, allowing the user to choose which variant to play.
6.) Use sound in your application to indicate state, such as announcing whose turn it is.
7.) Allow networked gameplay on multiple computers.
8.) Allow more than two players (assuming the game rules supports this; Chess, for example, does not).
9.) Almost anything else that goes beyond the project specification; if you have any doubts, discuss with your instructor.

Build the program from your design specification, test it, make sure it is working. As a refresher, the following is the project assignment specification:

“Games are systems of rules, in which players make choices [Salen and Zimmerman 2004]. Game rules, based on game state, restrict the choices a player may make. For example, if we are playing Tic-Tac-Toe and I have selected the middle space as my first move, you are now restricted to the other eight spaces. In this assignment, you will **select a game and implement its rules, with a graphical user interface and the ability to save game state**.

“You will craft a **game played on a grid**. This team project will test your ability to build simple graphical user interfaces, handle erroneous input from the user (prevent them from making illegal moves), and your ability to do file I/O and handle errors.

What to Turn In (the Deliverable)

Produce and turn in your source code for your working application as a .zip file in CSNet; also, turn in a Windows-formatted CD to your TA with all source code.

Also, include instructions on how to run your game, its system requirements, and documentation of your team (team name and team members' names).

Grading

The intent of this assignment is to implement a complex application using object-oriented principles. You will be graded accordingly. **This assignment will count as 30% of your team project grade (effectively, 7.5% of your final average for the class).**

Code compiles and application runs without errors for "reasonable" user input	/ 2
Code is human-readable and easy to understand	/ 4
Instructions included	/ 2
Code follows all requirements regarding number of statements / function, etc. (other requirements below)	/ 4
Able to play a complete game with one or more players (this means only actions allowed by the game rules!)	/ 6
Implementation uses Buttons with callbacks	/ 4
User can save at any time, implemented with <<	/ 4
User can load at any time, implemented with >>	/ 4
(Extra credit)	/ (+8?)
total	**/ 30 (+8?)**

References

Salen, K., and Zimmerman, E. *Rules of Play: Game Design Fundamentals*. MIT Press, Cambridge, MA, USA, 2004.