Concordia University
Department of Computer Science
and Software Engineering

Advanced program design with C++
COMP 345 --- Fall 2015

Individual assignment #2

Deadline:	Friday, November 6th, 2015
Evaluation:	5% of final mark
Late submission:	not accepted
Teams:	this is an individual assignment

Problem statement

This is an individual assignment. It is divided into three distinct parts. Each individual student is expected to select one of these parts as his/her assignment. Each part is about the development of a part of the topic presented as the team project. Even though it is about the development of a part of your team project, each assignment has to be developed independently of the others and is not to be presented as an integrated part of the team project. Each member of your team is free to choose to do any part, and is expected to follow a different design approach than the other team members that have selected the same assignment topic. Note that the following descriptions describe the baseline of the assignment, and are related to the project description (see the course web page for a full description of the team project).

Part 1: Player observer pattern

Use an Observer pattern to enable a *player view* C++ component. The *player view* should display the following characteristics of a player: 1) list of countries owned 2) list of continents owned 3) current number of reinforcements 4) total number of armies owned 5) number of battles won. The player view should be implemented as an observer class and the player as an observable class. When the values stored in the player object are changed, the player view should be automatically notified and updated using the observer pattern notification mechanism.

Part 2: Map observer pattern

Use and Observer pattern to enable a *map view* C++ component. The *map view* should display the following characteristics of a map: 1) all countries should show to what player they belong, to which continent they are associated, and how many armies are on this country 2) adjacency between countries should be displayed. The map view should be implemented as an observer class and the map as an observable class. When the values stored in the map object are changed, the map view should be automatically notified and updated using the observer pattern notification mechanism.

Part 3: Interactive map editor

Develop a user-interactive C++ component that enables the user to create/edit a map. The map should have the specifications and constraints as stated in the project description and the first assignment. Using the map editor, the user should be able to 1) create new countries 2) define the adjacency relationships between countries 2) create continents and assign countries to continents. The editor should allow the user to either create a new map from scratch or load and edit an existing map previously saved as a file (see first assignment). The editor should allow the user to save the edited map to a file. Verification of map validity (as expressed in project description and in the first assignment) should be applied before the file is loaded or saved.

Part 4: Player strategy pattern

As a computer player is taking its turn, you will need to define according to what criteria and situation its actions are taken. Use a Strategy pattern to enable the implementation of various decision-making strategies used by the computer players in the Risk game. There must be at least three different strategies such as: 1) *aggressive*: always attack if the player controls a country that has one adjacent enemy country with less armies than it has; 2) *defensive*: never attack unless the player controls a country for which all adjacent enemy countries have significantly less armies; 3) *random*: randomly choose between attacking or not, then if attacking, choose a random valid target. The goal of this assignment is not to come up with complex AI solutions, but rather to demonstrate that you can use the Strategy pattern to easily select between different simple strategies.

Assignment submission requirements and procedure

You are expected to submit a group of C++ files implementing a solution to one of the problems stated above (Part 1, 2, 3 or 4). Your code must include a *driver* that allows the marker to compile and execute your code on a standard lab computer. The driver should use the pattern to manage game objects and somehow demonstrate that the code conforms to the above-mentioned specifications, as well as following the applicable Risk game rules, and that the patterns are correctly implemented. The use of unit testing such as `cppUnit` is not mandatory but encouraged. Along with your submitted code, you have to explain your analysis and design. Briefly explain the game rules involved in the creation of you assignment, citing external sources for specific game rules. Briefly describe the design you adopted as a solution. The design description can be backed-up, for example, by `doxygen`-generated documentation, regular code comments, or a simple diagram. The focus of this course being the coding aspect of software development, you are discouraged to submit extensive documentation.

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category “programming assignment 2”. Late assignments are not accepted. The file submitted must be a .zip file containing all your code. You are allowed to use any C++ programming environment as long as they are available in the labs. No matter what programming environment you are using, you are responsible to give proper compilation and usage instructions to the marker in a README file to be included in the zip file.

Evaluation Criteria

Analysis:	
Clarity and correctness of statement of game rules involved:	5 pts (indicator 4.1)
Design:	
Compliance of solution with stated problem:	20 pts (indicator 4.4)
Simplicity and appropriateness of the solution:	3 pts (indicator 4.3)
Clarity of design description:	3 pts (indicator 7.3)
Use of tools:	
Rationale for selection of tools/libraries:	2 pts (indicator 5.2)
Proper use of language/libraries:	4 pts (indicator 5.1)
Implementation:	
Code readability: naming conventions, clarity, use of comments:	3 pts (indicator 7.3)
Coding style: `.h` and `.cpp` files, use of comments:	3 pts (indicator 5.1)
Relevance of driver and presented results:	7 pts (indicator 4.4)
Total	**50 pts (indicator 6.4)**