Concordia University
Department of Computer Science
and Software Engineering

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

Individual assignment #1

Deadline:	Friday, October 23, 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, or include the implementation of one another's aspects. 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: Map

Implement a group of C++ classes that implement a map for the game of Risk. The map must be implemented as a connected graph, where each node represents a country. Edges between nodes represent adjacency between countries. Continents must be implemented as connected subgraphs, where each country belongs to one and only one continent. Each country is owned by a player and contain armies.

Part 2: Game driver

Implement a group of C++ classes for the game driver, i.e. a component that controls each of the phases of the game and gives control to each of the players in turn during game play. The phases of play are: 1) startup phase (where the number of players and map are chosen, and the countries are randomly assigned to the players); 2) the main play phase (a round-robin turn-based phase where the players are allowed to reinforce, attack and move). The sub-phases of the main phase are: a) reinforcement phase (where a player is given some armies and places them on some if the countries he owns) b) attack phase (where a player may declare a series of attacks from one of his countries to one of its adjacent countries owned by another player) c) fortification phase (where a player may move some armies from one of his countries to another of his countries). Note that this game driver should only the "empty shell" of the eventual driver used in the project, i.e. it only implements the ordering of the phases and leaves the implementation details of each phase as stubs. The resulting implementation should implement a round-robin loop over the players and allow the players to play each of their phase in their own turn.

Part 3: Battle

Implement a group of C++ classes that implement the mechanics of a country attacking another using dice as described in the project description. The player chooses one of the countries he owns that contains two or more armies, and declares an attack on an adjacent country that is owned by another player. A battle is then simulated by the attacker rolling at most 3 dice (which should not be more than the number of armies contained in the attacking country) and the defender rolling at most 2 dice (which should not be more than the number of armies contained in the defending country). The outcome of the attack is determined by comparing the defender’s best

dice roll with the attacker's best dice roll. If the defender rolls greater or equal to the attacker then the attacker loses an army, otherwise the defender loses an army. If the defender rolled two dice then his other dice roll is compared to the attacker's second best dice roll and a second army is lost by the attacker or defender in the same way. The attacker can choose to continue attacking until either all his armies or all the defending armies have been eliminated. If all the defender's armies are eliminated the attacker captures the territory. The attacking player must then place a number of armies in the conquered country which is greater or equal than the number of dice that was used in the attack that resulted in conquering the country. There must be an "all-in" attack mode that uses the maximum number of dice on both sides and runs automatically until the attack results in either a) the defending country to be conquered and all armies of the conquering country are moved to the conquered country or b) the attacking country runs out of armies and cannot attack anymore.

Part 4: Save/load

Implement a group of C++ classes that reads and loads a map file in the `.map` text file format as found in the "Conquest Maps" resource, available at: http://www.windowsgames.co.uk/conquest_maps.html, for example the classic Risk map can be downloaded at http://www.windowsgames.co.uk/ConquestMaps/World.zip. The map saver/loader must be able to read any such map, and do the reverse operation and save a game map in the `.map` text file format. Note that this assignment does not require that the map be created as a connected graph (even though the project will eventually have that requirement). The saver/loader should store the map into some kind of data structure that enables the following verifications that should be applied as a map is being saved/loaded: 1) the map represents a connected graph, 2) continents are connected subgraphs and 3) each country belongs to one and only one continent.

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 observe the execution of your code on a standard lab computer. The driver should simply create one of the components described above and demonstrate that it behaves as mentioned above. 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 your 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, and if you do, you will not receive additional marks for doing so.

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "programming assignment 1". 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 it is usable 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 used:	2 pts (indicator 5.2)
Proper use of language/tools/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:	3 pts (indicator 5.1)
Relevance of driver and completeness of presented results:	7 pts (indicator 4.4)
Total	**50 pts (indicator 6.4)**