Concordia University
Department of Computer Science
and Software Engineering

Advanced Programming Practices
SOEN 6441 --- Fall 2019

Project Build 1 Grading

Instructions for Incremental Code Build Presentation

You must deliver an operational version demonstrating a subset of the capacity of your system. This is about demonstrating that the code build is effectively aimed at solving specific project problems or completely implementing specific system features. The code build must not be just a "portion of the final project", but rather be something useful with a purpose on its own, that can be demonstrated by its operational usage.

The presentation should be organized as follows:

1. Brief presentation of the goal of the build.
2. Brief presentation of the architectural design of your project.
3. Demonstration of the functional requirements as listed on the following grading sheet.
4. Demonstration of the use of tools as listed on the following grading sheet.

You are graded according to how effectively you can demonstrate that the features are implemented. If you cannot really demonstrate the features through execution, you will have to prove that the features are implemented by explaining how your code implements the features, in which case you will get only partial marks.

During your presentation, you have to demonstrate that you are well-prepared for the presentation, and that you can easily provide clear explanations as questions are asked about the functioning of your code, or your required usage of the tools/techniques.

Before the presentation starts, you have to submit your current project code base to the Electronic Assignment Submission System under “project 1”.

Identification

Team	Evaluator	Signature	Date

Grading

Presentation		**5**
Effectiveness, structure and demonstrated preparation of the presentation		2
Fluid exposition of knowledge of code base/clarity of explanations		3
Functional Requirements: The game must include a command prompt that is available throughout the usage of the game (even if you have a GUI). All the commands should be validated and give proper feedback on their effect or invalidity. Each of the commands should only work in the game phase for which they are designed for. The commands should use the exact same syntax as defined below. Some commands have options preceded by a dash. Either or both options can be used and may be used multiple times in a command. Command parameters are noted in italics.		**30**
Map editor		**12**
User-driven creation of map elements: country, continent, and connectivity between countries. Map editor commands: **`editcontinent -add` *`continentname continentvalue`* `-remove` *`continentname`*** **`editcountry -add` *`countryname continentname`* `-remove` *`countryname`*** **`editneighbor -add` *`countryname neighborcountryname`* `-remove` *`countryname neighborcountryname`*** **`showmap`** (show all continents and countries and their neighbors)		4
Saving a map to a text file exactly as edited (using the "domination" game map format). Map editor command: **`savemap` *`filename`***		3
Loading a map from an existing "domination" map file to edit or create a new map from scratch if the file does not exist. Map editor command: **`editmap` *`filename`***		3
Verification of map correctness. The map should be automatically validated upon loading and before saving (at least 3 types of incorrect maps). The **`validatemap`** command can be triggered anytime during map editing. Map editor command: **`validatemap`**		2
Game Play		**18**
Implementation of a game driver implementing and controlling the game phases according to the Risk rules. Game play command: **`showmap`** (show all countries and continents, armies on each country, ownership, and connectivity)		2
Startup phase		**8**
Game starts by user selection of a user-saved map file. Startup phase command: **`loadmap` *`filename`***		1
Map is loaded as a connected graph, which is rendered effectively to the user to enable efficient play.		3
User creates the players, then all countries are randomly assigned to players. Startup phase commands: **`gameplayer -add` *`playername`* `-remove` *`playername`*** **`populatecountries`**		1
Players are allocated a number of initial armies, depending on the number of players.		1
In round-robin fashion, the players place their given armies one by one on their own countries. Startup phase commands: **`placearmy` *`countryname`*** (by each player until all players have placed all their armies) **`placeall`** (automatically randomly place all remaining unplaced armies for all players)		2
Reinforcement phase		**5**
Calculation of correct number of reinforcement armies according to the Risk rules.		2
Player place all reinforcement armies on the map. Reinforcement phase command: **`reinforce` *`countryname num`*** (until all reinforcements have been placed)		3
Fortification phase		**3**
Implementation of a valid fortification move according to the Risk rules. Fortification phase commands: **`fortify` *`fromcountry tocountry num`*** **`fortify none`** (choose to not do a move)		3

Programming process		15
Architectural design—short document including an architectural design diagram. Short but complete and clear description of the design, which should break down the system into cohesive modules. The architectural design should be reflected in the implementation of well-separated modules and/or folders.		3
Software versioning repository—well-populated history with dozens of commits, distributed evenly among team members, as well as evenly distributed over the time allocated to the build. A tagged version should have been created for build 1.		3
Javadoc API documentation—completed for all files, all classes and all methods.		3
Unit testing framework—at least 10 relevant test cases testing the most important aspects of the code. Must include tests for: (1) map validation – map is a connected graph; (2) continent validation – continent is a connected subgraph; (3) calculation of number of reinforcement armies; (4) reading valid/invalid map files.		3
Coding standards—short and simple document describing coding standard used. Consistent and proper use of code layout, naming conventions and comments, absence of “commented out” code.		3
Total		**50**

Notes