Concordia University

Department of Computer Science and Software Engineering
Advanced program design with C++

COMP 345 --- F2019 (Section N)
Assignment #1

Deadline: Oct. 12, 2019 by 11:55PM

Type: this is a team assignment (4 members max.)

Evaluation: 8% of the final mark

Submission: Electronic Submission through your Moodle course homepage. Create an appropriate zip archive including the entire C++ *source* files and all related deliverables.

Problem statement

This is a team assignment. It is divided into five distinct parts. 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 is to be developed/presented/tested separately. The description of each part portrays what are the features that the part should implement, and what you should demonstrate. Note that the following descriptions describe the baseline of the assignment, and are related to the project description [1]. See the course web page for a full description of the team project, as well as links to the details of the game rules to be implemented.

Design Requirements (for all the five parts)

1. All data members of all classes must be of pointer type.
2. All file names and the content of the files must be according to what is given in the description below.

Part 1: Maps

Implement a group of C++ classes that implement the structure and operation of a map for the Eight-Minute Empire game. The map must be implemented as a connected graph, where each node represents country/region. Edges between nodes represent adjacency between countries/regions. Each region can have any number of adjacent regions. Continents must also be connected subgraphs, where each country belongs to one and only one continent. Each country is owned by a player and can contain a certain number of armies. The map component can be used to represent any map configuration (i.e. not only the "Eight-Minute Empire game" map).
All the classes/functions that you implement for this component must all reside in a single .cpp/.h file duo named Map.cpp/Map.h. You must deliver a file named MapDriver.cpp file that contains a main function that creates a map and demonstrates that the component implements the following verifications: 1) the map is a connected graph of adjacent countries/regions, 2) continents are connected subgraphs and 3) each country belongs to one and only one continent. The driver must provide test cases for various valid/invalid maps.

<u>Requirements part 1:</u>

- Map implemented as a connected graph. Node represents a country. Edges between nodes represent adjacency between countries.

- Continents are connected subgraphs. Each country belongs to one and only one continent.
- Country is owned by a player and contain a number of armies.
- Map class can be used to represent any map configuration.
- Driver creates a map and demonstrates that the map class implements the following verifications: 1) the map is a connected graph, 2) continents are connected subgraphs and 3) each country belongs to one and only one continent. The driver must provide test cases for various valid/invalid maps.

Part 2: Map loader

Implement a group of C++ classes that reads and loads a map file in the .map text file format (generated from the project resources). The map loader must be able to read any of Eight-Minute Empire game such map. The map loader should store the map as a graph data structure (see Part 1). The map loader should be able to read any text file (even ones that do not constitute a valid map). All the classes/functions that you implement for this component must all reside in a single .cpp/.h file duo named MapLoader.cpp/MapLoader.h. You must deliver a file named MapLoaderDriver.cpp file that contains a main function that reads various files and successfully creates a map object for valid map files, and rejects invalid map files of different kinds.

Requirements part 2:

- Map loader can read any Eight-Minute Empire map file.
- Map loader creates a map object as a graph data structure (see Part 1).
- Map loader should be able to read any text file (even invalid ones).
- Driver reads many different map files, creates a graph object for the valid ones and reject the invalid ones.

Part 3: Player

Implement a group of C++ classes that implement an Eight-Minute Empire game player using the following design: A player owns a collection of region/countries (see Part 1). A player owns the cubes, disks, and tokens armies and a card (see Part 4). A player has his own biding facility object (see Part 5). A player must implement the following methods, which are eventually going to get called by the game driver: `PayCoin()`, `PlaceNewArmies()`, `MoveArmies()`, `MoveOverLand()`. `BuildCity()`, and `DestroyArmy()`.

All the classes/functions that you implement for this component must all reside in a single .cpp/.h file duo named Player.cpp/Player.h. You must deliver a file named PlayerDriver.cpp file that creates player objects and demonstrates that the player objects indeed have the above-mentioned features.

Requirements part 3:

- Player owns a collection of regions/countries (see Part 1)
- Player owns a hand game cards (see Part 4)
- Player has his own biding facility object (see Part 5)
- Player must implement `PayCoin()`, `PlaceNewArmies()`, `MoveArmies()`, `MoveOverLand()`. `BuildCity()`, and `DestroyArmy()`.
- Driver creates players and demonstrates that the above features are available.

Part 4 : Cards deck/hand

Implement a group of C++ classes that implement a deck of Eight-Minute Empire game cards. Note that the cards should not be read from the file. The deck object is composed of as many cards of the Eight-minute empire cards (e.g. 42). Each card gives goods and action. The deck must have a `draw()` method that allows a player to draw a card from the cards remaining in the deck and place it in with the cards space. The hand object is a collection of cards that has an `exchange()` method that allows the player to select the card from its position in the row and pay the coin cost listed at the top of the board (see the game rules for details). All the classes/functions that you implement for this component must all reside in a single .cpp/.h file duo named Cards.cpp/Cards.h. You must deliver a file named CardsDriver.cpp file that creates a deck of Eight-Minute Empire game cards.

Requirements part 4:

- Deck object is composed of the 42 cards
- Each card has a good and action
- Deck has a `draw()` method that allows a player to draw a card from the cards remaining in the deck and place it card space.
- Hand object is the collection of face-up cards with assigned coin cost.
- Driver creates a deck of cards. Creates a hand object that is filled by face up cards from and that return the each card with its goods and action characteristics.

Part 5: Biding facility

Implement a group of C++ classes that implement a biding facility to be used during start of the game to see who will start first. There is only one bid per game. The biding consists of each player picking up his coins and privately chooses a number to bid. When all players are ready, all players reveal the amount they have chosen to bid at the same time. The player who bids the most coins wins the bid and puts the coins he bid in the supply. Other players do not pay coins if they lost the bid. If the bids are tied for most, the youngest player wins the bid and pays his coins. If all bids are zero, the youngest player wins the bid. . You must deliver a driver that creates the biding facility objects, with the following tests: 1) one can shows a player who bid the most coins wins, 2) that show bids that are tied and the youngest win the bid, 3) one that if all bids are zero the youngest player win.

Requirements part 5:

- Enable the player object to pick up his coins and privately chooses a number to bid.
- Request all players to reveal the amount they have chosen to bid at the same time.
- The player who bids the most coins wins the bid and puts the coins he bid in the supply. If the bids are tied for most or the bids are zero, the youngest player wins the bid and chooses who will first play.
- Driver that test: 1) one can shows a player who bid the most coins wins, 2) that show bids that are tied and the youngest win the bid, 3) one that if all bids are zero the youngest player win.

Assignment submission requirements and procedure

You are expected to submit a group of C++ files implementing a solution to all the problems stated above (Part 1, 2, 3, 4, and 5). Your code must include a *driver* (i.e. a main function) for each part that allows the marker to observe the execution of each part during the lab demonstration. Each driver should simply create the components described above and demonstrate that they behave as mentioned above.

Along with your submitted code, you have to explain your design. provide documentation, and regular code comments, or a simple diagram. You are also responsible to give proper compilation and usage instructions in a README file to be included in the zip file.

Important Note: A demo for about 10 minutes will take place with the marker. The demo times will be determined and announced by the markers, and students must reserve (arrange directly with the markers) a particular time slot for the demo. No demo means a zero mark for your assignment.

You have to submit your assignment before midnight on the due date using Moodle under *A#1_SubmissionBox*. 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 you can demonstrate your assignment in the labs.

Evaluation Criteria

Knowledge/correctness of game rules:	2 pts (indicator 4.1)
Compliance of solution with stated problem (see description above):	12 pts (indicator 4.4)
Modularity/simplicity/clarity of the solution:	2 pts (indicator 4.3)
Proper use of language/tools/libraries:	2 pts (indicator 5.1)
Code readability: naming conventions, clarity of code, use of comments:	2 pts (indicator 7.3)
	Total 20 pts (indicator 6.4)

Reference

[1] "Game rules." Red Raven, game Design: Ryan Laukat Illustration: Ryan Laukat, Alex Davis, John Breckenridge, Malorie Laukat ©2012 Red Raven Games.
https://redravengames.squarespace.com/eightminute-empire