Concordia University

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

COMP 345 --- F2019 (Section N)
Assignment #2

Deadline: Nov. 2, 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 six 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 six 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: Game start

Provide a group of C++ classes that implements a user interaction mechanism to start the game by allowing the player to:

1) Select a map from a list of files as stored in a directory.

2) Select the number of players in the game (2-5 players).

The code should load all the game pieces and use the map loader to load the selected and appropriate map, create all the players, assign biding facility to the players, create a deck of cards, and assign an empty hand of cards to each player.

Requirements part 1: You must deliver a driver that demonstrates:

- The different valid maps can be loaded and their validity is verified (i.e. it is a connected graph, etc.), and invalid maps are rejected without the program crashing.
- The right number of players is created.
- An assigned deck with the right number of cards is created.
- An assigned biding facility to each player.
- An assigned empty hand of cards to each player.

Part 2: Game play: startup phase

Provide a group of C++ classes that implements the startup phase following the rules of the Eight-Minute Empire game. This phase is composed of the following sequence:

1. Shuffling the cards of the deck, and draw six cards, that will be put face up along the top of the board.
2. Each player takes a set of one color of cubes (14 armies) and discs (3 cities). Each player places 3 armies on the starting region on the board. If playing with 2 players, each player takes turns placing one army at a time of a third, non-player color in any region on the board until ten armies have been placed.
3. Player places the coin tokens in a pile next to the board. This is the supply.
   - if playing with five players, each player takes 8 coins
   - if playing with four players, each player takes 9 coins
   - if playing with three players, each player takes 11 coins
   - if playing with two players, each player takes 14 coins.
4. Players bid (using the biding facility implemented in Assignemnt#1). The winner of the bid will be chosen to start to play and the game will be going in clockwise order. If the bids are tied for most, the youngest player wins the bid. If all bids are zero, the youngest player wins the bid. There is only one bid per game.

Requirements part 2: You must deliver a driver that demonstrates:

- The shuffling of the cards from the deck and the draw of six cards.
- The resources assignment to each player, and their placement on the board as well as the supply management.
- Performing the biding and assigning the player who will starts.

Part 3: Game play: main game loop

Provide a group of C++ classes that implements the main game loop following the rules of the Eight-Minute Empire. During the main game loop, starting with the first player and going in clockwise order, players take turns taking one of the face-up cards. When a player takes a card, he must pay the appropriate coin cost for the card depending on where it is in the row (this cost is listed at the top of the board). From left to right, these are the coin costs of the cards: 0, 1, 1, 2, 2, and 3. For example, if a player selected the third card from the left, he would pay one coin.

Each card gives a good and an action. The player takes the action immediately. Actions allow players to build their empires and take control of the board.

Requirements part 3: You must deliver a driver that demonstrates:

- Running of a game loop with each player turn starting with the first and going clockwise order.
- Each player takes one face-up card and plays the appropriate cost for the card depending on where it is in the row.
- A display of each player card's action.

Part 4: Main game loop: The player actions

Provide a group of C++ classes that implements the player actions by incorporating the methods (from part 3 of assignment #1) in the game loop. These are the possible actions:

- `PlaceNewArmies():` Place new armies on the board.
- `MoveArmies():` Move armies.
- `MoveOverLand() and MoveOverWater():` Move over land and/or water.
- `BuildCity():` Build a city.

- `DestroyArmy():`Destroy army.
- `AndOrAction():` "And/Or" actions.
- `Ignore():` Player may take the card and ignore the action.

Requirements part 4: You must deliver a driver that demonstrates all the possible game card's actions listed above.

Part 5: Main game loop: after the action

Provide a group of C++ classes that implement what fellows after a player takes his/her card and action, that is, sliding the remaining cards to the left to fill in the empty card space, then, draw a new card and place it in the right-most space. The play then passes to the next.

Requirements part 5: You must deliver a driver that demonstrates:

- The sliding of the remaining cards to the left to fill in the empty card space.
- The draw of a new card and place it in the right-most space.
- The play passes to the next.

Part 6: Main game loop: Compute the game score

Provide a group of C++ classes that compute the game score. Each player needs to implement `ComputeScore()` method, that counts the victory points for each player that includes (i) the regions he/she won, (ii) continents, (iv) and goods. The game ends when each player owns a certain number of cards depending on the number of players.

2 Players 13 Cards
3 Players 10 Cards
4 Players 8 Cards
5 Players 7 Cards

Requirements part 6: You must deliver a driver that demonstrates:

- The result of the `ComputeScore()` method that counts the victory points for each player.
- Displays the winner

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