Concordia University

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

COMP 345 --- Winter (Section S)
Assignment #1

Deadline: Feb. 19, 2020 by 11:55PM

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

Evaluation: 10% 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 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: GBMaps(Game Board A[1])
Implement a group of C++ classes that implement the structure and operation of a map for the *New Haven* Game Board side A. The map must be implemented as a connected graph, where each node represents a square. Edges between nodes represent adjacency between squares. Each square can be assigned a Tile (e.g. Harvest or Pond). The map can be set to one of the three configurations below depending on the number of players:
- four players, the entire playing area will be used.
- three players, the light center 5x5 area and the top and bottom green squares will be used.
- two players, only the center 5x5 area is used.

All the classes/functions that you implement for this component must all reside in a single .cpp/.h file duo named GBMap.cpp/GBMap.h. You must deliver a file named GBMapDriver.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 squares, 2) set of adjacent

[1] You will receive One bonus point in you implement also Side B of the Game board.

squares belong to one or more players. The driver must provide test cases for various valid/invalid GBMaps[2].

Part 2: VGMaps(Game Village Boards A[3])

Implement a group of C++ classes that implement the structure and operation of a map for the *New Haven* Game Village boards. The Village Game board is a 5 by 6 matrix of circular spaces. The spaces on the Village Game board are marked with numbers (1 to 6) to show the cost to play on that space. The figures (i.e. label) at the side and bottom of each row and column show colonists (and therefore score value) that will be gained by completing a row or column with Buildings. Each player owns one Village Game Board.

All the classes/functions that you implement for this component must all reside in a single .cpp/.h file duo named VGMap.cpp/VGMap.h. You must deliver a file named VGMapDriver.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 numbered circle. The driver must provide test cases for various valid/invalid VGMaps.

Part 3: Maps loader

Implement a group of C++ classes that reads and loads the game map files of either a Game Board or a Village Game Board, in the format you choose (.txt, gif, etc.). The map loader must be able to read any of New Haven game such map. The map loader should store the map as a graph data structure (see Part 1 and 2). The map loader should be able to read the 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 GBMapLoader.cpp/GBMapLoader.h. and VGMapLoader.cpp/VGMapLoader.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.

Part 4: Player

Implement a group of C++ classes that implement the New Haven game player using the following design:

- A player can place on any open square/space on the game board (see Part 1).
- A player owns one Village Game Board (see Part 2).
- A player owns Harvest tiles (see Part 5).
- A player owns Buildings (see Part 5).
- A player owns/ give up Resource markers (see Part 5).
- A player has his own resources Gathering and Building Scoring facilities (see Part 6).
- A player must implement the following methods, which are eventually going to get called by the game driver: `PlaceHarvestTile(),DrawBuilding(),DrawHarvestTile(),ResourceTracker(),BuildVillage()and CalculateResources()`[4].

[2] Valid map is a correct connected graph set for 2, 3 or 4 players, invalid is otherwise.

[3] There are 4 identical Village Game boards named: Stratford, Guilford, Fairfield, and Milford)

[4] The implementation of these two methods is in part 6.

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.

Part 5: Buildings and Harvest deck/hand

Implement a group of C++ classes that implement a deck of New Haven game Harvest Tiles, and buildings. Note these resources should not be read from the file.

The deck of Harvest Tiles object is composed of 60 Harvest tiles, where each tile is divided into 4 squares showing 2 or 3 different Resources. The deck of buildings object is composed of 144 Buildings, with 6 Buildings in each of the four colors numbered from 1 to 6, the opposite side of each building has either a meadow, quarry, forest or wheatfield label.

The deck must have a `draw()` method that allows a player to draw (ref. `DrawBuilding(),DrawHarvestTile(),` from the player). The hand object is a collection of resources that has an `exchange ()` method that allows the player to select the Harvest tile from its position in the row and the column on the Game Board and assign the resource markers a value of the accumulated resources from the Harvest Tiles (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 Resources.cpp/Resources.h. You must deliver a file named ResourcesDriver.cpp file that creates a deck of New Haven game harvest and building resources.

Part 6: Resources Gathered and Village Building Scoring facilities

Implement a group of C++ classes that implement the calculation of:

i. Resources collected for each color shown on the Harvest Tile the player just placed. For each color on the Tile, the value is equal to the squares of that color on the Tile plus the number of squares forming a connected chain of matching color from the Harvest Tile just placed. These chains may only connect orthogonally across edges, never diagonally through corners.

ii. Village building Scoring facility: Scoring is based on how many colonists each player has attracted to his Village. A score points for every row and column that is completely filled with Buildings. The value of each row and column is the number of colonists shown at the right or bottom. However, the value of a row or column is doubled if every Building played on that row or column is face up. Buildings leftover with the player at the end of the game do not score but may be a tie-breaker.

You must deliver a driver that creates the **Resources Gathered and Scoring** facility objects, with the following tests: 1) showing that it can perform different resource gathered from the placement of the harvest tiles on the game board. (see [1] page 4). ii) implement the scoring of colonist for different settings of the Buildings on the Village Game board (see [1] 5, 6 and 7).

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, 5 and 6). 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] New Haven "Game rules."