Concordia University

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

COMP 345 --- Winter 2020 (Section S)
Assignment #2

Deadline: Mar. 21, 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 four 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: <u>Game start</u>

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

1) select the number of players in the game (2-4 players).
2) select the appropriate game map from a list of files as stored in a directory. Use the map loader to load the appropriate game map (e.g. for 2 players the 5X5 game map, for 3 players 5X5 plus the top and bottom green squares game map, etc.).
3) use `ResourceTracker()` method to set the four game resource markers.
4) create the players objects.
5) use the map loader to load a village map and assign one to each player.
6) create a concreate deck of 60 harvest tiles objects (not random).
7) create a concreate deck of 144 buildings tiles objects (not random).
8) use `DrawBuilding()` method to draw 5 buildings tiles to form the initial face up pool of the game.
9) assign an empty hand of cards to each player.
10) each player uses `DrawBuilding()` method to draw six buildings tiles, that stays hidden to the other players and two harvest tiles from each deck. Player's harvest tiles and buildings are private (hidden to the other players).

11) each player uses `DrawHarvestTile()` to draw one harvest tile to be placed face down with his village board as his 'shipment' tile[1].

Requirements part 1: You must deliver a driver that demonstrates the above game set up steps.

Part 2: Game play: Main loop

Provide a group of C++ classes that implements the startup phase following the rules of the New Haven game. This phase is composed of the following sequence:

1. *The player with the smallest ID number will begin*. The play continues with each player taking a turn clockwise until one space is left unfilled on the board. The game ends when there is only one space left on the board. Each player will take an equal number of turns.
2. On each turn the active player begins by selecting one of their harvest tiles and placing it onto the Game Board using `PlaceHarvestTile()` method. This play will generate a supply of resources. The amount of resources collected is indicated by increasing the Resource Markers. The active player will then construct buildings in his village using `BuildVillage()` method with the corresponding resources. When the active player is finished, each other player in clockwise order will have the option of also using any remaining resources to construct buildings in their own village.
3. After everyone has had one chance to build or passes, the active player draws new buildings based on the final position of the Resource Markers. The Resource Markers are then reset to zero value and the player draws a new harvest tile to replace the one just played. The selection of face up Buildings is replenished and the play proceeds with the next player in a clockwise turn.

Requirements part 2: 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 draws his resources (building, harvest), uses the `exchange()` method to select the position on the board to place the harvest tile, then plays a building tile with the appropriate cost of the resources.
- A display of each player's turn and possessions.

Part 3: Game play: Turn Sequence

Provide a group of C++ classes that implements each of the following player sequences.

1. Play a harvest tile

The active player selects either a harvest tile from his possession and place it on any open space on the Game Board. The colors and symbols on the harvest tile do not need to align with adjacent tiles already on the game board. The new harvest tile doesn't even need to be placed adjacent to tiles already on the board. The harvest tile must only be played within the area allowed for the number of players.

The player can also choose to play with his ***SHIPMENT tile, instead of his normal harvest tile. In this case he needs to place SHIPMENT tile face down onto the Board. This tile represents his receiving a supply of resources.***

[1] *Never look at this tile until AFTER it has been played.* This tile represents the opportunity that once per game player may receive 4 of the same resources and temporarily ignore what is actually on the tile.

2. Determine Resources Gathered

Use `CalculateResources()` to calculate the amount of 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.

Updates the `ResourceTracker()` to reflect the total collected of each color. Note: since there are 1, 2 or 3 colors possible on each tile placed, there will always be at least one Resource which remains at zero.

3. Build player Village

Each active player can use any of his building tiles as a potential addition to his village. He can build as many as he likes and can afford. A building tile may only be played if the remaining resources in the matching color equals or exceeds the number on the Village Board space. For every Building played, decrease the corresponding Resource Marker by the number on the board space where the Building was played.

There are a set of restrictions on how player may place Buildings:

- Only one Building may occupy each space.
- The first Building of a particular color that the player places into his Village may be put into any empty space. It does not matter what other colors may be adjacent to that space.
- If the color of the Building the player places is already on his Village Board, then the player must place the new Building adjacent to a Building of the same color. Buildings are only adjacent if they are connected across edges, not diagonally. This rule means that all Buildings of a certain color will be connected as part of a single large chain in the Village.
- Buildings may only be played *face up* if the number on the space matches the number showing on the Building.
- Buildings may be played *face down* onto any number space, regardless of the number on the Building.

4. Share the Wealth

It will often be the case that the player cannot use all of the resources created by placing his harvest tile. Proceeding clockwise, every other player has one chance to erect Buildings using the resources still available.

Proceeding clockwise, each player may use any leftover resources to construct buildings in their village, or pass. If they build, they follow the same rules as listed above under Build Player Village. Each player in turn order only gets one opportunity to erect Buildings in their Village, or pass.

5. Draw Buildings

After all other players have built or passed, the active player draws new Buildings. He must draw one Building for every Resource Marker on the 0 (zero) space of the Resource Track.

The first Building the player takes must be from the set of five Buildings of the game pool. Any other buildings he takes may be from this set of game pool or from the deck. Once the player has finished drawing, he must replace Buildings drawn from the face up, set with new Buildings from the deck.

6. End of Turn

The player needs to rest all four Resource Markers back to zero for the Resource Track, then draw a new Harvest Tile to replace the one he played at the start of the turn. Then the turn passes to the next payer following clockwise order.

Note: If the player is playing his SHIPMENT Tile, he does not get to draw a replacement Harvest Tile. Instead, he has to flip the Shipment Tile over from left to right so it is now face up. It is no longer 4 of anything and becomes only what is actually shown on the Tile.

Requirements part 3: You must deliver a driver that demonstrates the above game turn sequences.

Part 4: Main game loop: Compute the game score

The game ends *when there is exactly one open space left on the Game Board*. Scoring is based on how many colonists each player attracted to his Village.

The player with the highest score is the winner. In the event of a tie, the player with the fewest empty spaces on their board wins. If still tied, then the player with the least buildings leftover wins. If this result is still a tie then it is a shared win between those players.

Requirements part 4: You must deliver a driver that demonstrates:

- The result of the `ComputeScore()` method that counts the number of colonists 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] New Haven "Game rules."