CSI2372
Project
Fall 2013

A Boardgame

The game will be played on a game board and each player will put in turn tiles on the board. The board is a grid in which each row is identified by a letter (e.g. A to I) and each column by a number (e.g. 1 to 12) when a player places a tile at one given position (e.g. C3) then this position becomes occupied and no more tiles can be placed at this location. When a tile is placed adjacent to another one they form a group (there is a minimum of 2 tiles per group). Other players can also place tiles adjacent to this group and the tile becomes a member of the group. We use 4-adjacency (up, down, right, left) to determine the grouping.

1. Because tile placement creates groups dynamically, you must create a map of groups. Each group has a unique symbol (one letter – a `char`), a name (a `std::string`), an age (`int`) which is the turn that this group was created and a group size (number of adjacent tiles) (`int`). You must create getter functions for these attributes. The objects of this class will be placed in a std::map of groups. [10 points]

2. Create a template class Board with a constructor specifying the number of rows and columns. The class should have the following methods:
   - `placeTile(unsigned char, int, T)` // place a tile on the board
   - `T getTileAt(unsigned char, int)` // return the value at location
   - `vector<T> getAdjacent(unsigned char, int)` // up to 4 in the list

   In our case, T will be an unsigned char where the value designates the group and a value of 0 indicates free space. A dummy group instance is created to designate an isolated tile
   - `T Group::getDummy()` [10 points]

3. Create a BoardGame class that contains a Board instance and a `std::map<T,Group*>`. Here are the methods and operators [10 points]:
   - `playAt(Position)` to play a tile at the location (throws an exception if the location results in an illegal move). Position is a helper structure consisting of a char and an int for positions on the board.
     In addition, when the play creates a group, the group is created and added to the map of groups (for now simply create the groups as A, B, C etc...)

If the play increases a group, the group size is incremented. If the tile is placed such that two groups or more are merged, then the larger group absorbs the smaller one. So you would have to delete the smaller group and update the group size of the other. Note that you would also have to update the keys in the board. If there are more than two groups to merge, start with the largest that absorbs the second largest, and then the merged group will absorb the next one etc. until all required groups are merged. In case of equal sizes, the group that was created first (i.e. in alphabetic order) absorbs the newer group.

- `operator<<` which prints the BoardGame to console.

For this simple version, create a loop in main that will read a position at which the player wants to play a tile. The players are taking turns. At the end of each turn, you display the existing group (symbol and size) [10 points].

The first player that creates a group of 41 wins.

Adanced Players : Buying, merging and selling of companies.

This version is based on the above game but creates more stimulating gameplay. Each player (between 3 and 6) is also an investor who seeks to maximize his/her profit by acquiring and selling of companies.

1. Create a subclass of the class `Group` and name the class `Company`. This class needs to have the following methods [10 points]:
   - `mergeRequest()` : returns false if this company has a size greater than 10. As a result companies of sizes 11 or greater will not be able to merge with others. Therefore placing a tile at a position which would cause such a merger is now an illegal move.
   - `getValue()` : returns the value of the company. The value is a function of the size of the group and is calculated as follows :

```
  i. if size <= 6 :
     value= size * $100;
 ii. else if size <= 10 :
     value= $600;
iii. else if size <= 20 :
     value= $700;
 iv. else if size <= 30 :
     value= $800;
  v. else if size <= 40 :
     value= $900;
 vi. otherwise :
     value= $1000;
```

At each turn, the player must place a tile. Each player has to chose between the current 8 randomly generated possibilites.

2. Design a function `generatePositions` which creates a `std::stack` of structure Position. This list contains all the positions on the board(i.e., A1 to I12). Utilisze the function `std::random_shuffle` to mix the tiles in the pile. Once the game gets underway, eight positions are offered to the players. Each time one of the players uses a position, a new position is drawn and removed from the pile. Positions which leads to an illegal play are discarded. [10 points]
3. Design a class Player. Each player has a name and an amount of money (initially $800) and a std::map<unsigned char, int> that holds the number of shares for each existing company by this player. This class must also implement the insertion `operator<<` that prints the player to console. The players are kept in a `std::array<Player>` in the class `BoardGame` For the gameplay we also need to add
   `Company c = BoardGame.getGroup(unsigned char)`
   `player.buy( numberOfShares, Company )`
   `player.sell( Company )` // Sell all shares of a company [10 points].

At a player's turn, the player must place a tile on one of the 8 positions. If this placement causes a merger of companies, then the player receives an amount corresponding to the value of the company which has been absorbed multiplied by the number of shares which the player owns. These shares are then sold and the player no longer owns these shares after the sale and the company does not exist anymnore.

After the player placed a tile, the player can also buy 0, 1, 2 or 3 shares of existing companies. These shares can be bought at the value of the company.

The game ends when the size of a company exceeds 40. All the players sell their shares and the player with the most money wins.

The simplified ***pseudo-code*** of the main loop follows. This code will not run as is, you will need to design the actual implementation.

```
BoardGame boardGame(9,12);
Std::stack<Position> position= generatePositions();
std::array players = createPlayers();
boardGame.addPlayers( players );
int turn= 0;
std::list<Position> availablePositions;
// obtain the initial positions initiales
```

```
for (int i=0; i<8; i++)
   availablePositions.push_back(positions.pop());

do {
  // Print board and current player
  cout << boardGame;

  Player& currentPlayer = players[turn%players.size()];
  cout << currentPlayer;
  // ask player what move to make
  Position p = askPlayer(availablePositions);
  // check if merging occurred
  boardGame.playAt(p);
  // ask player if he want to buy shares
  askPlayerToBuy(boardGame, currentPlayer);
  // add a new position
  do {
     availablePositions.push_back(positions.pop());
  } while (boardGame.isLegal(availablePositions) &&
         availablePositions.size()==8);
  turn++;
} while (!endOfgame());

for (auto player: players)
  cout << player;
```

This game is inspired by the game *Acquire* by Sid Sackson (1962).