// ConsoleApplication1.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Online C++ Compiler.

Code, Compile, Run and Debug C++ program online.

Write your code in this editor and press "Run" button to compile and execute it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <iostream>

#include <iomanip>

#include <iostream>

#include <string>

#include <cstring>

#include <vector>

using namespace std;

//login stub class.

class Login

{

public:

bool signIn(string username, string password)

{

return true;

}

};

//resource stub class. generates a set amount of resources

class Resource

{

public:

string resourceType;

int amount;

Resource()

{

resourceType = "wood";

amount = 4;

}

};

//Bulding stub class. generates 1 type of building for use

class Building

{

public:

string type;

int pointValue;

int cost;

void assignPoints()

{

type = "House";

pointValue = 3;

cost = 2;

}

};

//Slightly implemented gameboard class. in future iterations, should generate a gameboard that has different buildings to build for the player

class GameBoard

{

public:

vector<string> buildings;

vector<int> pointValues;

vector<int> costValues;

void GenerateBuilding()

{

Building newBuildings;

for (int i = 0; i< 15; i++)

{

newBuildings.assignPoints();

buildings.push\_back(newBuildings.type);

costValues.push\_back(newBuildings.cost);

pointValues.push\_back(newBuildings.pointValue);

}

}

};

//Player class. Has implemented finalized versions of most functions.

class Player

{

//example of high cohesion: each player class is specific to one player

// also example of low coupling: Player only connects to resources and login, does not conslut the building class

public:

int resources;

int buildings;

int points;

string loginInfo;

Player()

{

resources = 0;

buildings = 0;

points = 0;

}

bool gatherResource()

{

char answer;

Resource gatherResource;

cout << "You are about to gather " << gatherResource.amount << " resources" << endl;

cout << "Do you wish to continue with this action or cancel? Y or N\n";

while (true) {

cin >> answer;

if (toupper(answer) == 'Y')

{

resources += gatherResource.amount;

cout << "Player has gathered " << resources << " total resources." << endl;

return true;

}

if (toupper(answer) == 'N')

{

return false;

}

}

}

//updates point totals and building numbers for the player

void buildBuilding(int cost, int pointValue)

{

resources -= cost;

points += pointValue;

buildings++;

cout << "Current resources: " << resources << endl << "Current Points: " << points << endl << "Current Buildings: " << buildings << endl;

}

//checks to see if end condition met

bool endGame()

{

if (buildings < 5) { return false; }

return true;

}

//checks to see if it is a valid user. Currently, always returns true

void validate()

{

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

Login newLogin;

bool isValid = newLogin.signIn(username, password);

if (isValid)

{

cout << "User Validated" << endl;

}

else

{

cout << "User not Authenticated" << endl;

}

}

};

//fleshed out game class. Is at the top of the hierarchy.

class Game

{

// instance of Information expert: houses all the information for the system

public:

void tellWinner(int playerNumber)

{

cout << "Player number " << playerNumber << " is the Winner!" << endl;

}

//instance of Controller: handles how the information is changed and manipulated through the system

// the meat of the program. creates instances of players and plays the actual game calling all the classes in proper order

void createGame()

{

//instance of creator: creates the player classes

Player player1;

Player player2;

Player player3;

player1.validate();

player2.validate();

player3.validate();

GameBoard currentGameboard;

currentGameboard.GenerateBuilding();

bool foundWinner = false;

int currentPlayer = 1;

int connected[3] = { 1,2,3 };

char input;

// loop that runs the game

while (!foundWinner)

{

// done using innefficent ways, but was fast and accurate.

if (currentPlayer == 1)

{

if (connected[0] == 1)

{

cout << "Player " << currentPlayer << " choose gather resource (g), build building (b), end game (e), or leave game(q): ";

cin >> input;

if (input == 'g')

{

if (player1.gatherResource())

{

currentPlayer++;

}

}

else if (input == 'b')

{

if (player1.resources < currentGameboard.costValues[0])

{

cout << "no available building with current resources" << endl;

}

else

{

player1.buildBuilding(currentGameboard.costValues[0], currentGameboard.pointValues[0]);

currentPlayer++;

}

}

else if (input == 'e')

{

bool test = player1.endGame();

if (test)

{

foundWinner = true;

}

else

{

cout << "You cannot end the game. " << endl;

}

}

else if (input == 'q')

{

connected[0] = 0;

currentPlayer++;

}

}

else

{

currentPlayer++;

}

}

else if (currentPlayer == 2)

{

if (connected[1] == 2)

{

cout << "Player " << currentPlayer << " choose gather resource (g), build building (b), end game (e), or leave game(q): ";

cin >> input;

if (input == 'g')

{

if (player2.gatherResource())

{

currentPlayer++;

}

}

else if (input == 'b')

{

if (player2.resources < currentGameboard.costValues[0])

{

cout << "no available building with current resources" << endl;

}

else

{

player2.buildBuilding(currentGameboard.costValues[0], currentGameboard.pointValues[0]);

currentPlayer++;

}

}

else if (input == 'e')

{

bool test = player2.endGame();

if (test)

{

foundWinner = true;

}

else

{

cout << "You cannot end the game. " << endl;

}

}

else if (input == 'q')

{

connected[1] = 0;

currentPlayer++;

}

}

else

{

currentPlayer++;

}

}

else if (currentPlayer == 3)

{

if (connected[2] == 3)

{

cout << "Player " << currentPlayer << " choose gather resource (g), build building (b), end game (e), or leave game(q): ";

cin >> input;

if (input == 'g')

{

if (player3.gatherResource())

{

currentPlayer = 1;

}

}

else if (input == 'b')

{

if (player3.resources < currentGameboard.costValues[0])

{

cout << "no available building with current resources" << endl;

}

else

{

player3.buildBuilding(currentGameboard.costValues[0], currentGameboard.pointValues[0]);

currentPlayer = 1;

}

}

else if (input == 'e')

{

bool test = player3.endGame();

if (test)

{

foundWinner = true;

}

else

{

cout << "You cannot end the game. " << endl;

}

}

else if (input == 'q')

{

connected[2] = 0;

}

}

else

{

currentPlayer = 1;

}

}

}

//checks to see who the winner is

if (player1.points> player2.points)

{

if (player1.points> player3.points)

{

tellWinner(1);

}

}

else if (player2.points>player3.points)

{

tellWinner(2);

}

else

{

tellWinner(3);

}

}

};

int main()

{

Game newGame;

newGame.createGame();

}