My final implementation of my program utilised 10 classes

CSquare – Abstract base class for all squares. Contains 2 virtual functions; Land for the action taken for players landing on it which takes in a pointer to a player and vector of pointers to the squares. The other virtual function is PropertyGroupCheck; this is specifically for the property squares, for all other it will simply return true. Has a variable of name and a getter for it.

The following classes of squares were derived from the base class of CSquare:

CPropertySquare – Contains an integer value for the player owner and a pointer owner, cost, rent, and id based colour group. The Land function will firstly check if the square is not owned, if this is the case and the player has a balance greater than zero then the cost will be removed from their money and become the owner. If the square is owned then it will check if the landing play is the owner, if they are not it will call the OwnerGroupCheck to see if it doubles the rent payed/received by the players and do so for the correct values. OwnerGroupCheck goes through the vector of properties calling the PropertyGroupCheck. If a property of that colour type does not belong to the same owner it will return false, else true. PropertyGroupCheck takes player number and colour id and compares them to the square, if colours don’t match it will return true, if colours match and the ids match it will return true, else it will return false indicating that a property of that group does not belong to the same player.

CAirportSquare – Works in the same way as the property square except there isn’t a colour id assigned to it and none of the checks for doubling of rent will occur.

CPenaltySquare – This class contains 2 arrays, one is a string array that contains the type of penalty that the player has encountered and the other array contains the amount that the player needs to pay for this penalty. The Land function will get the next pseudo random number that has been generated in randomSelection and select the corresponding penalty, output that the player has received this penalty and remove the amount from the player’s money.

CBonusSquare – This class works identically to the penalty square except for it stores the different types of bonus’ instead and will add the corresponding amount of money to the players money based of the roll.

CGoSquare – Land function outputs that the player is resting.

CFreeParkingSquare - Land function outputs that the player is resting; this could have been the same class as the Go Square as they function identically.

CJailSquare - Land function outputs that the player is just visiting Jail.

CGotoJailSquare – Land function will move player to jail, output that the play has paid £50 for jail, it will remove £250 due to the nature of passing go adding £200 so that the only change to the players money is -£50.

CMonopoly – Acts as manager class for the game, reads monopoly.txt into vector of pointers to square objects, builds vector of pointers of players based on user’s input, seeds random function using seed.txt. Calls function Play which cycles for the amount of rounds in the game, for this scenario 20 loops. For each player in the game it grabs a number from 1-6 using the randomSelection function and calls movePosition function of the player using this number. It will then get the players position to output name of the square where the player has landed, then get the players current money to output their balance. When the destructor is called it will cycle through the vectors of squares and players deleting each instance of them.

CPlayer – Has a player number, name, money and an iterator for the vector of squares to hold their position. It has a getter for each of these variables. Has a function for adding/removing money from the player. And a function to move the player across the board, this function takes in the number of places the player needs to move and the vector of pointers to squares. It will increment the iterator for the number places to move, if it reaches the end of the vector it will move the iterator back to the beginning and add £200 to the players balance for passing Go. After it has finished moving the player’s position it will call the land function of the square it is on.

randomSelection – A global function to grab the next number that has been generated from the initial random seed that has been generated.

gPound – Global constant value of the £ character.

This implementation is a full scenario, I believe it should receive close if not full marks.