MONOPOLY IN C++

Ashikul Alam

ESE 224 Final Project

Ashikul Alam
108221262

ESE224 Final Project Report

Project Description

Create a simulation of MONOPOLY in C++.

"In the game, there are 4 players. They roll two dices, move their markers, and then buy the cities on which the players stand. Finally, the player who have the most money win. Your final goal of the project is just like this game. In other words, you are going to play/built/program 'Computer-Version MONOPOLY.' Your game will be displayed on Console window as usual."

Architecture and Design

- Player Class
    - This class is used to create an object to store attributes of a game player.

```
 private:
       string name; //Player's Name
       int id;       //Player's Id
       double money; //Player's Money
       int position; //Player's Position

public:
       Player(); //Default Constructor

       //Accessors
       string getName() const { return name;}
       int getId() const { return id;}
       double getMoney() const { return money;}
       int getPosition() const { return position;}

       //Mutators
       void setId(int v);
       void setMoney(double v);
       void setName (string v);
       void setPosition(int v);

       //Additional Methods
       void loseMoney(double m);   //Subtract Money
       void gainMoney(double m);   //Add Money
       void print(ostream& out) const;    //Print Player's Variables
```

- Functions

- loseMoney and gainMoney are used to manipulate Player's money data.
- print is used to display Player's data to console output.

- City Class

- This class is used to create an object to store attributes of a city tile.

```
private:
	int id;		//City Id
	int ownerId;  //City Owner's Id
	double value; //City's Property Value
	double charge;		//City's Property Value/5
	int xcoord;   //City's X position on Board
	int ycoord;   //City's Y position on Board
	string cityname;	//City's Name
	string owner; //City's Owner
	string tile; //City's Board Display


public:
	City(); //Default constructor
	City(int idN, string ncityName, double nValue,
		int x, int y); //Parameterized constructor

	//Accessors
	int getId() const {return id;}
	int getOwnerId() const {return ownerId;}
	double getValue() const {return value;}
	double getCharge() const {return charge;}
	int getXcoord() const {return xcoord;}
	int getYcoord() const {return ycoord;}
	string getCityName() const {return cityname;}
	string getOwner() const {return owner;}
	string getTile() const {return tile;}

	//Mutators
	void setId(int v);
	void setOwnerId(int v);
	void setValue(double v);
	void setCharge (double v);
	void setXcoord ( int v);
	void setYcoord (int v);
	void setCityName (string v);
	void setOwner (string v);
	void setTile (string v);

	//Additional Methods
	void print(ostream& out) const; //Print City's Attributes
```

- Functions
    - print is used to display Player's data to console output.
- Card Class

    - This class is used to create an object to store attributes of a card.

```
public:
	//Constructors
	Card(); //Default
	Card(char aSuit,
		int aRank);//parameterized
	//Accessors
	int getRank() const;
	char getSuit() const;
	//Formatted Display Method
	//Example: if char is 'S' and rank is 11
	//output will be:
	//Jack of Spades
	void displayCard(ostream& outStream) const;
private:
	char suit;
	int rank;
```

- Functions
  - displayCard is used to display Player's data to console output.

- Card Deck Class

  - This class is used to create an object to store attributes of a deck of cards.

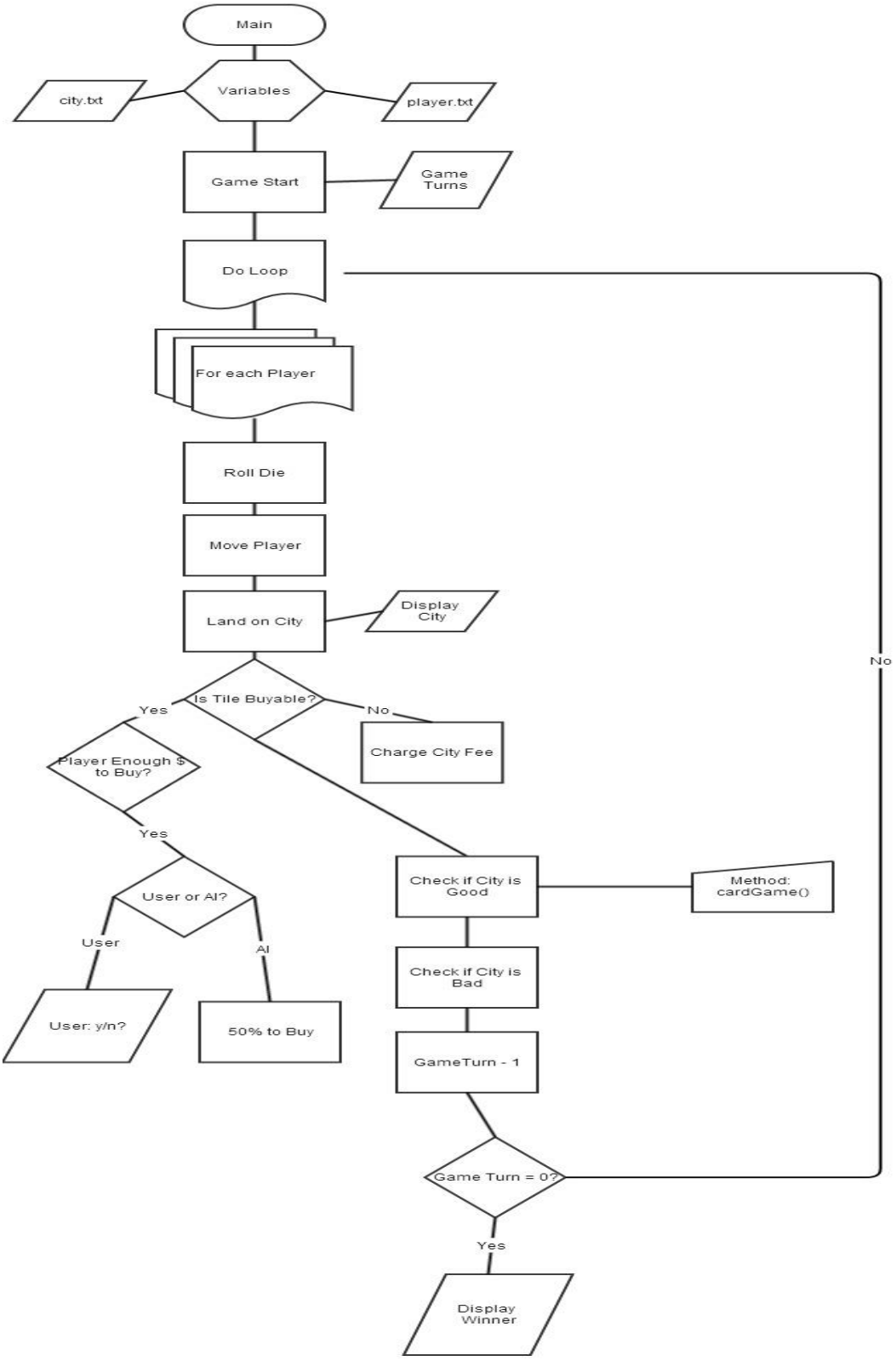
```
public:
	CardDeck();
	void shuffleDeck();
	Card draw();
private:
	vector<Card> theDeck;
	vector<Card> deltCards;
```

- Functions

  - shuffleDeck is used to refresh the deck to 52 cards in random order
  - draw is used to push a card object

- Main Class

  - This class utilizes all the previous classes in simulating a game of Monopoly.


Main
Variables
city.txt
player.txt
Game Start
Game Turns
Do Loop
For each Player
Roll Die
Move Player
Land on City
Display City
Is Tile Buyable?
Yes
No
Charge City Fee
Player Enough $ to Buy?
Yes
User or AI?
User
AI
User: y/n?
50% to Buy
Check if City is Good
Method: cardGame()
Check if City is Bad
GameTurn - 1
Game Turn = 0?
No
Yes
Display Winner

- Important Functions
    - cardGame() utilizes the Card and CardDeck classes to draw 2 cards randomly and determine if they are opposite colors. This is used as the mini-game when a Player lands on a “good” tile.

```
double cardGame (){
	CardDeck theDeck;    //New CardDeck Object
	theDeck = CardDeck(); //Constructor
	Card card1, card2; //2 New Card Objects
	card1 = Card();      //Constructor
	card2 = Card(); //Constructor
	theDeck.shuffleDeck(); //Shuffle Deck
	theDeck.shuffleDeck(); //Again Incase
	card1 = theDeck.draw();     //Draw a card from the Deck
	card2 = theDeck.draw();     //Draw a card from the Deck
	bool isCard1Black(false); //Bool to check, assumed red
	bool isCard2Black(false); //Bool to check, assumed red
	//Check if card1 is black
	if( (card1.getSuit() == 'D' || card1.getSuit() == 'C')){
		isCard1Black = true;
	}
	//Check if card2 is black
	if( (card2.getSuit() == 'D' || card2.getSuit() == 'C')){
		isCard2Black = true;
	}
	cout << "\nCard Mini-Game! Drawing two cards...";
	cout << "\n";
	card1.displayCard(cout);
	cout << "\n";
	card2.displayCard(cout);
	cout << "\n";
	//If card1 and card 2 are opposite colors...
	if (isCard1Black != isCard2Black){
		return 100*(card1.getRank()+card1.getRank());
	}
	return 0;

}
```

Algorithms and Design Considerations

To navigate multiple players and cities quickly, array of objects were used.
Other options were to use nodes, linked lists, or queues. However, arrays were used for ease.
This approach is useful if a variable number of cities or players is desired. To which, easily changing 2 constants makes it easy to implement.

```
	City arrayCity[NUMBER_OF_CITIES+1];
	Player arrayPlayer[NUMBER_OF_PLAYERS+1];
```

To get the needed data for cities and players, <iostream> library was used to locate the txt files and stream in the data to the array of Player and City objects. This approach is useful if the txt files are in the appropriate format and if more rows are added.

```
	fin.open("C:\\city.txt");
```

```
	for (int i = 1; i < NUMBER_OF_CITIES+1; ++i)
	{	fin >> id >> cityName >> value >> x >> y;
	fin2.open("C:\\player.txt");
	for (int i = 1; i < NUMBER_OF_PLAYERS+1; ++i)
	{	fin2 >> id >> name >> money ;
```

To get random values the rand() function was used when rolling a die or determing if the AI will purchase a city 50% of the time.

```
                    //Roll Die
                    roll = rand() % 6 + 1;
//50% Roll Check    int buyChance = rand() % 2 + 1;
```

To navigate the player around the board, modulo was used so that when the player exceeds the board's max position of 16, the player returns back to 0.

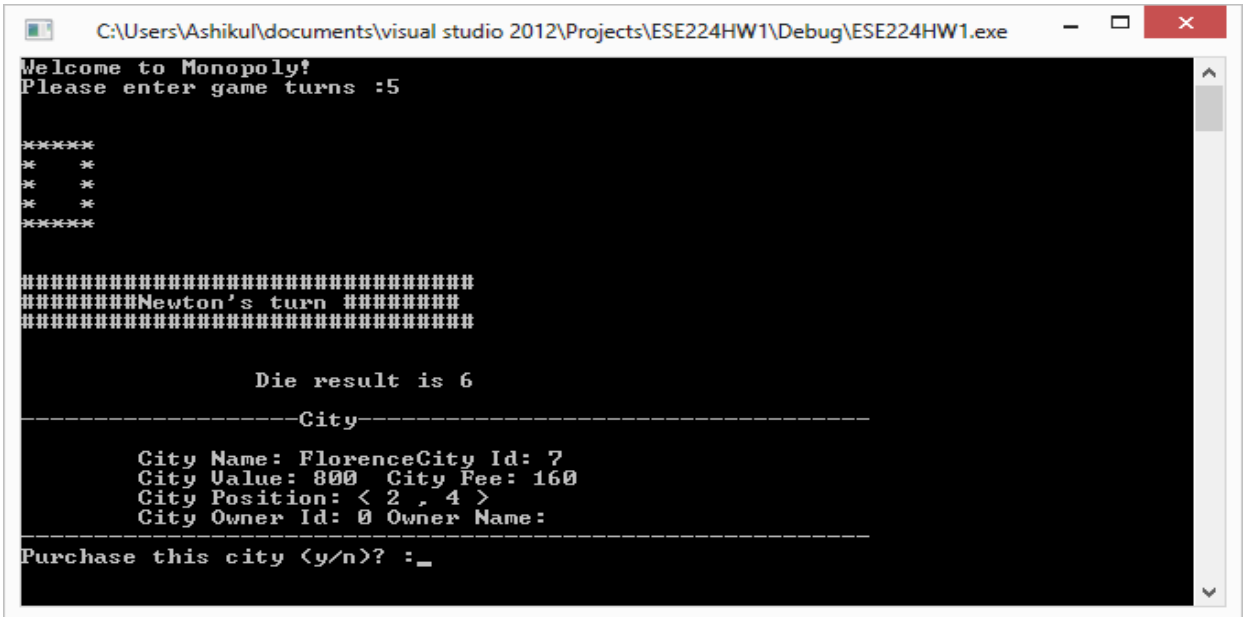
```
                    //Move the Player
                    arrayPlayer[i].setPosition((arrayPlayer[i].getPosition()+roll)%17);
```

To display the board, each of the 16 Cities had a tile string associated with it. If the city was empty, it was filled with "*". Otherwise, if a player was on it, that player's ID would show. Multiple players on one tile was not possible, and is not seen. This design constraints use to only 16 Cities. If more cities are added or if the extra credit was to be implemented, this approach would not work. It is not flexable or modular.

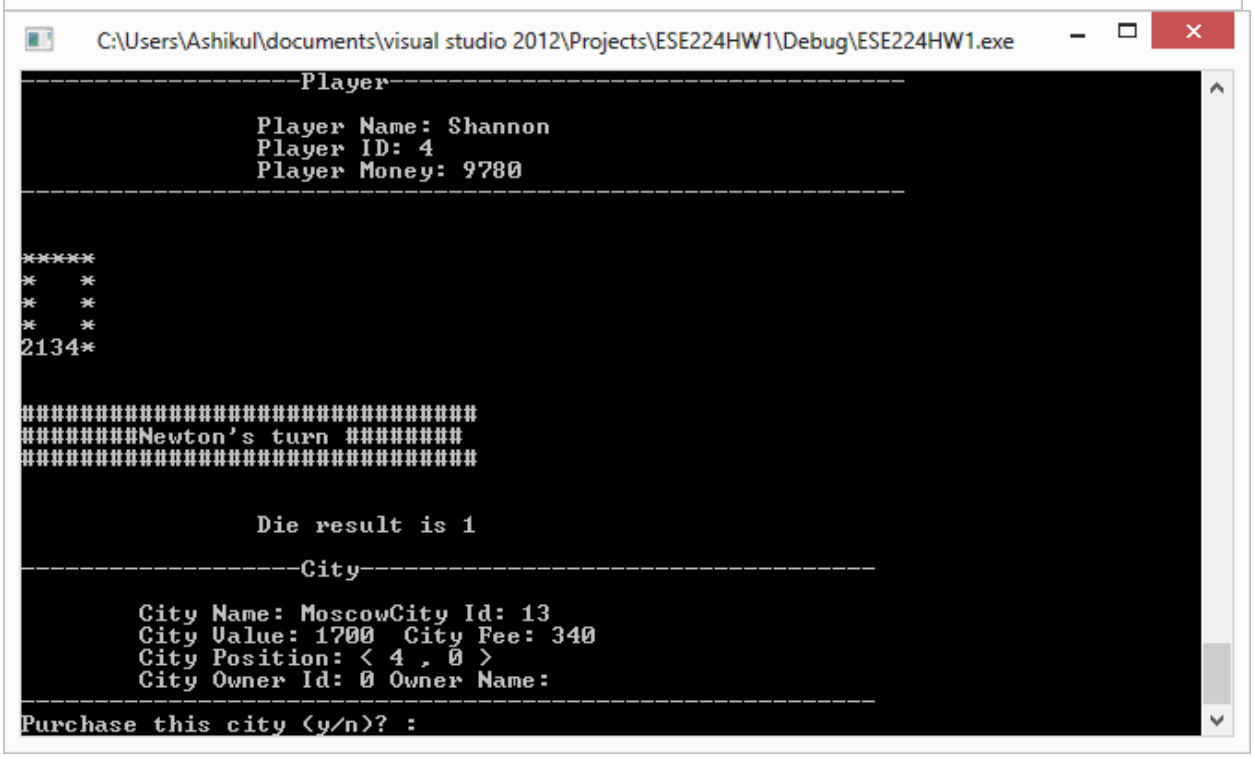
```
cout << "\n" <<endl;
cout << arrayCity[1].getTile() << arrayCity[2].getTile() << arrayCity[3].getTile()
<<arrayCity[4].getTile() <<arrayCity[5].getTile() <<endl;
cout << arrayCity[16].getTile() << "    " << arrayCity[6].getTile() << endl;
cout << arrayCity[15].getTile() << "    " << arrayCity[7].getTile() << endl;
cout << arrayCity[14].getTile() << "    " << arrayCity[8].getTile() << endl;
cout << arrayCity[13].getTile() << arrayCity[12].getTile() << arrayCity[11].getTile()
<<arrayCity[10].getTile() <<arrayCity[9].getTile() <<endl;
cout << "\n" <<endl;
```

A console output limits MONOPOLY in many ways. Player and City stats have to be constantly displayed. The board has to be minimally designed so that the screen does not need constant scrolling. Single colored white text also visually limits MONOPOLY.

Example Outputs:

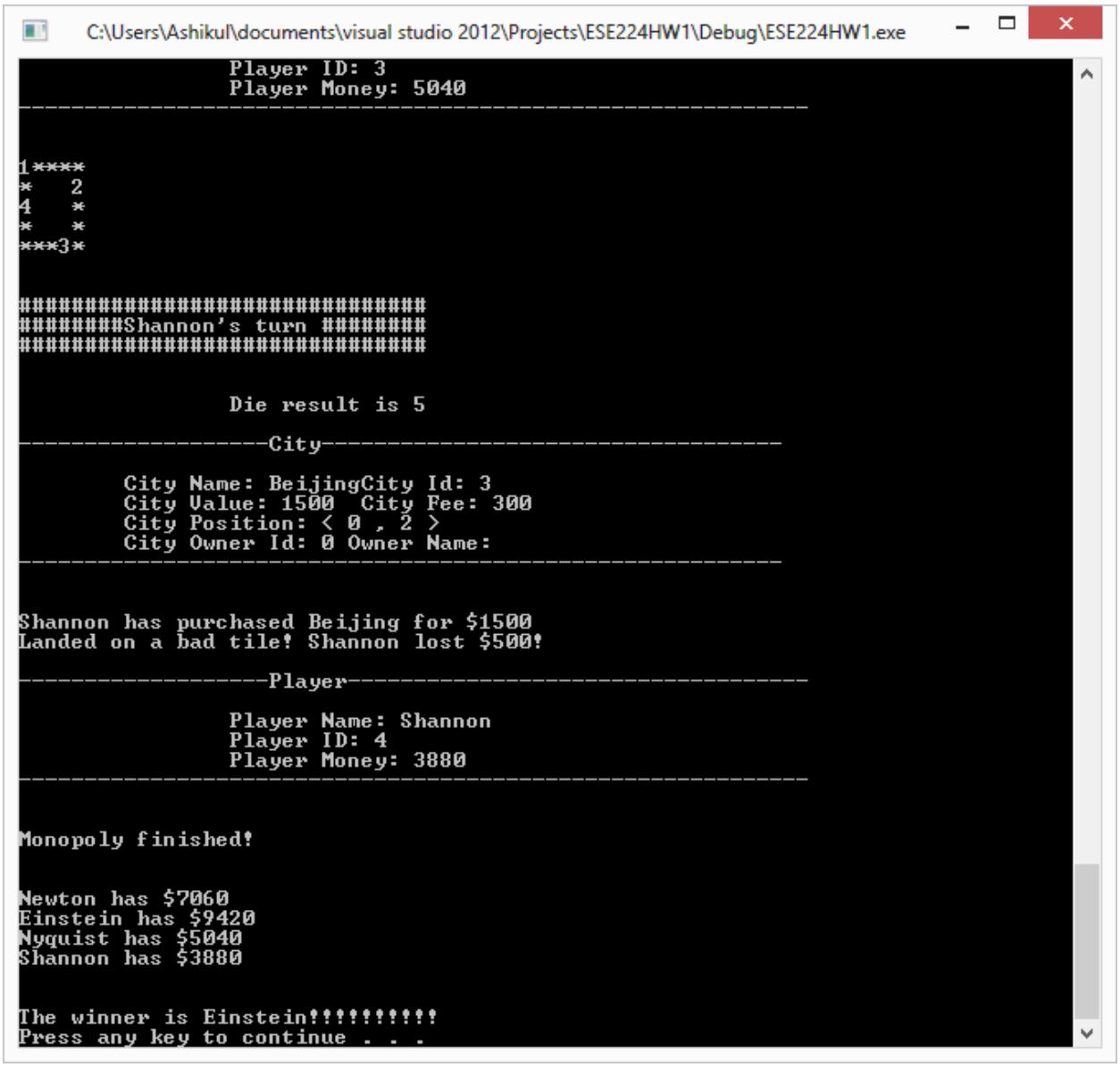

C:\Users\Ashikul\documents\visual studio 2012\Projects\ESE224HW1\Debug\ESE224HW1.exe
Welcome to Monopoly!
Please enter game turns :5
*****
*   *
*   *
*   *
*****
##################################
########Newton's turn ########
##################################
Die result is 6
-------------------City-----------------------------------
City Name: FlorenceCity Id: 7
City Value: 800 City Fee: 160
City Position: < 2 , 4 >
City Owner Id: 0 Owner Name:

Purchase this city (y/n)? :


C:\Users\Ashikul\documents\visual studio 2012\Projects\ESE224HW1\Debug\ESE224HW1.exe
-------------------Player---------------------------------
Player Name: Shannon
Player ID: 4
Player Money: 9780

*****
*   *
*   *
*   *
2134*
##################################
########Newton's turn ########
##################################
Die result is 1
-------------------City-----------------------------------
City Name: MoscowCity Id: 13
City Value: 1700 City Fee: 340
City Position: < 4 , 0 >
City Owner Id: 0 Owner Name:

Purchase this city (y/n)? :



Contribution
Ashikul Alam – Independent Project