Monopoly

Steve Fox

CSC-5

Project 2

40715

Introduction

Monopoly is game that many of us played as we were growing up. Who could generate the largest banks and acquire the most property while the other players are reduced to nothing. Coding this game provided many inquiries into the c++ language do to the logic necessary for the program to function properly. This version is a two player game of monopoly that utilizes a random number generator to simulate actual dice roles. These dice rolls accumulate and dictate what function is called that corresponds to the current dice roll. If the total dice roll goes over 39 then it is set to zero and $200 is passed into the players bank to simulate passing go. In this version, every aspect works except I did not want to enable mortgaging in this version. Instead if a player’s bank goes below zero then the player will have to sell property and/or houses and hotels.

The users are presented with a menu choice for what action they would like to execute in a given turn. One can roll the dice, view properties purchased, sell properties, sell houses and hotels, buy houses, and view a list of the properties and times they occur in the monopoly game. So, how does this game executes these actions? Each player is designated a two – dimensional character array that is at first initialized with ‘?’ character indicating that a property and the houses and hotels associated have not been purchased. When a player lands on a given property and decides to purchase the property then a ‘b’ is placed in the array that correlates with that given properties position. The same is said for purchasing houses and hotels if the given condition of owning the required properties is present. All the costs of properties and their actual rents are congruent with the Monopoly board game. Now, if a player wishes to sell any of these elements then the ‘b’ is replaced with the ‘?’ indicating that the player no longer owns this property. A player can sell property for whatever they want as long as the other player agrees.

Furthermore, When a player sells a property the appropriate conversions are made. A player’s bank who is selling the property has the sum requested deposited into their bank and subtracted from the purchasers bank. Also, the purchasing player receives a ‘b’ in their array indicating that they now own this property. The fact that parallel arrays are utilized is a great benefit due to the fact if a ‘b’ is discovered in a players array indicating ownership then the name of the property will appear in the property purchase list of that player. This enables the player to be able to keep track of properties and figure out which ones are needed in order to buy houses which may greatly increase their revenue. However, a player can deny any offer if he or she does not agree with the terms.

Many of the properties utilize the same logic in deducing whether or not a player owns a property and whether or not it has houses or hotels on it and the rate for each of these aspects. If a player lands on a property the following algorithm is initiated:

1.If not purchased then the player is prompted if they would like to buy it

2.If the property is purchased the program then sees if the present player owns

the property or if it’s the opponent.

3. If the opponent owns the property then a six step process is initiated.

a. The program sees if there are six ‘b’s in a given row indicating there is a

hotel

b. The program sees if there are five ‘b’s in a given row indicating there are

four houses on the property.

c. The program sees if there are four ‘b’s in a given row indicating there are

three houses on the property.

d. The program sees if there are three ‘b’s in a given row indicating there

are two houses on the property.

e. The program sees if there are two ‘b’s in a given row indicating there is

one house on the property.

f. The program sees if there are one ‘b’ in a given row in the first column

indicating there are no houses on the property but it has been purchased

4. Then the player who landed on that property is charge the appropriate amount depending on the level of development of the property if an opponent owns.

Now this process happens on most of the properties. However, there are instances when this algorithm is modified. Railroads rent, for example, is dictated by how many other railroads the owner has. Here, the program looks for ‘b’s in the first column that correlates with the position of railroads in the prop and rent arrays. Furthermore, Utilities function on a whole different level. A pointer called totDie is utilized to send a given dice roll to the utility to calculate the appropriate rent. If the owner only owns one utility then the dice roll is multiplied by four. However, if the owner owns two utilities then the rent for the other player is ten times the dice roll. All these properties mimic the exact actions of the board game itself.

Conversely, there are other positions on the board that cannot be purchased yet still execute an action. Start and free parking were easy because the only action that is taken is that the program outputs that the player has landed there. Chance and Community Chest on the other hand was extremely innovative. Again, a random number generator is utilize to figure out what type of action will be executed. Each of these actions is based on the actual cards. Each action is assigned a number and then a random number is generated then that action is executed. Now, this brings a deviation from the game. Usually, when Chance or Community Chest is drawn in the actual board game that card is placed at the bottom of the stack. In this game, a card can be “drawn” in any order and I think gives the game more of an edge. Actions such as repairs on houses and hotels and the fact that it may be drawn numerous times makes Chance and Community Chest very dangerous or very advantageous. Only randomness will tell.

Furthermore, Jail and Go to Jail are other positions that function with their own unique actions. Go to Jail is very straight forward. If a player lands there, their dice roll is reduced to 10 and a ‘j’ is placed in the tenth row, first column of the character array of that player indicating that the player is in jail. Now when in jail, three actions can be executed. The player can spend $50 to get out of jail and wait till their next turn to roll. On the other hand, a player can attempt to roll doubles which will also get them out of jail. A player has three chances to roll doubles or they are charged $50 and kicked out of jail. There are Chance and Community Chest actions that function like Go to Jail so there is a great chance that a player may experience what it is like to enter monopolies jail .

The game continues until a player’s bank is reduced to negative numbers. Instead of losing right then and there, the player has the chance to sell property and/or house and hotels. If they are unable to do so. The game is over and the player whose bank is positive wins the game. Just like the real game, the time it may take to complete a game may be extensive. Good luck, and may fortune be on your side.

Variables

|  |  |  |
| --- | --- | --- |
| const int row=40 | 73 | Rows for Arrays |
| string prop[row] | 74 | Holds property names |
| char playr1[row][COL] | 75 | Holds properties purchased by player 1 |
| char playr2[row][COL]; | 76 | Holds properties purchased by player 2 |
| int bank1=2000; | 77 | Player one’s bank totals |
| int bank2=2000; | 78 | Player one’s bank totals |
| int player1=0; | 79 | Player one’s dice rolls |
| int player2=0; | 80 | Player two’s dice rolls |
| int rent[row][COL]; | 81 | Rents for various properties |
| int count=0; | 82 | counter |
| char ch; | 83 | Waits for player to press any key |
| char choice; | 84 | Holds players choice to make another menu selection |
| char menuCh; | 85 | Holds players menu selection |
| int times=0; | 86 | Holds number off dice rolls in a given roll |
| totDie | 134 | Holds total dice roll |
| Bool turn | 135 | False for player 1’s turn. True for player two’s turn. |
| Int dbls | 136 | Accounts for doubles being rolled |

Required Structures

(Note there are numerous times when these structures are utilized so this will be an example of each)

|  |  |  |
| --- | --- | --- |
| Structure | Line | Purpose |
| For loop | 108 | Writes contents to file |
| For loop | 117 | Initialize Arrays with ? |
| While loop | 193 | Input validation for menu |
| Do while loop | 143 | Players 1 turn and loops if doubles is rolled |
| Switch Statement | 165 | Players menu for various tasks |
| If/else if/else | 262 | Player ones various function calls depending on dice roll |
| Nested if statements | 4694 | Menu and check to see if player 1 owns required properties to purchase houses on purple properties |
| Array | 74 | Holds properties names imported from files |
| 2D Array | 75 | Holds the properties that player1 owns |
| Files | 99&107 | Holds property names and  rents |
| Sorted Array | 8701 | Puts list of properties into alphabetical order |
| Default parameter | 65 | In Function void hsrep( ); |
| Reference variable | 25 | Bank 1 and 2 passed to purp1() function by reference. |
| Pointers | 25 | totdie and dbls passed to  rollDie() function |

Functions

|  |  |  |
| --- | --- | --- |
| Functions | Line | Purpose |
| int rollDie(string [], char [][7],int&,int&, int,int \*); | 580 | Rolling dice |
| int rollDie(string [], char [][7],int&,int&, int); | 613 | Rolling Dice in Jail |
| void purp1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 643 | Mediterranean Ave Buy and rent costs |
| void purp2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 765 | Baltic Ave Buy and rent costs |
| void teal1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 886 | Oriental Ave Buy and rent costs |
| void teal2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1008 | Vermont Ave Buy and rent costs |
| void teal3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1133 | Connecticut Ave  Buy and rent cost |
| void pink1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1254 | St Charles Place  Buy and rent cost |
| void pink2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1376 | States Ave  Buy and rent cost |
| void pink3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1499 | Virgina Ave  Buy and rent cost |
| void orange1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1621 | St James Place  Buy and rent cost |
| void orange2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1746 | Tennessee Ave  Buy and rent cost |
| void orange3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1869 | New York Ave  Buy and rent cost |
| void red1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 1991 | Kentucky Ave  Buy and rent cost |
| void red2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2115 | Indiana Ave  Buy and rent cost |
| void red3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2239 | Illinois Ave  Buy and rent cost |
| void yellow1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2364 | Atlantic Ave  Buy and rent cost |
| void yellow2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2487 | Ventnor Ave  Buy and rent cost |
| void yellow3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2609 | Marvin Gardens  Ave  Buy and rent cost |
| void green1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2734 | Pacific Ave  Buy and rent cost |
| void green2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2857 | North Carolina Ave  Buy and rent cost |
| void green3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 2980 | Pennsylvania Ave  Buy and rent cost |
| void blue1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 3102 | Park Place  Buy and rent cost |
| void blue2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 3225 | Board walk  Buy and rent cost |
| void start(); | 4615 | Starting point |
| int chance(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool,int &,int &,int); | 3347 | Chance cards |
| int comChest(string[], char[][7],char[][7], int&,int&,int); | 3571 | Community Chest Cards(Not working) |
| int incTax(string[],int&,int&,bool); | 4406 | Income tax charges |
| int luxTax(string[], int&,int&,bool); | 4391 | Luxury tax charges |
| int jail(string[], char[][7],char[][7], int&,int&,int); | 4290 | Jail |
| int goToJail(string[], char[][7],char[][7], int&,int&,int); | 4378 | Go To Jail, |
| void rr1(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 3707 | Reading Railroad  Buy and rent cost |
| void rr2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 3850 | Pennsylvania railroad  Buy and rent cost |
| void rr3(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 3995 | B&O Railroad  Buy and rent cost |
| void rr4(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 4140 | Shortline Railroad  Buy and rent cost |
| void elecCom(string[], char[][COL],char[][COL], int&,int&,bool,int); | 4553 | Electric Company  Buy and rent cost |
| void watWork(string[], char[][COL], char[][COL],int&,int&,bool,int); | 4452 | Water works Ave  Buy and rent cost |
| int frePark(string[]); | 4284 | Free Parking |
| int menu(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 4620 | For placing houses on properties |
| void proView(string[],char[][COL],char [][COL],bool); | 5356 | Shows properties owned |
| void sell(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 5378 | Sell properties |
| void sellhs(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 6544 | Sell houses and hotels |
| void hsrep(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool,int=40,int=115); | 8547 | Calculate house repairs  For community chest |
| void hsrep2(string[], int[][COL],char[][COL],char[][COL], int&,int&, bool); | 8623 | Calculate house repairs for chance |
| void srtArry(string [ ]); | 8701 | Sorts array of property names in alphabetical order |

PseudoCode

do

Bool turn=false

Output menu choice

input menuCh;

If menuCh =‘1’ Player 1 rollDie();

If ‘menuCh =2’ View properties owned - proView(prop,playr1,playr2,turn);

If menuCh= ‘3’ Sell Property - sell(prop,rent, playr1,playr2,bank1,bank2,turn);

If menuCh= ‘4’ Sell House and Hotels- sellhs(prop,rent, playr1,playr2,bank1,bank2,turn);

If menuCh =‘5’ Buy houses and Hotels - menu(prop,rent,playr1,playr2,bank1,bank2,turn);

If menuCh =’6’ List of properties in the game- srtArry(prop);

Player1= rollDie()= dice roll total=sum of dice;

Function call( ) based on based on player 1 Dice Total;

Bool buy = false

Buy=property;

Choice= yes or no;

If yes bool switch to true;

Character b placed in array;

Collect rent if opponent lands on it

Pay rent if player 1 lands on opponents property

If player1>39 then player1-39

Bank1+=200;

Bool turn=true;

Bool turn=true

Output menu choice

input menuCh;

If menuCh =‘1’ Player 2 rollDie();

If ‘menuCh =2’ View properties owned - proView(prop,playr1,playr2,turn);

If menuCh= ‘3’ Sell Property - sell(prop,rent, playr1,playr2,bank1,bank2,turn);

If menuCh= ‘4’ Sell House and Hotels- sellhs(prop,rent, playr1,playr2,bank1,bank2,turn);

If menuCh =‘5’ Buy houses and Hotels - menu(prop,rent,playr1,playr2,bank1,bank2,turn);

If menuCh =’6’ List of properties in the game- srtArry(prop);

Player2= rollDie()= dice roll total=sum of dice;

Function call( ) based on based on player 1 Dice Total;

Bool buy = false

Buy=property;

Choice= yes or no;

If yes bool switch to true;

Character b placed in array;

Collect rent if opponent lands on it

Pay rent if player 2 lands on opponents property

If player2>39 then player1-39

Bank2+=200;

While(bank1>0 &&bank2>0)

If bank1<0

Player 1 losses

If bank2<0

Player 2 losses

End of Progra,m

Sample Output

Welcome to the game of Monopoly where fortunes are won....and Lost

How far will your bank roll take you.

Player One it is turn. Press enter to continue.

1

Player 1 0

Bank 1 2000

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

1

You rolled a 1 and 5 which equals 6

Would you like to make another menu selection

n

Welcome to Oriental\_Ave

Player one this property is not purchased yet. Would you like

to purchase this property for $100 Press Y for Yes and No for no.

y

Player 1 Bought Oriental\_Ave

Player Two it is your turn. Press Enter to continue

Player 2 0

Bank 2 2000

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

1

You rolled a 4 and 5 which equals 9

Would you like to make another menu selection

n

Welcome to Connecticut\_Ave

Player two this property is not purchased yet. Would you like

to purchase this property for $120. Press Y for Yes and No for no.

y

Player 2 Bought Connecticut\_Ave

Player One it is turn. Press enter to continue.

Player 1 6

Bank 1 1900

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

2

Player one you own the following properties:

Oriental\_Ave

Would you like to make another menu selection

y

Player 1 6

Bank 1 1900

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

1

You got doubles!! You get to go again!!

You rolled a 1 and 1 which equals 2

Would you like to make another menu selection

n

Welcome to Vermont\_Ave

Player one this property is not purchased yet. Would you like

to purchase this property for $100 Press Y for Yes and No for no.

y

Player 1 Bought Vermont\_Ave

Player One it is turn. Press enter to continue.

Player 1 8

Bank 1 1800

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

1

You rolled a 3 and 5 which equals 8

Would you like to make another menu selection

1

Welcome to St.\_James\_Place

Player one this property is not purchased yet. Would you like

to purchase this property for $180 Press Y for Yes and No for no.

y

Player 1 Bought St.\_James\_Place

Player Two it is your turn. Press Enter to continue

Player 2 9

Bank 2 1880

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

1

You got doubles!! You get to go again!!

You rolled a 1 and 1 which equals 2

Would you like to make another menu selection

1

Welcome to St\_Charles\_Place

Player two this property is not purchased yet. Would you like

to purchase this property for $140. Press Y for Yes and No for no.

y

Player 2 Bought St\_Charles\_Place

Player Two it is your turn. Press Enter to continue

Player 2 11

Bank 2 1740

Please make a menu selection:

1: Roll Dice

2: View properties owned

3: Sell Property

4: Sell House and Hotels

5: Buy houses and Hotels

6: List of properties in the game

RUN TERMINATED (exit value 1, total time: 57s

Flowchart

Note a copy of this flowchart is in the project 2 folder

This is just for player one and illustrates what happens on this turn. The same is true for player 2 except some variables switch. There are a few functions illustrated that are representative of the whole;

