Project Report of Lab Batch 56

Index

Project Description

This is essentially an effort to digitize the paper and physical version of the famous game monopoly. For more information on the game, please refer to the documentation which has clear details about the nitty-grittys of the game. All the monopoly jargon is also defined in the documentation.

Initial Project Objectives

1.GUI

Creating a GUI, which is **user friendly**, **intuitive** and **takes all input from the window** itself, without having the need to go back to the command screen. This would also involve mouselicks, timer functions and videos generated by a collection of images.

2. Coding/Algorithms

A game such as monopoly involves a lot of coding. The computer needs to check for a given set of conditions (eg. Ownership, Bank Balance) and provide a set of options (buying, selling, paying a penalty, mortgaging). It is due to these numerous functionalities that the coding part is complex. Our aim is to create code to go with the GUI and give all options.

3. Data Handling

The facilities for load game and saved game will be created. Structures will be created for the properties and players.

Functionalities offered by the program

Before going through this section, it is strongly recommended that you refer to the attached documentation for understanding the definition of the terms.

1. Buying Properties
2. Selling Properties
3. Mortgaging Properties
4. Unmortgaging Properties
5. Develop Properties
6. Paying rent
7. Rolling the dice

Overview of the final code

Part 1 – Input data

In EzWindows, The API used by us to create the GUI, each class is associated with a window. Even Mouseclick based functions are linked to a unique window. Hence, if we required several mouseclick functions, we used separate windows.

The game starts with a welcome window, with an embedded bitmap. A function is used in this window, which registers a mouse click and passes the control to the code inside the function. Actually, the rest of the code is wholly contained indirectly inside this function.

The next screen shows a few bitmaps. A further mouse click function is called to see, if the mouse pointer click was within a certain area of the window. If the bitmap is within 'Infinite', the game runs till all but one player are bankrupt. Else, the game is timed, and the net worth of each player is calculated at the end of the stipulated time. The player having greater worth wins. This is implemented by changing the condition for the while loop under which our main code runs. Previously it would have been, while(player[1].bankbalance>0|| player[2].bankbalance>0|| player[3].bankbalance>0), it would also involve a while(clock_t <final time>-clock_t <initial time > < clock _ t gametime) term.

The next window would involve the selection of the number of players. The code here is pretty straightforward, but one part worth explaining is the necessary action, when one mouseclick is registered, but does not occur within an area which leads us to the next part. This is done, by loading the next window and the bitmaps in that window iff the click is in one of the desired windows.

The next part is the selection of the colour marker for each player. The code here is not particularly efficient as, depending on the number of players previously selected, the number of times the screen must repeat itself varies. This part of the code can definitely be developed.

Part 2 – Game Play

After the game mode, the number of players and their choices of colour are selected, the main screen is displayed. This involves the loading of all bitmaps related to the sites, and all the buttons for the game.

An iteration of the main game function is defined by a single unique variable called p, the player identifier. All changes to the player and board are made with respect to this p. After the end, the function is called again within the same function, provided, the while loop for bankruptcy and time are satisfied.

The game involves a gameplay variable called **rolldicestatus.**

If this variable is put to 0, the program is designed to not allow to player to buy, sell or take any action. He can only check the info about any place and roll the dice. After the roll die button is called, a function is invoked, in which there are more functions, like generation of the random dice values, simulating the dice video in another window, and updating the player position. If the player position is greater than a certain vale, it is made 0 again. A message window is also displayed which might say “This property (new) belongs to you” or “You can buy this property “ or “ You have to pay a rent “. This ends the iteration. After the end of an iteration of the main game function, the value of p is not updated.

If the roll dice status is updated to 1, all the options are made available. If the player clicks on any button, a new window opens regarding that function. For example, If one clicks buy, the window, buywindow opens. If the person further clicks on the confirmational yes, the buy function is invoked, and the player number updated. Also the rolldicestatus is set to 1. This implies all the other functions are restricted to the player. As according to the monopoly rules, the player can take only one decision per turn.

The functions for others are also similar.

Now, we show you an example of the functions. It would serve two purposes, reveal info about the data structures and also the functions part.

Example

```
void buying(int p){//p for player no
   if((college[player[p].boardpos].category)==(1||2||3)){
      if(college[player[p].boardpos].owner_no==0){
         player[p].bankbalance-= college[player[p].boardpos].cost;
         college[player[p].boardpos].owner_no==p;
      }
   }
}
```

Here, the function first check, if property is unowned and buyable. To do this, the player position is computed as (player[p].boardpos), then the owner number of that position is calculated and checked against p. Also the category is checked. If the conditions are met, the property owner is made p, and the rent of the propert is deducted from p's account.

Another Example

```
void develop_lab(int p){//developing a site when the player owns 3 or more cards of same colour code
& he is in that particular position
    if (college[player[p].boardpos].owner_no==p) {
        int  q,i;
        count=0;//count for counting the number of properties he owns of a particular colour & q for
storing the property colour code
        q=college[player[p].boardpos].colour;
        for (i=0;i<=39;i++)
            if (college[i].colour==q)
                if (college[i].owner_no==p)
                    count++;
    }

    if (count>=3) {
        if (college[player[p].boardpos].no_of_labs<3) {
            college[player[p].boardpos].no_of_labs++;
            player[p].bankbalance-=college[player[p].boardpos].cost_of_a_lab;
        }
        else if(college[player[p].boardpos].no_of_hostels<1){
            college[player[p].boardpos].no_of_hostels++;
            player[p].bankbalance-=college[player[p].boardpos].cost_of_a_hostel;
        }
    }
}
```

This function is invoked to develop properties. First, it is checked if the property is owned by player itself. Then the colour of the group to which the property belongs is found and checked if the player owns 3 of the same colour. Then the number of labs in the property are less than the limit, else it builds a hostel and then checks the hostel limit. If even this is not met, The control exits from the function.