**MONOPOLY- THE INDIAN EDITION**

A

Mini Project Report

By

**G.UTTEJ KUMAR REDDY 1602-15-733-114**

**P.V.SAI NIKHIL 1602-15-733-115**

****

**Department of Computer Science & Engineering**

**Vasavi College of Engineering**

**(Autonomous)**

**(Approved by A.I.C.T.E)**

**9-5-81, Ibrahimbagh, Hyderabad-31**

**2016**

**ACKNOWLEDGEMENT**

We take this opportunity with enormous gratitude, to express gratefulness to our respectable guide **Mrs.** **Syamala,** Department of Computer Science and Engineering whose guidance was unforgettable and whose constructive suggestions has made the presentation of my project a grand success.

**ABSTRACT**

Monopoly is the most commercially-successful board game in the United States history. It first originated in 1903. The current version of the game was first published in 1935 by the Parker Brothers. The game is named after the economic concept Monopoly – the domination of a market by a single entity.

The game has various cities priced according to their cost of living and player has to play till all of the market belongs to him or till he forces a fellow player into bankruptcy. This game can be played by a maximum of 6 players, including computer players.

This game is released in many different versions containing cities of various countries. But till date, an Indian version of this phenomenal game has not been released. This is what we are endeavoring to do; a game of Monopoly with Indian cities and played with Indian currency.

**SOFTWARE: C++**

**TOOLS : QT CREATOR**

**TABLE OF CONTENTS**

1. QT Designer
   1. Introduction
   2. Projects
   3. Headers (.h) Files
   4. Source (.cpp) Files
   5. Forms (.ui)
2. Monopoly
   1. Introduction
   2. The Rules of the Game
      1. The Object of the Game
      2. Equipment
      3. Preparation
      4. Banker
      5. The Play
      6. GO
      7. Buying Property
      8. Paying Rent
      9. Chance and Community Chest
      10. Income Tax
      11. Jail
      12. Free Parking
      13. Houses
      14. Hotels
      15. Mortgages
   3. List of classes and their functions
      1. Cards\_details
      2. Confirmbankruptcy
      3. Confirmbuild
      4. Confirmpurchase
      5. Credits
      6. Data
      7. Gamedata
      8. Homepage
      9. Howtoplay
      10. Injail
      11. Lowoncash
      12. Mainboard
      13. Messagewindows
      14. Mortgagetitles
      15. Mycards
      16. Numberintake
      17. Players2
      18. Players3
      19. Players4
      20. Players5
      21. Players6
      22. Playerstatus
      23. Rolled
      24. Rolled\_occ\_deficit
      25. Rolled\_occupied
      26. Rolled\_own
      27. Specialcards
      28. Tradepreliminaries\
      29. Tradeprelims2
      30. Tradingoptions
3. Screenshots
4. References

**LIST OF FIGURES**

* 1. QT Logo and Motto
  2. Files included under Monopoly project (Final\_monopoly.pro)
  3. Header Files
     1. User defined Header File (gamedata.h)
     2. Header File with ui functions
  4. Source File
     1. Including different header files in a source file
     2. Constructor for ui objects
  5. Forms
     1. Design of the ‘mainwindow’ window in QT Designer
     2. The generated xml code for the ‘mainwindow’ window
  6. Interconnection of Windows

1. Screenshots
   1. Home Page
   2. Number of Players Intake
   3. Player Details
   4. Main Board
   5. Rolled onto FreeSpace
   6. Purchase Confirmation
   7. Rolled onto Chance Card
   8. Rejection of Purchase due to Insufficient Funds
   9. Mortgage window
   10. Trade Preliminaries
   11. Trade Window
   12. Paying rent
   13. Bankrupt Confirmation
   14. Bankrupted
   15. Winner Declaration
2. **QT Designer**

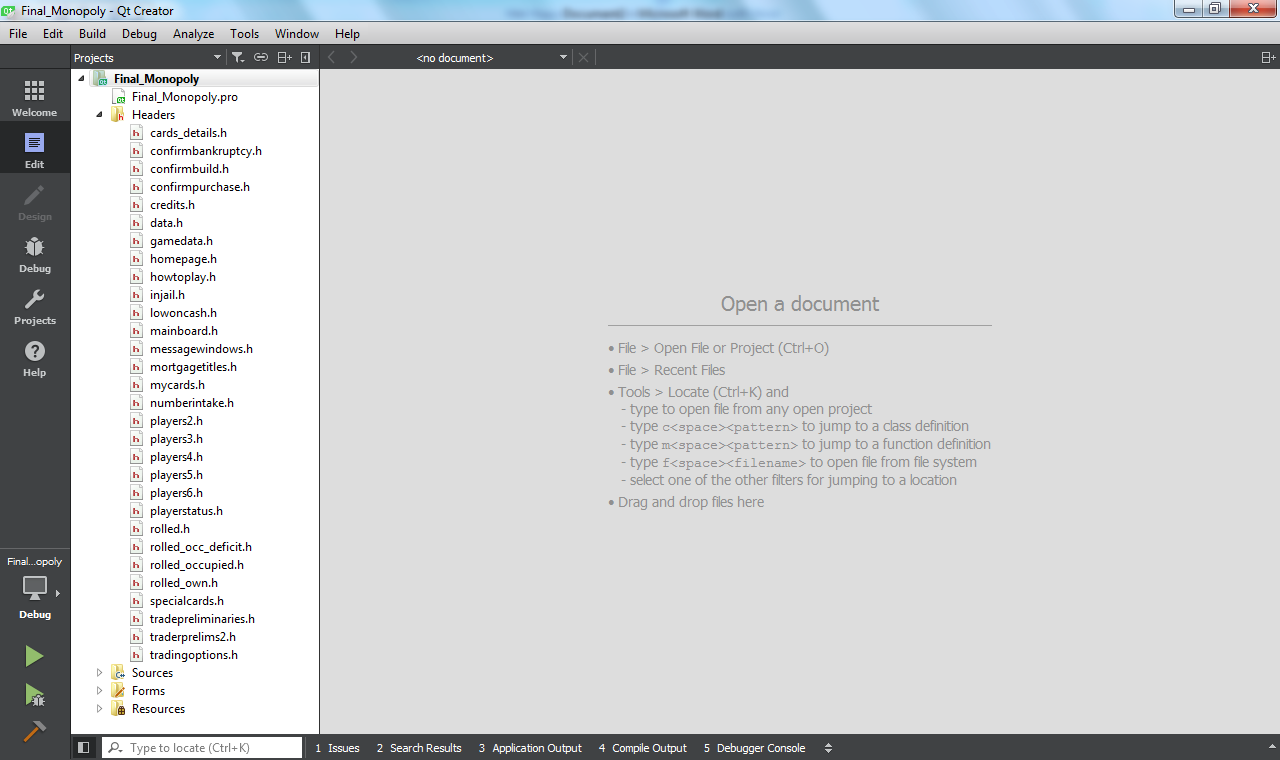
**1.1.Introduction**

**1.1 Qt – Logo and Motto**

Qt creator is a cross platform C++, JavaScript and QML Integrated Development Environment which is part of the SDK for the QT Integrated Development Framework. It includes a visual debugger and an integrated GUI layout and forms designer. The editor’s features include syntax highlighting and auto-completion but purposely not tabs (although plug-ins are available). QT Creator uses C++ compiler from GNU Compiler Collection on Linux and FreeBSD. On windows, it can use MinGW or MSVC with the default install and can also use Microsoft Console Debugger when compiled from source code.

**1.2.** **Projects**

Qt includes a project manager that can use a variety of project formats such as .pro, CMake, Autotools and others. A project file can include information such as what files are included into the project, custom build steps and settings for running the application.



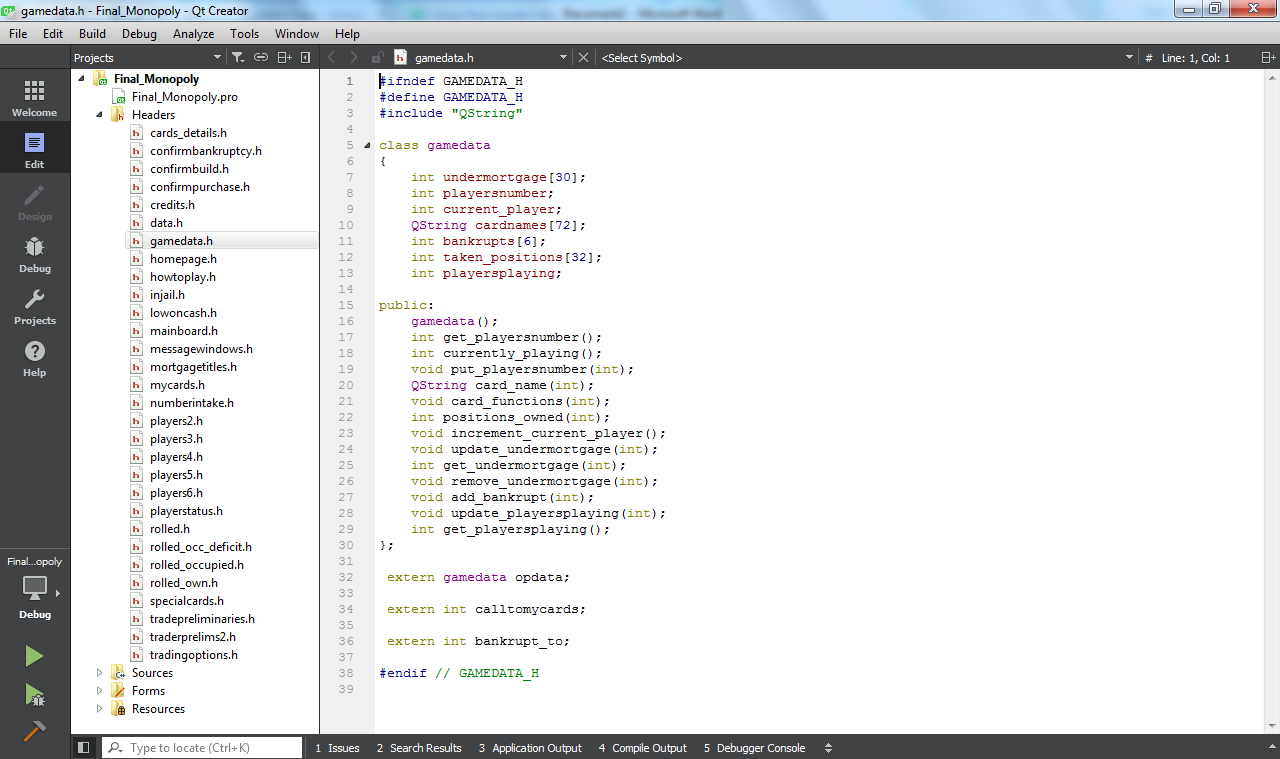
**1.2 Files included under Monopoly project (Final\_monopoly.pro)**

The project we built includes header files, source files, the .ui files and the resource files. Each of their functionality is as below.

**1.3. Header (.h) Files**

Precompiled headers are a performance feature supported by some compilers to compile a stable body of code, and store the compiled state of the code in a binary file. During subsequent compilations, the compiler will load the stored state, and continue compiling the specified file. Each subsequent compilation is faster because the stable code does not need to be recompiled.

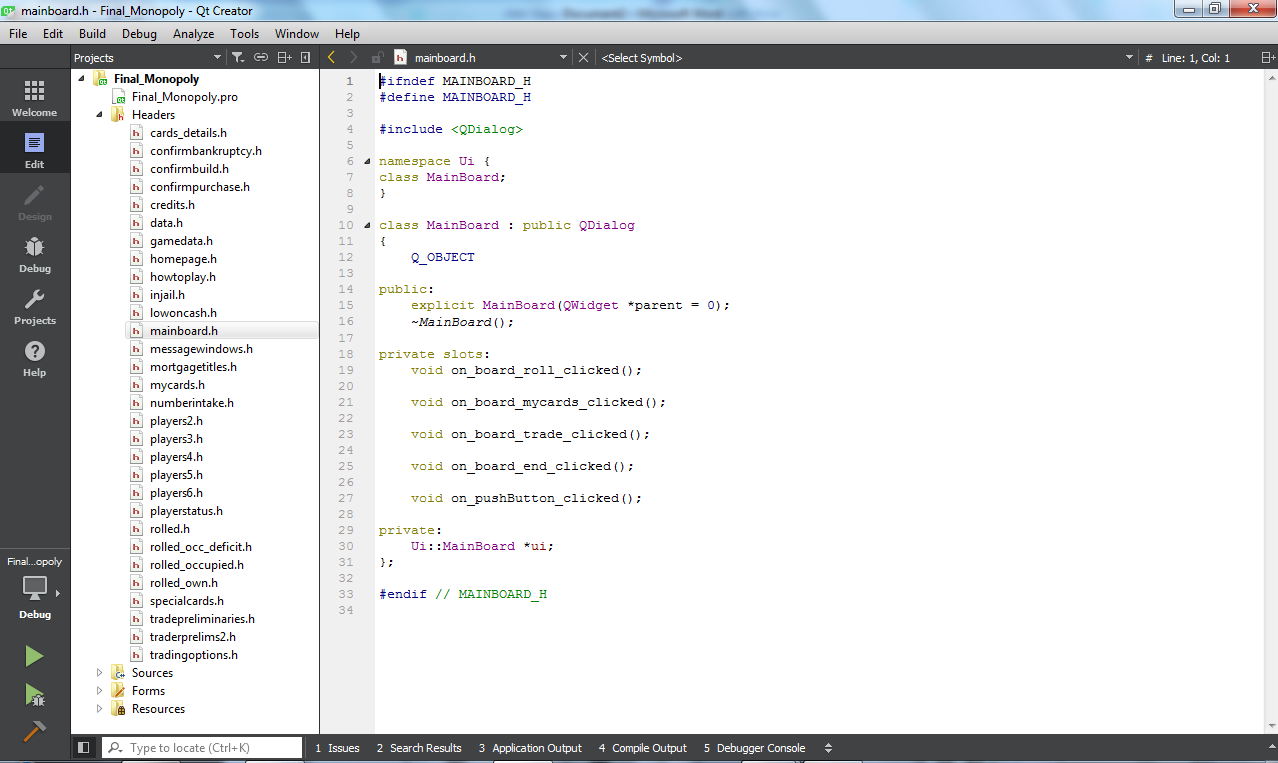
For example, <QMessagebox> is repeatedly used in our project to give status information to the player. The QMessagebox object can be called anywhere by including the header file “QMessagebox.h” where it is defined. This is a predefined header file by Qt. For a user defined header file example, gamedata.h is a header file defined by us and which contains data which is to be used throughout the running of the program. For this purpose, “gamedata.h” is included in every source file used.



**1.3.1 User defined Header File (gamedata.h)**

Then there are the header files which include the objects created by the user in the corresponding forms (.ui files). These objects too are declared in the header file and their definition is done in the source (.cpp) files.

In the following window, void on\_board\_roll\_clicked(), on\_board\_mycards\_clicked(), on\_board\_trade\_clicked(), on\_board\_end\_clicked() are the declarations of the push buttons included on the mainboard in the project. “on\_board” tells the position where the push button objects (“roll”,”mycards”,”trade”,”end”) are present and “clicked” tells the action which toggles the functions associated with their definitions.

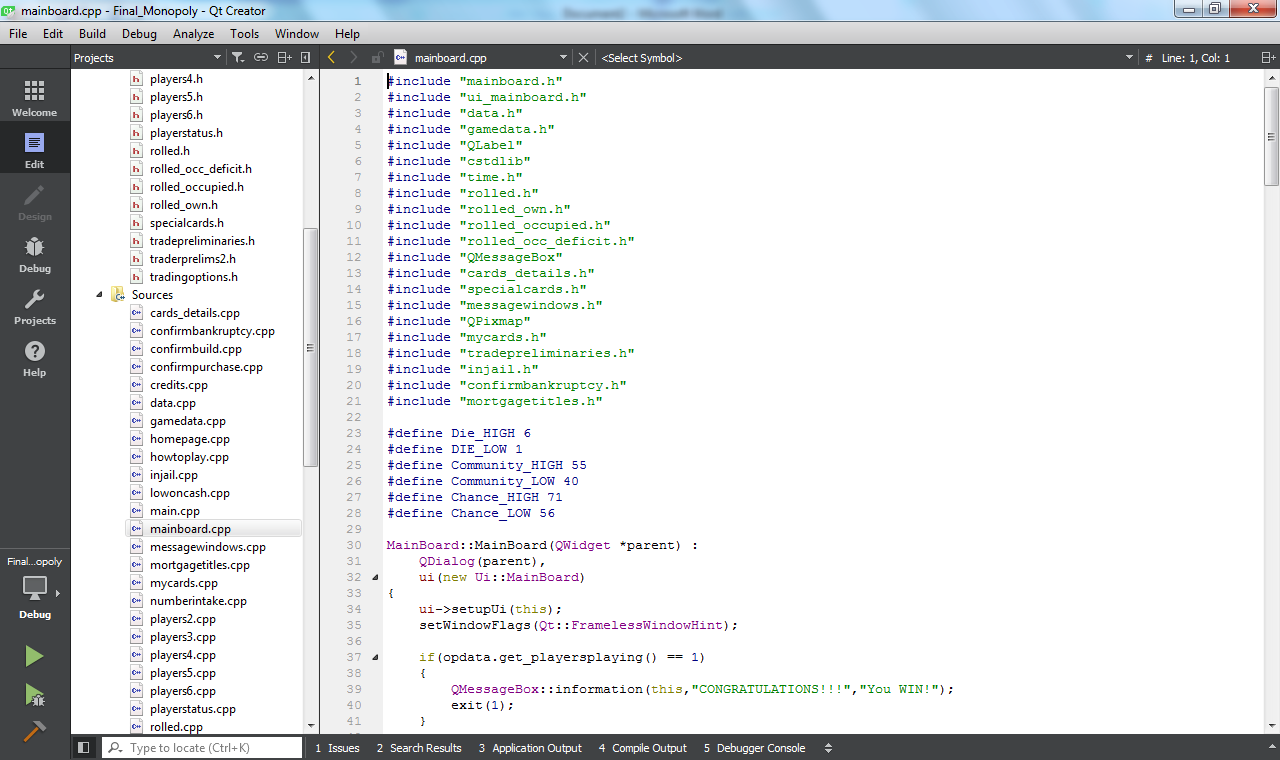


**1.3.2 Header File with ui functions**

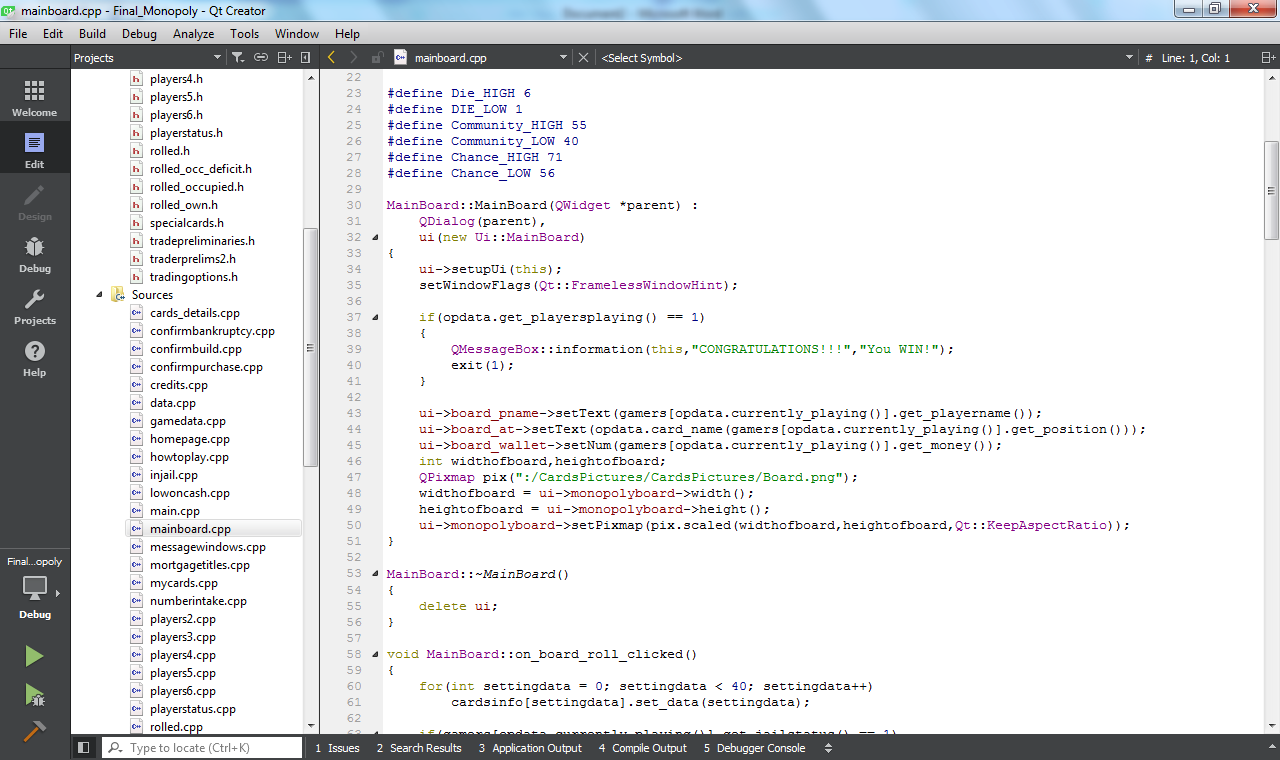
**1.4. Source (.cpp) Files**

The constructors for data members and the definitions of member functions declared in the corresponding header file of the class are written here. Apart from these, functions of the ui objects are also coded in .cpp files. New functions can also be defined. These are generally accessible only within the defined source file. The data which is to be shared between different source files is defined in a header file as stated and is accessed by including that header at the beginning of the source file. The windows which are accessed by toggling an action is also considered to be data and the method to access these windows from another is the same as above, i.e, including the called window in the calling window.

In the following screenshots, the mainboard source file is seen to include many windows as it can be considered to be the core window. In the next screenshot, the constructor of the mainboard is seen in which the ui labels included in forms are initialised.



**1.4.1 Including different header files in a source file**

****

**1.4.2 Constructor for ui objects**

**1.5. Forms (.ui)**

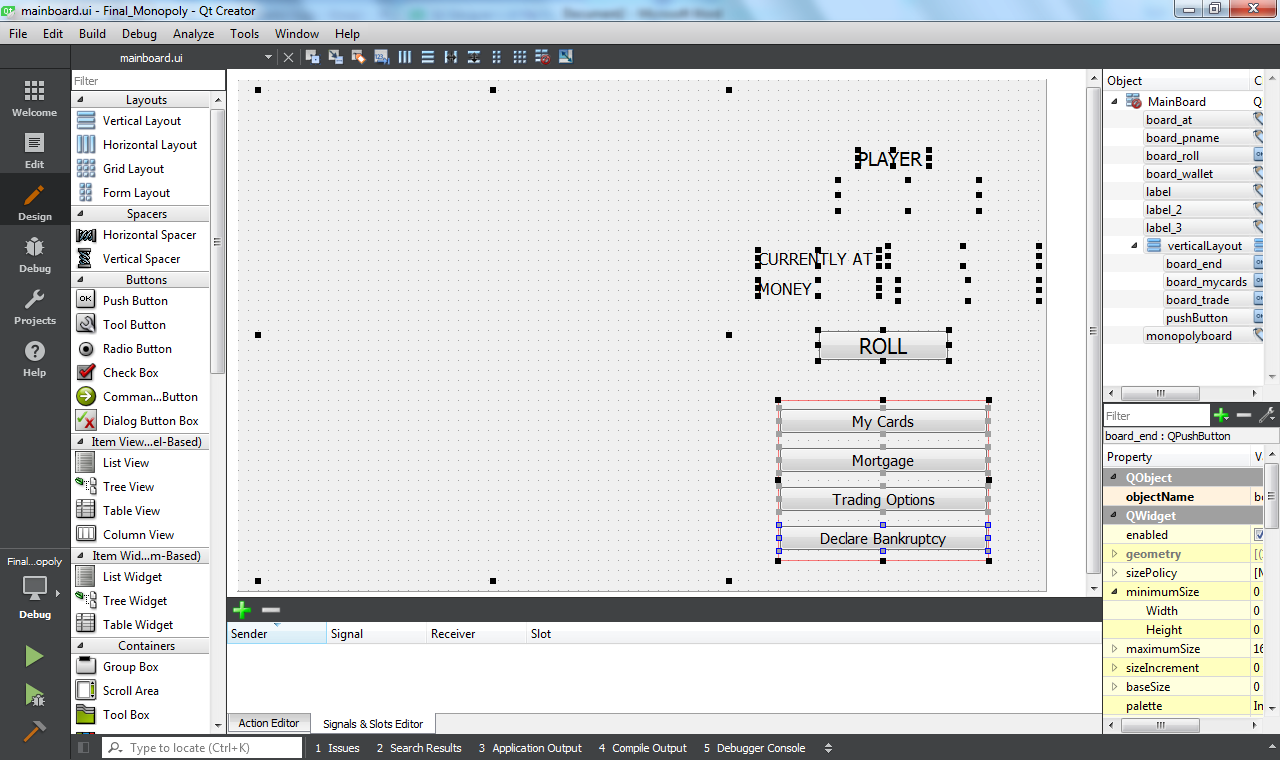
The user interface of the project is done in the forms. The .ui files can be constructed in two ways in Qt. In the Qt Designer mode the window can be designed by dragging and dropping pre-defined widgets available in designer mode. The properties of the widgets, which are objects of pre-defined classes, can be then changed and their functions can be defined in the corresponding source files by selecting the toggle method of the objects.

Qt automatically generates “xml” code for the layout of the dialog designed by the user and includes into the project file. Users can also write their own xml code for the display but it is tedious and time-taking but the written xml code is more customizable.

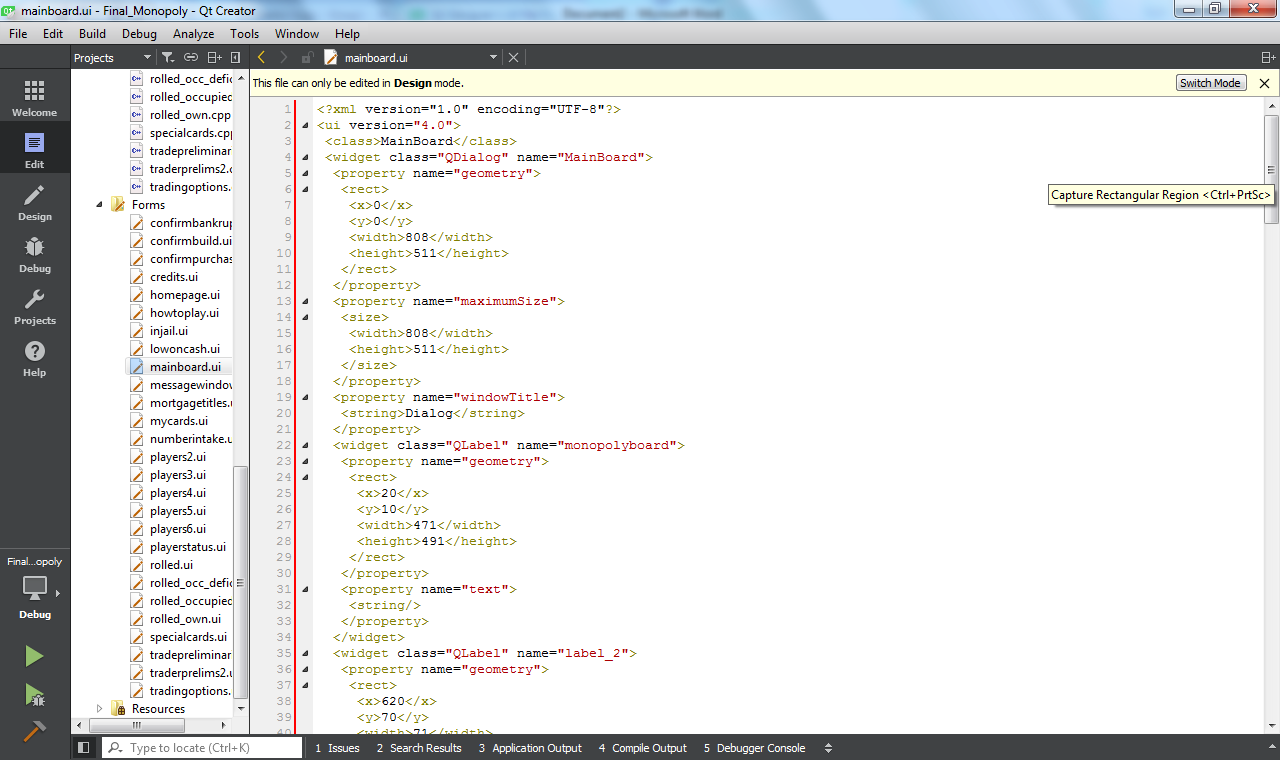
Seen in 1.5.1 is the layout of the ‘mainboard’ window. To the left side of the screen are the all widgets available in Qt Design mode. The used widgets in this particular window are the output widgets, labels and buttons widgets, pushbutton. The toggle mode for the pushbutton is set to clicked() and when clicked, the function to run is defined in the .cpp file. Seen on the right are various widget objects included in this form. Below it is the object property editor wherein the properties of the selected objects can be customised. Properties such as name, size, toggle mode and any other specifics are included in the object editor.

Seen in 1.5.2 is the xml code generated by Qt for the layout designed by the user. The xml code for the mainboard layout is of 210 lines and coding this manually would take a lot of time and effort. Herein lies the advantage of using Qt designer.

More widgets not seen in the following pictures are used in the project namely, the input widgets, LineEdit which is used to take in strings from the user, SpinBox used to take in integers, CheckBox to select multiple options, Radio Button to select a single option, and the output widget Text Browser, used to show large amount of text to the user.



**1.5.1 Design of the ‘mainwindow’ window in Qt Designer**

****

**1.5.2 The generated xml code for the ‘maiwindow’ window**

**2.0. Monopoly**

**2.1. Introduction**

***Monopoly*** is a board game that originated in the United States in 1903 as a way to demonstrate that an economy which rewards wealth creation is better than one in which monopolists work under few constraints and to promote the economic theories of Henry George and in particular his ideas about taxation. The current version was first published by Parker Brothers in 1935. Subtitled "The Fast-Dealing Property Trading Game", the game is named after the economic concept of monopoly—the domination of a market by a single entity. It is now owned and produced by the American game and toy company Hasbro. Players move around the game-board buying or trading properties, developing their properties with houses and hotels, and collecting rent from their opponents, with the goal being to drive them all into bankruptcy leaving one monopolist in control of the entire economy.

The game Monopoly – The Indian Edition is constructed using the existing rules of monopoly. The rules on which our project works are as follows. The first paragraph under the sub-heading shows the rules of the tangible game and the second paragraph shows how the cases are achieved in coding.

**2.2. The Rules of the Game**

**2.2.1. The Object of the Game**

The object of the game is to become the wealthiest player through buying, renting and selling of property.

When the number of players playing the game gets down to one, the game ends and the remaining one player is declared the winner. This is achieved by a playersplaying variable in gamedata which decrements each time a player leaves the game or gets bankrupt.

**2.2.2. Equipment**

The equipment consists of a board, 2 dice, tokens, 32 houses and 12 Hotels. There are 16 Chance and 16 Community Chest cards, 28 Title Deed card (one for each property), and play money.

The 2 dice are two random number generating functions which generate a number between 1 to 6, including 1 and 6, The tokens are disregarded and the position of the player is shown using a label in the ui. All the data of the player, game and cards is stored in the classes data, gamedata and cards\_details. The data members are generalized to common data which is possessed by individual entities with unique values and these values are initialized using constructors and the ability to modify them is given to member functions.

**2.2.3. Preparation**

The board and cards are set and money is distributed to the players. The players choose their tokens and the dice is rolled to start the game.

The number of players playing the game is taken in, the names of each individual player is then taken. The position of the players is initialized to zero and each player is given 2,00,000 rupees money to spend by assigning the value of money data member in data class to 20000.

**2.2.4. Banker**

A player is set as a banker and the bank’s funds and properties are kept separately.

The computer acts as the banker with the class data, gamedata and card\_details being the main classes it manipulates the data in.

**2.2.5. The Play**

Place your token on the corner marked "**GO**", then throw the dice and move your token. After you have completed your play, the turn passes to the left. The tokens remain on the spaces occupied and proceed from that point on the player's next turn. Two or more tokens may rest on

the same space at the same time. Depending on the space your token reaches, you may be entitled to buy real estate or other properties, or be obliged to pay rent, pay taxes, draw a Chance or Community Chest card, Go To Jail, or etc.

The same results are achieved by using the functions in gamedata.

**2.2.6. GO**

Each time a player's token lands on or passes over **GO**, whether by throwing the dice or drawing a card, the Banker pays that player a $200 salary.  
200000 is added to the player’s wallet when the player’s position exceeds 39 (0 to 39 being the values assigned to each square present on the board.)

**2.2.7. Buying Property**

Whenever you land on an unowned property you may buy that property from the Bank at its printed price. You receive the Title Deed card showing ownership. Place the title deed card face up in front of you.

“rolled” class takes care of this. The status of the card in cards\_details is changed from 0 to 1. The card number is added to the possession of the player and the money is deducted from the money data member using subtract\_money member functions.

**2.2.8. Paying Rent**

When you land on a property that is owned by another player, the owner collects rent from you in accordance with the list printed on its Title Deed card.  
If the property is mortgaged, no rent can be collected. When a property is mortgaged, its Title Deed card is placed face down in front of the owner.  
It is even more advantageous to have houses or hotels on properties because rents are much higher than for unimproved properties. The owner may not collect the rent if they fail to ask for it before the second player following throws the dice.

The rolled\_occupied and rolled\_occ\_deficit classes are used for this purpose. If the user has enough money to pay the rent, the control goes to rolled\_occupied, otherwise the control goes to

rolled\_occ\_deficit. The money is deducted if in rolled\_occupied and control is given to next player. In the rolled\_occ\_deficit, the player can select from three options, Mortgage, Trading Options and Declare Bankruptcy which are explained in following sections.

**2.2.9. Chance and Community Chest**

When you land on either of these spaces, take the top card from the deck indicated, follow the instructions and return the card face down to the bottom of the deck. The "**Get Out of Jail Free**" card is held until used and then returned to the bottom of the deck.

The specialcards class is called when a player falls on the chance or community space. A random number is then generated and the card associated with that random number is displayed and the functions of that card, defined in cards\_details class is executed.

**2.2.10. Income Tax**

If you land here, you may estimate your tax at 200000 and pay the Bank.

Money is subtracted from the player. In case the money the player has is less, the lowoncash class is called which contains the Mortgage, Trading Options and Declare Bankruptcy options.

**2.2.11. Jail**

You land in Jail when, your token lands on the space marked **"Go to Jail".** When you are sent to Jail you cannot collect your 200000 salary in that move since, regardless of where your token is on the board, you must move directly into Jail. Your turn ends when you are sent to Jail. If you are not "sent to jail" but in the ordinary course of play lands on that space, you are  
**"Just Visiting"**, you incur no penalty, and you move ahead in the usual manner on your next turn. You still are able to collect rent on your properties because you are **"Just Visiting"**.  
  
A player gets out of Jail by doing one of the following two things:   
(1)Throwing doubles on any of your next three turns,If you succeed in doing this you immediately move forward the number of spaces shown by your doubles throw. Even though you had thrown doubles, you do not take another turn.  
(2)Paying a fine of 10000 before you roll the dice on either of your next two turns. If you do not

throw doubles by your third turn, you must pay the 10000 fine. You then get out of Jail and immediately move forward the number of spaces shown by your throw. Even though you are in Jail, you may collect rents.

When a player lands in jail, the data member jailstatus in data class is changed to 1.

The getting-free conditions are achieved by the injail class. It has two functions in it which either accepts payment of 10000 or chooses to roll the dice. When the dice is rolled random numbers and generated and compared and the associated functions executed. The Jail Status of the player is changed back to 0 when he gets free

**2.2.12. Free Parking**

A player landing on this place does not receive any money, property or reward of any kind.  
This is just a "free" resting-place.

Control is given to the next player.

**2.2.13. Houses**

When a player owns a property, they may buy houses from the Bank and erect them on that property. The price you must pay the Bank for each house is shown on your Title Deed card for the property on which you erect the house.

The option to build a house is shown when the player lands on his own property. He may build any number of houses, between 1 to 4, on that property, at a time as long as he has the money.

**2.2.14. Hotels**

When a player has four houses on a property, he may buy a hotel from the Bank and erect it on that property. Only one hotel may be erected on any one property.

The option for this is also shown as in the case of building houses. This option can only be toggled when there are 4 hotels built on that property. In case of not having 4 properties, the player is redirected to the card he owns/

**2.2.15. Mortgages**

Properties can be mortgaged through the Bank at any time when a player’s turn comes. The mortgage value is printed on each Title Deed card. No rent can be collected on mortgaged properties or utilities.

The mortgage option is present on the mainboard ui and can accessed at any time when that player’s chance comes. The mortgage option can also be invoked when a player is in need of money, either from the rolled\_occ\_deficit class or lowoncash class. The mortgagetitles class is invoked when the option is selected and the names of the titles which the player owns is shown. The player can tick whichever titles he chooses to mortgage and the corresponding mortgage value is added to the player’s wallet.

**2.2.16. Bankruptcy**

You are declared bankrupt if you owe more than you can pay either to another player or to the Bank. If your debt is to another player, you must turn over to that player all that you have of value and retire from the game.

Should you owe the Bank, instead of another player, more than you can pay (because of taxes or penalties) even by selling off buildings and mortgaging property, you must turn over all assets to the Bank. A bankrupt player must immediately retire from the game. The last player left in the game wins.

When Declare Bankruptcy option is toggled from either mainboard class or the classes which are invoked when there is money defieciency, the playing status of the player is changed to 0 and depending on the position where he declared bankruptcy, all the cards and money return to either the bank or another player. The playersplaying is decremented by 1 and whenever the turn of player comes, the playing status of that player is checked and if it is 0, the turn goes to the next player. In this way, a bankrupted player is effectively removed from playing.

**2.3. List of classes and their functions**

* + 1. **cards\_details**

The details of all the cards is stored in this class. The rents of the properties, the house costs, mortgage values and the player who possesses that particular card is stored in this class. Apart from these, the code to be executed for a particular chance or community chest card is also stored in this class. There is no .ui file for this class.

* + 1. **confirmbankruptcy**

This class presents two options, “Yes” or “No”. When the “yes” option is toggled, that player is declared bankrupt and all the changes to the data mentioned in 2.2.16 are done. When the “no” option is toggled, the player is returned to the previous window from which the confirmbankruptcy window was accessed.

* + 1. **confirmbuild**

When a player selects to build a house, this window appears to ask for confirmation.

* + 1. **confirmpurchase**

When a player chooses to buy a property, this window is accessed and it shows the cost of the property and the remaining amount a player will have in case of buying it.

* + 1. **credits**

This window contains the credits of the project.

* + 1. **data**

Player data such as name, position, possessions, houses and hotels owned, playing status, jail status, bankrupt status are stored in this class. This is one of the 3 core classes and the data of this class is accessed by almost all other classes. It has no ui.

* + 1. **Gamedata**

This is also one of the 3 core classes which has the number of players playing the game, the cards under mortgage, card names, the current player and the players who went bankrupt as well as the titles which are bought.

* + 1. **homepage**

New game, Credits and How to Play windows are accessed through this window. This is the window which opens when the application is opened.

* + 1. **howtoplay**

Has a text browser which shows how the game works.

* + 1. **injail**

Shows the in-jail options; throw the dice for a double score or pay 10000 to get out of jail. The corresponding functions are executed when clicked upon as explained in 2.2.11.

* + 1. **lowoncash**

Shows the options; mortgage, trading options and declare bankruptcy and when clicked upon goes to mortgage window, tradingpreliminaries window and confirmbankruptcy window respectively.

* + 1. **mainboard**

The main window of the project to which every player comes back to, it contains the option to roll the dice and depending on the outcome of it, accesses one of the following: rolled, rolled\_occupied, rolled\_occ\_deficit, rolled\_own. It also contains options My Cards, Mortgage, Trading options and Declare Bankruptcy which access mycards, mortgagetitles, tradingpreliminaries and confirmbankruptcy classes respectively.

* + 1. **messagewindows**

In special cases like Just Visiting the Jail and landing on Free Parking, this window is called wherein it shows the dice thrown and the position of the player.

* + 1. **mortgagetitles**

It contains 28 check boxes which show names of cards if and only if the player who called the window has that card in his possession. The player can check whichever card he wants to mortgage and the corresponding amount will be added to his wallet.

* + 1. **mycards**

Shows all the cards in possession of the player.

* + 1. **numberintake**

Takes in the number of player who are playing the game using radio buttons.

* + 1. **players2**

Takes in the names of 2 players if the number of players playing the game is 2.

* + 1. **players3**

Takes in the names of 3 players if the number of players playing the game is 3.

* + 1. **players4**

Takes in the names of 4 players if the number of players playing the game is 4.

* + 1. **players5**

Takes in the names of 5 players if the number of players playing the game is 5.

* + 1. **players6**

Takes in the names of 6 players if the number of players playing the game is 6.

* + 1. **playerstatus**

Shows the status of the player after the execution of code of the corresponding chance or community chest card he fell on.

* + 1. **rolled**

In case the player rolls onto a title which is occupied by none, this window is toggled. It has the options “Buy” and “Skip”. The player can either buy the property or skip to do so.

* + 1. **rolled\_occ\_deficit**

In case the player lands on a place occupied by another player and has not sufficient money to pay the rent, this window is accessed. It has mortgage, tradingoptions and declare bankruptcy options in it.

* + 1. **rolled\_occupied**

When a player lands on a title occupied by another player and has enough money to pay the rent, he is taken to this window. The money is subtracted from the player in form of the rent associated with the place, depending on the number of houses and hotels built on it.

* + 1. **rolled\_own**

When a player rolls on a title which he owns, this window is accessed. It has the build houses, build hotel and skip options. The options are self explanatory and their functions are as stated in 2.2.13 and 2.2.14.

* + 1. **specialcards**

When the player falls on chance or community card, this window is called. A random generator generates a number which decides the chance/community chest card which the user takes. The OK option executes the corresponding code associated with the picked card.

* + 1. **tradepreliminaries**

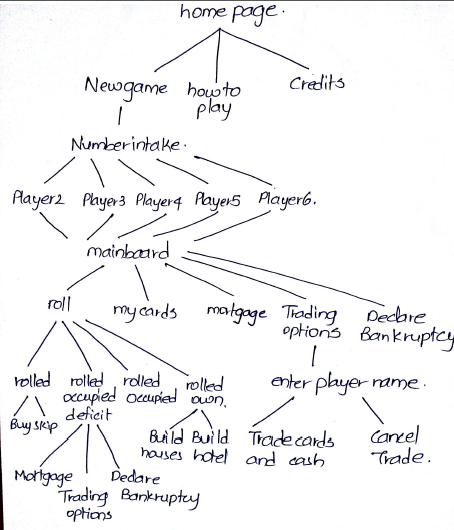
When Trading Options function is triggered, this window is opened. In this, the calling player has to enter the name of the player who he chooses to trade with.

* + 1. **tradeprelims2**

The player who is asked to trade with is asked for confirmation of trade.

* + 1. **tradingoptions**

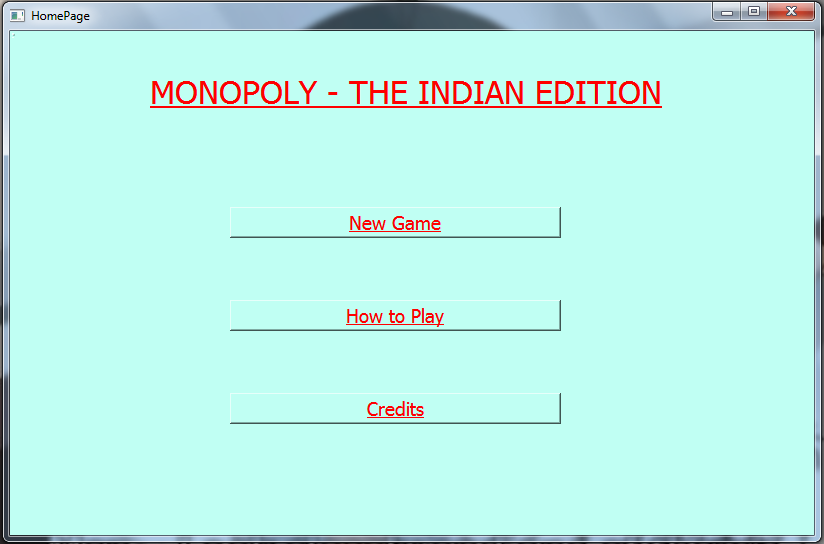
The cards of both the parties are displayed in this window. There is a spinbox through which the amount to be transferred by either player can be chosen. When the Trade Cards and Cash button is clicked, the necessary trade is done.



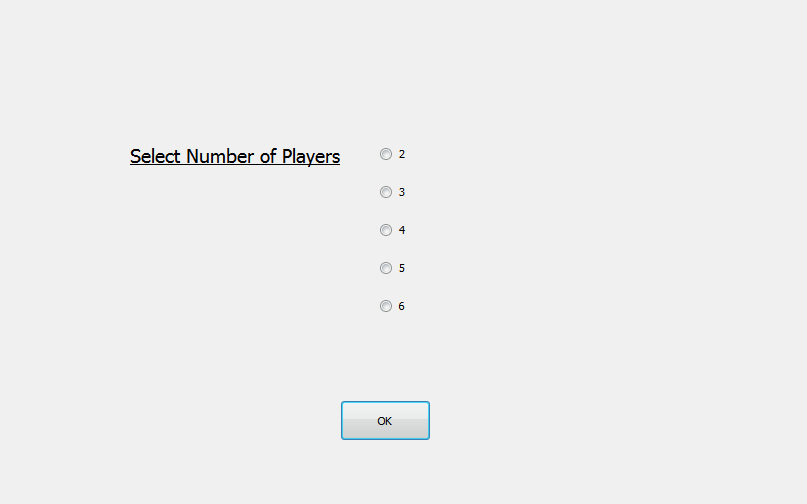
* 1. **Interconnection of Windows**

**3.0. SCREENSHOTS**

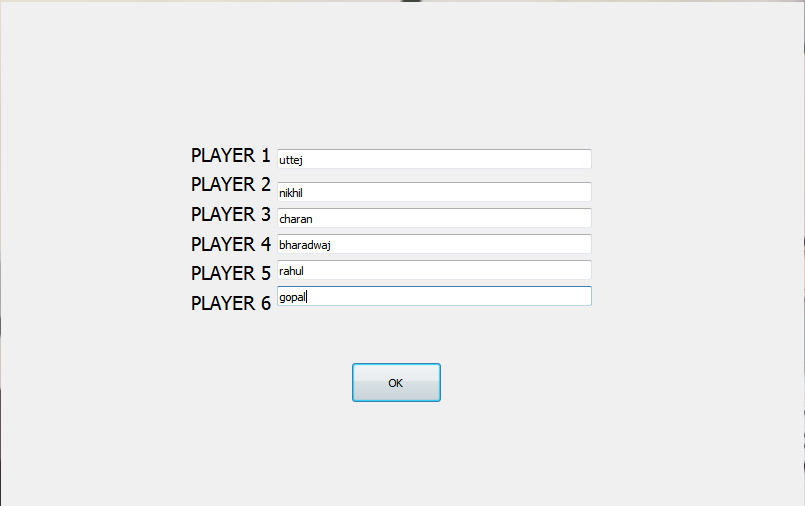
Some of the output windows are shown in this section.



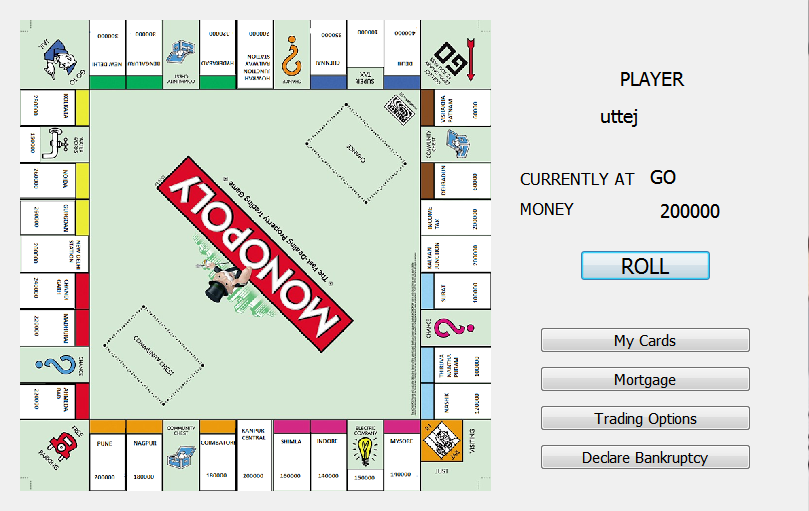
**2.1 Home Page**



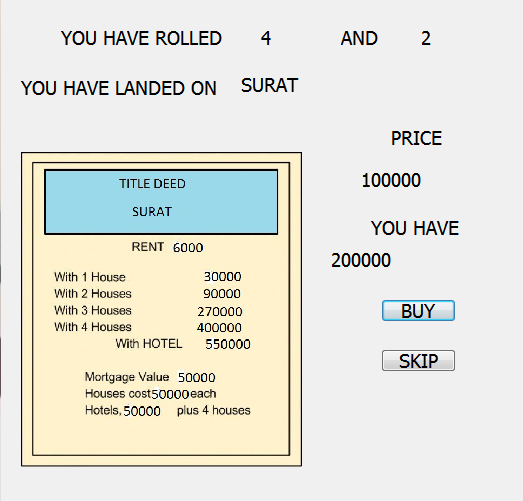
**2.2 Number of Players Intake**



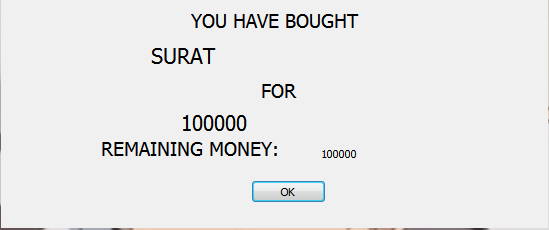
**2.3 Player Details**



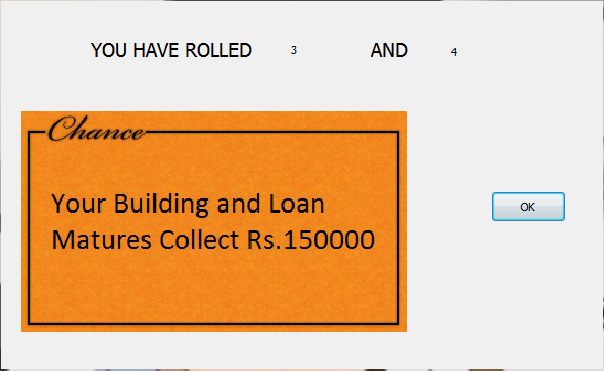
**2.4 Main Board**



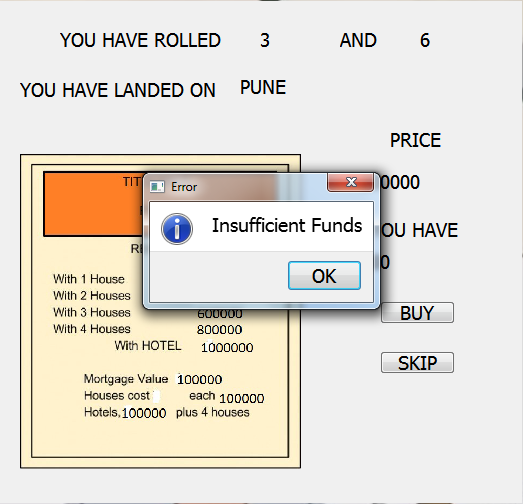
**2.5 Rolled onto Free Space**



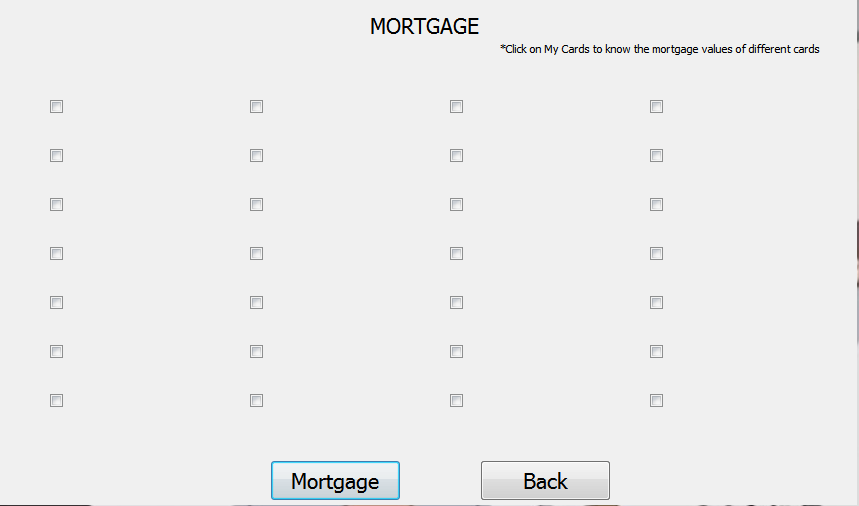
**2.6 Purchase Confirmation**



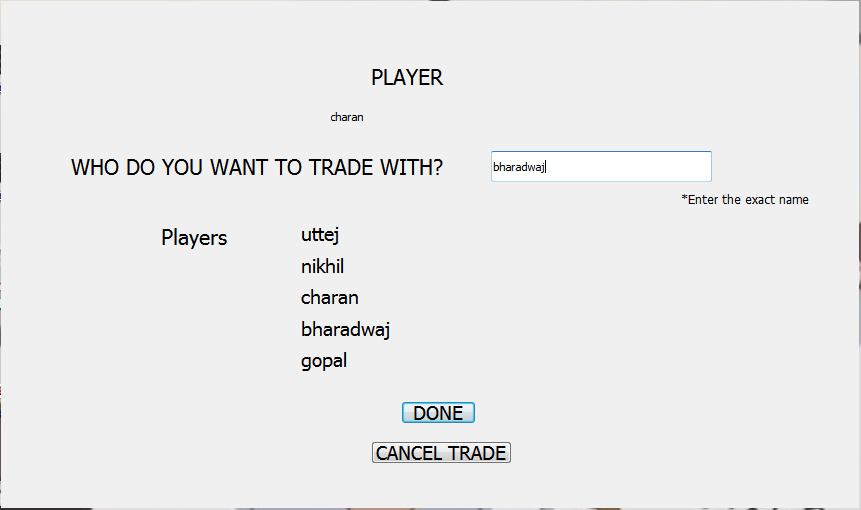
**2.7 Rolled onto Chance**



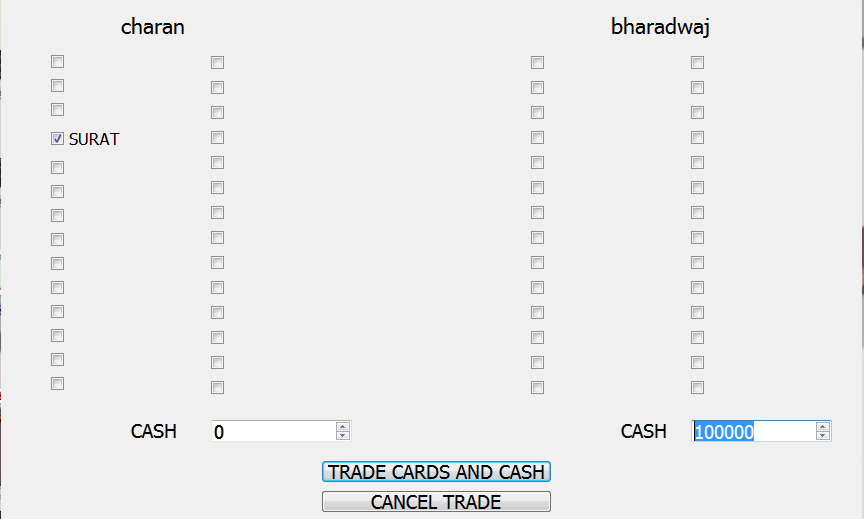
**2.8 Rejection of Purchase due to Insufficient Cash**



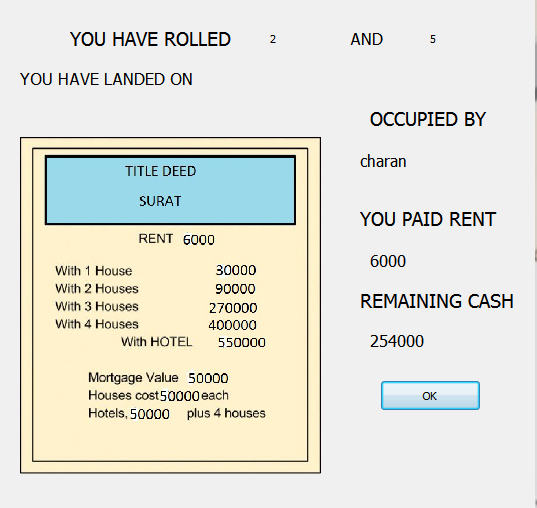
**2.9 Mortgage window**



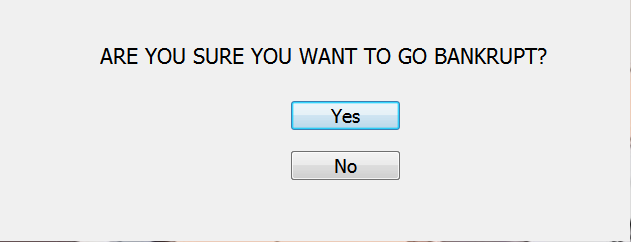
**2.10 Trade Preliminaries**



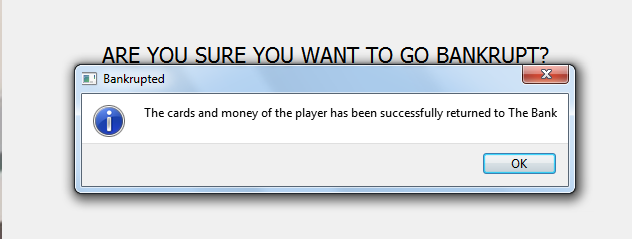
**2.11 Trade Window**



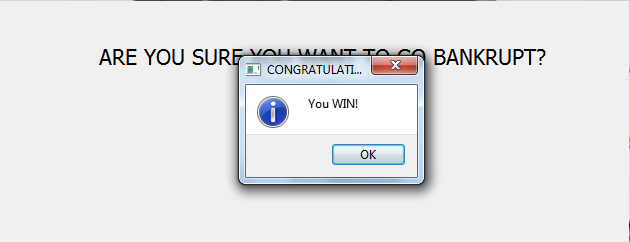
**2.12 Paying Rent**



**2.13 Bankrupt Confirmation**



**2.14 Bankrupted**



**2.15 Winner Declaration**

1. **REFERENCES**
2. https://en.wikipedia.org/wiki/Qt\_Creator
3. <http://doc.qt.io/qt-4.8/qmake-precompiledheaders.html>
4. <http://doc.qt.io/qt-5/designer-ui-file-format.html>
5. <http://richard_wilding.tripod.com/monorules.htm>