MONTREALOPOLY

Design Document
Version 1.8

TEAM MEMBERS

Name	Student ID

Revision History

Date	Version	Description	Author
9-10-03	1.0	Document start	Stefan Thibeault
10/12/03	1.1	Added comments describing what needs to be done for each section.	Robert
19/10/2003	1.2	Added section 4.2 “View” module	Robert
20/10/2003	1.3	Integrated section 4.1 “Model” module	Robert
21/10/2003	1.4	Integrated the remainder of section 4.1	Robert
22/10/2003	1.5	More Corrections	Robert, Zhi, Stefan
22/10/2003	1.6	Added section 3.3 – Dynamic Models	Robert
23/10/2003	1.7	Integrated sections 2, 3.2 and 4.3	Robert, Zhi, Stefan
23/10/2003	1.8	Added Appendix A	Robert, Zhi, Stefan

Table of Contents

Design Document

1. Introduction

The primary goal of this project is to develop the Montrealopoly game. This game is based on the original Monopoly© game, with some modifications. Some of the original rules of the game have been changed. Further, the game board and cell names have been modified to a Montreal-based theme.

The purpose of this design document is to provide all details of the Architectural Design (AD), Module Interface Design (MID), and Internal Module Design (IMD) for the Montrealopoly game. The AD part focuses on the high-level project decomposition, the MID focuses on the software interfaces between the high level modules, and the IMD focuses on the low level description of the implementation classes and all their attributes and methods.

1.1 Purpose

The purpose of this document is to present the design of the Montrealopoly game, which is in partial fulfillment of the requirements of COMP 354. It will provide details on the architectural design, the software interface design, and the internal module design. The architectural design will describe the software architecture that was chosen for the game and a class diagram of this architecture. The software interface design will have screen shots of the graphical user interface and how the users interact with the game. Finally, the internal module design will describe in detail the different modules through the use of class diagrams. This document is intended primarily for the members of Team Redmond and the project coordinator, Dr. Joey Paquet, as it will serve as a basis for the final phase of the project.

1.2 Scope

This document is intended to provide detailed design specifications of the Montrealopoly game that will be used as a basis for the implementation phase. The software architecture that will be used will be explained in great detail in order for the implementation team to actually create a game based on the software architecture described in this document. Furthermore, screen shots of the game will provide a basis for the actual graphical user interface used in the game. The class diagrams from the Internal Module Design section will be converted to Visual Basic code using Rational Rose. This code will then be used by the implementation team to develop the game.

1.3 Definitions and Abbreviations

1.3.1 Definitions

Term	Definition
Model View Controller	The architecture used in the Montrealoploy game, consisting of three individual components, the model, view and controller, which can be developed separately.
JFL Card	JustForLaughs Card. A card containing instructions, which must be followed by the player if they land on the JFL cell.
JFL Deck	A deck containing 20 JFL cards.

1.3.2 Abbreviations

Abbreviation	Term
OS	Operating System
FIFO	First In First Out
MVC	Model View Controller
JFL	Just For Laughs
GOJ	Get out of Jail
GOJFC	Get out of Jail Free Card
GUI	Graphical User Interface
AI	Artificial Intelligence
API	Application Programming Interface
VB	Visual Basic

1.4 References

- Pressman, Roger S. Software Engineering: A Practitioner's Approach. 5th ed. Toronto: McGraw-Hill, 2001.
- Cristobal Baray ,"The Model-View-Controller (MVC) Design Pattern." http://www.cs.indiana.edu/~cbaray/projects/mvc.html (Current October 9, 2003)
- Paula Bo Lu, "COMP 354 Tutorial 2" http://www.cs.concordia.ca/~grad/blu/comp354-2.ppt (Current October 9, 2003)
- Mark D'Aoust , “Coordinate User Interface Development with VB.NET and the MVC Pattern”, http://www.devx.com/dotnet/Article/10186/1954?pf=true (Current October 19, 2003)
- Dr. Volker Haarslev, "COMP 472 History and State of the Art in AI" http://www.cs.concordia.ca/%7Eteaching/comp472/01_intro.pdf (Current October 20, 2003)

1.5 Overview

The remainder of this document is divided into three major parts: Architectural Design description, Software Interface Design description and Internal Model Design description. The architectural design consists of the architecture rationale, software architecture diagram and system topology. The software interface design consists of the system and module interface diagrams and the dynamic models of system interface, which shows how the module interfaces are to be used. The internal module design describes each module of the system along with its class diagram and all the classes that it has. The last section contains the team member's log sheets.

There is an important topic that does not fit in any section of the report template. The topic is the general description of the flow of the game. The main scenarios of the game and the logical and chronological relationship among them can be found in this topic. It will give the implementation group a clear idea about the game itself, rather than the technical rules. We elected to add this as an appendix (see appendix A).

We have also added another section in the Appendices (see Appendix B), which includes some test scenarios that can be used as guidelines to the testing phase of the project.

2. Architectural Design

2.1 Rationale

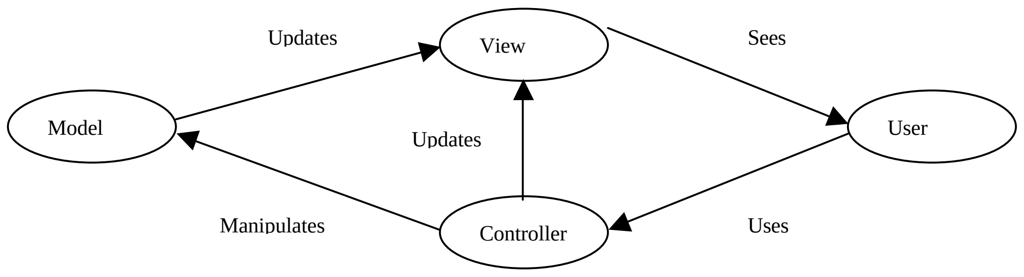
The architecture chosen for the Montrealopoly game is the Model View Controller model (MVC). The MVC architecture is made up of 3 separate components, called the model, view and controller

The model is the core of the game where the games state and player data are stored and manipulated. All of the computations that are performed during the game are done in this component as well as all data that needs to be processed. Moreover, any request to change the game's state is also handled by the model. Whenever there are changes made to the model, the model updates the view.

The view is the graphical user interface (GUI) of the game and displays the data from the model. Whenever the model changes, the view responds to those changes by updating itself. The view also gets updated by the controller, as the controller performs simple data validations on user input and updates the view in the form of a pop-up if any errors are detected. Different versions of the view can be developed in order to present the same data in different ways.

The controller is what the players use to interact with the game. All player data is entered via the controller, which it then passes on to the model. The controller also interprets all mouse clicks or game events and it determines which part of the model needs to be manipulated. Basic input validations are also performed in the controller, which updates the view if an error is detected.

The MVC architecture allows the three components to be developed separately from one another and they can be done in parallel. At anytime, either of the components can be updated without affecting the other components as long as their application program interface remains the same. With a common API, the 3 models can be seamlessly integrated into one game and different versions of the models can be used in the game. This architecture allows for many different versions of views to be developed, which could be used to make variations of the game based on different themes.



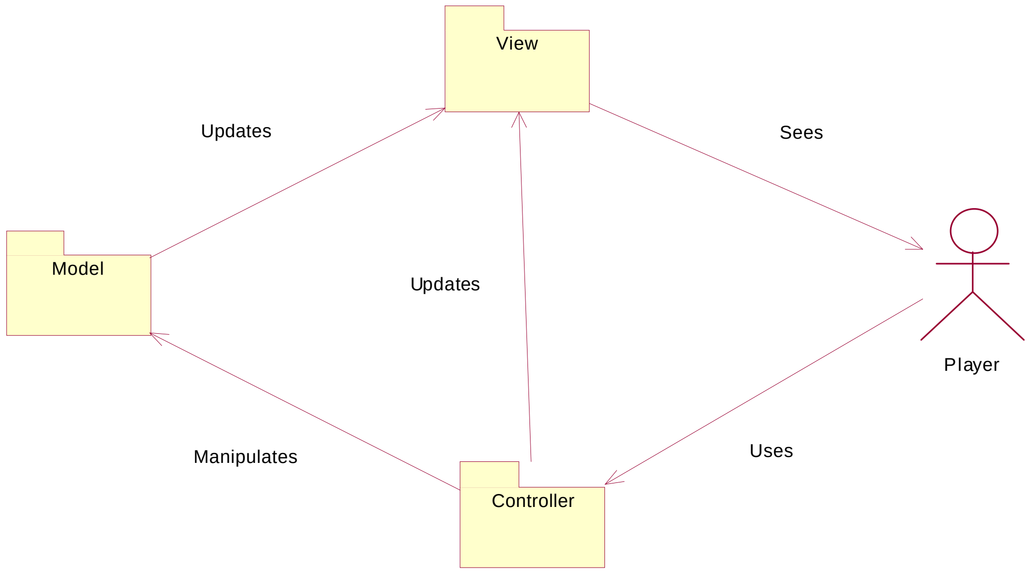
The MVC Architecture

The main reason why Team Redmond chose the MVC model for the Montrealopoly game is that it allows the GUI to be separated from the core application. This flexibility allows the game's core to be developed by the implementation team and the GUI to be designed by the design team. Both teams could then work in parallel and easily collaborate on any changes that need to be made. By using this architecture, enhancements to the game's GUI could be made without having to modify any of the game's core functionality.

Furthermore, several models could be developed in parallel with each model containing different game functionalities. Each model could be then thoroughly tested and debugged. Once the quality assurance on the model has been completed, its functionalities could be integrated into future models.

2.2 Software Architecture Diagram

The three components of the MVC model that are used in the Montrealopoly game are shown below in the form of a high-level class diagram. The players interact directly with the view and the controller, while the model accepts requests from the controller and updates the view for the players.



High-level Class Diagram of the MVC Model

The game's controller accepts input from the player via the mouse and the keyboard through the use of Visual Basic's events. Simple validations, like input checking, are done by the controller and will inform the user of any faulty data entered with a message box. Moreover, if a player tries to perform an illegal action, the controller will also inform the user via a message box. Any valid action performed by the player will be accepted by the controller, which will then relay the information to the model.

The model, when instructed by the controller, will perform the task that was requested. All of the game's data is stored in the model, which also contains the core functionalities of the game. Whenever there is a change made to the model's data, the model will update the view to reflect the change.

The view consists of the GUI, which is the visual aspect of the game. With the GUI, players can monitor the game flow, see where their opponents are on the board and decide their next course of action. The view also provides visual cues to let the player know where they can click on the screen and where they can enter data. This input is handled by the controller, which then gets manipulated by the model and in turn, updates the view.

2.3 System Topology

The Montrealopoly game is to be developed for a standalone environment and each installation of the game is to be run on a single computer. This will allow for easy distribution of the game, as all three components of the MVC model will be integrated into one executable. Moreover, the game does not require any third party software to run, or an Internet connection.

3. Software Interface Design

3.1 System Interface Diagrams

The only system level interface in the Montrealopoly game is the user interface, as the game does not employ any software or hardware interfaces. The user interface is the game's GUI, which allows the players to interact with the game. Through the game's GUI, the players will be able to see the different states of the game and make decisions based on what they see.

3.1.1 User Interface

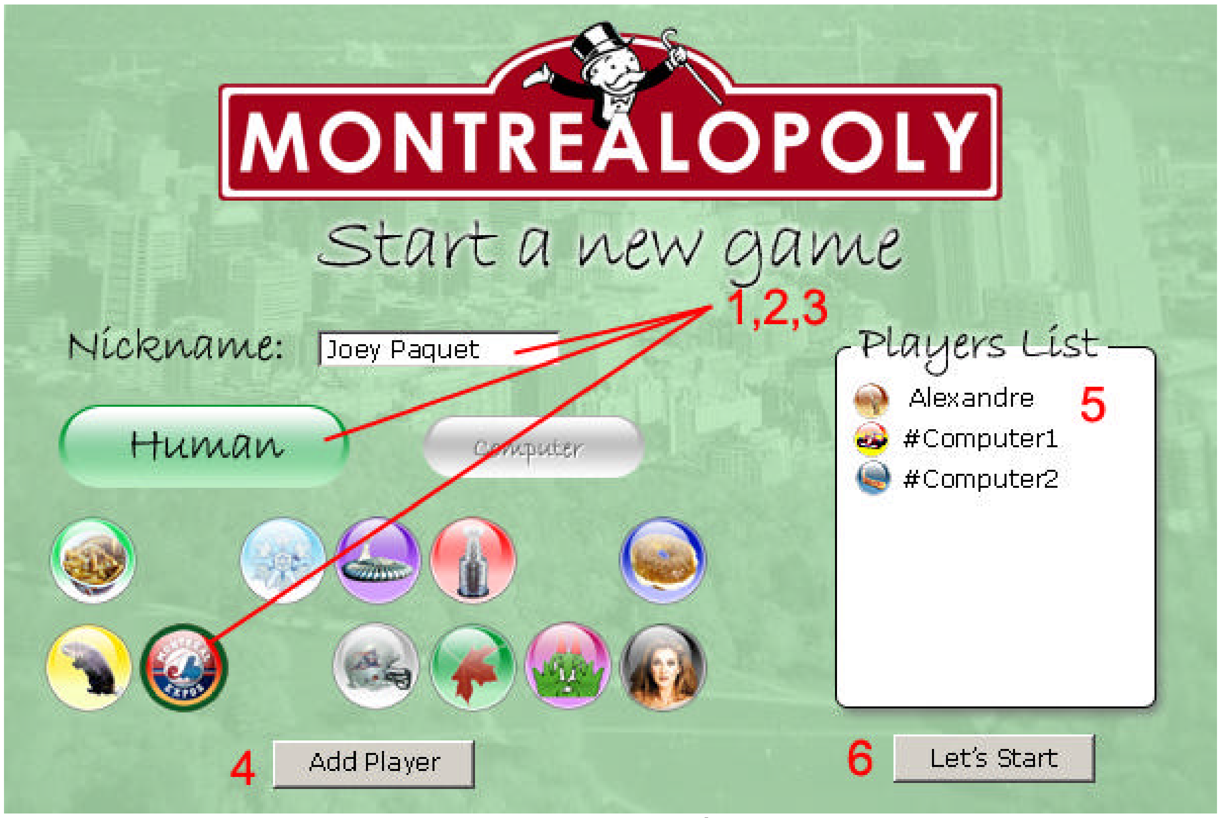
In the Montrealopoly game, the user interfaces are the links between the users and computer. In order to complete a task with ease, an efficient and intuitive user interface must be developed. In doing so, several points should be taken into consideration:

- Complete information: player should find all elements they need quickly to make a decision.
- Error avoidance: avoid utilization of ambiguous terms or concepts. If an error occurs, an error message should display what kind of error has occurred, why it has occurred, and how to solve the error.
- Usability: components should be explicit enough for users in order for them to intuitively know which actions they have to perform.

The graphical user interfaces (GUI) that the players will interact with are described below.

3.1.1.1 Game Start

Whenever the Montrealopoly game is launched, the Game Start user interface appears. It allows the players to enter their name, which token they want to use and whether the player will be a computer or a human player.



View 1.1 – Start Board

User interactions

1: User enters a nickname for the player by filling in the Nickname textbox.

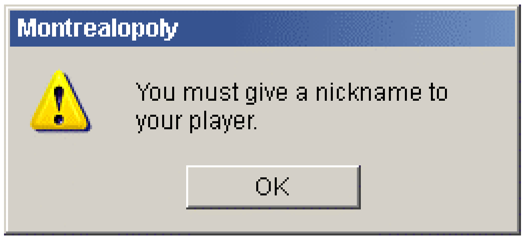
2: User chooses what kind of player he wants to assign to the current player by pressing one of the two buttons, “Human” or “Computer”. The “Human” button is selected by default.

3: User chooses a token by clicking on one of the available tokens. The selected token will then be highlighted.

Note: Steps 1, 2, 3 can be done in any order.

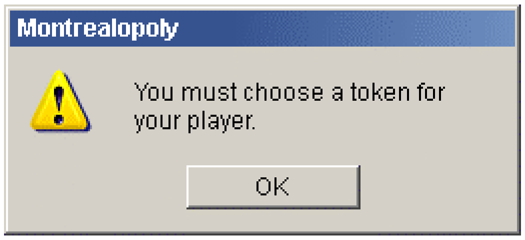
4: When steps 1-3 are completed, the user can click on the “Add Player” button. This action will add the player in the Players List panel (step 5).

If the user clicks on the button “Add Player” and the steps 1, 2,3 have not been completed, then an error message box will appear with the reason why this error occurred.


Montrealopoly

You must give a nickname to your player.

View 1.2 – Error message box: no nickname

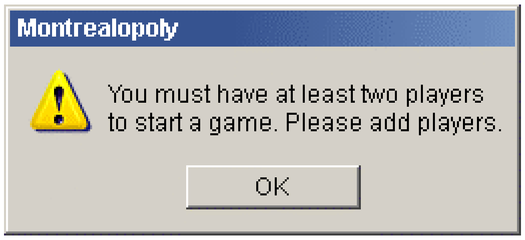

Montrealopoly

You must choose a token for your player.

View 1.3 – Error message box: no token

5: When the “Add Player” button is clicked, the player name, along with his token are displayed in the Players List panel. A “#” symbol next to the players name indicates that the player is a computer player.

6: When at least 2 players have been entered, game play can start. To start the game, the “Let’s Start” button must be clicked. If less than two players have been entered when the “Let’s Start” button has been clicked, an error message box will prompt the user for more players.

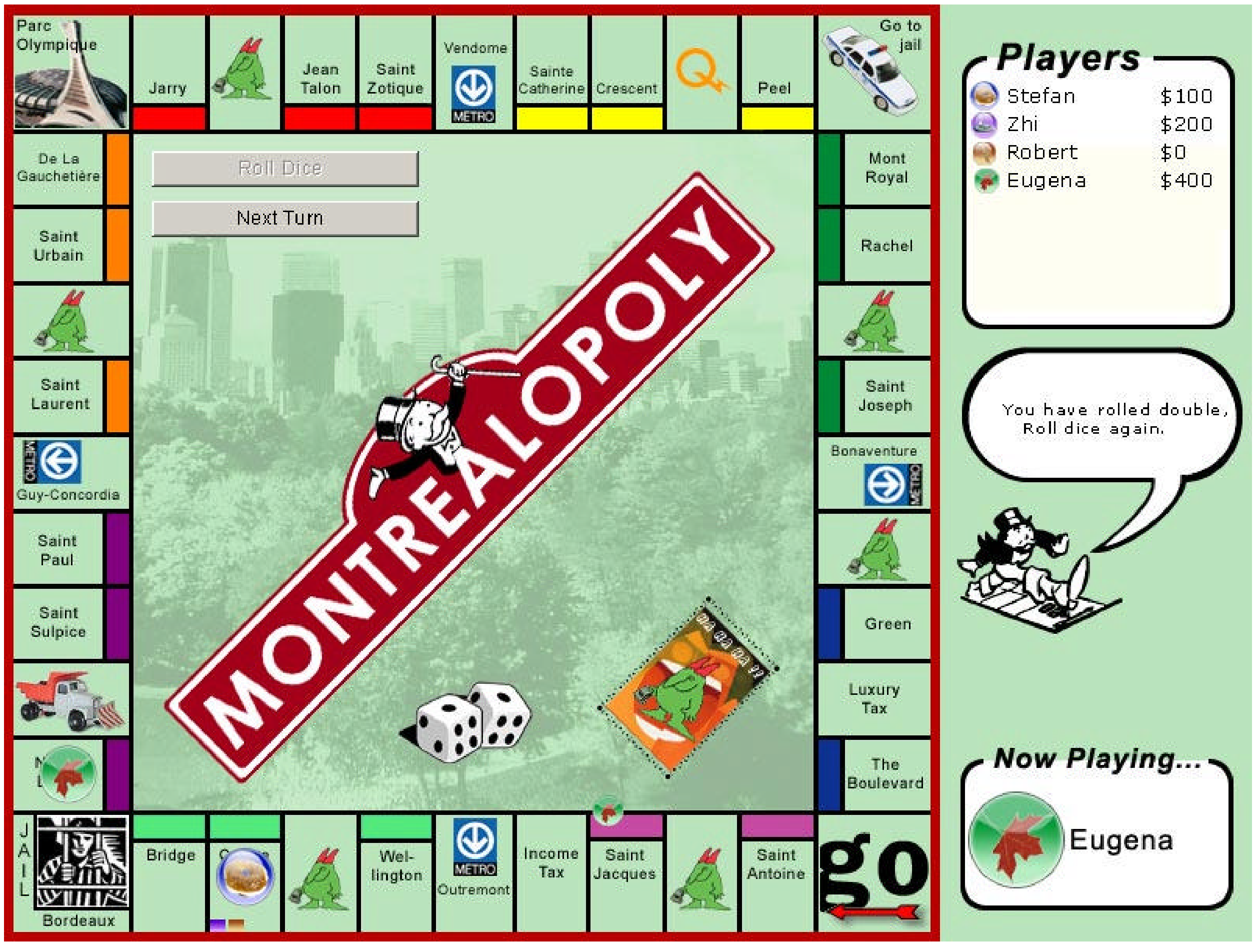

Montrealopoly

You must have at least two players to start a game. Please add players.

View 1.4 – Error message box: not enough players

3.1.1.2 Main View

Once enough players have been entered and the “Lets Start” button has been clicked on, the game can begin. The game’s main interface is loaded, which is the game’s main playing area, which the players interact with. The components with which the players can interact with are described below.



Stefan	$100
Zhi	$200
Robert	$0
Eugena	$400

You have rolled double, Roll dice again.

Now Playing... Eugena

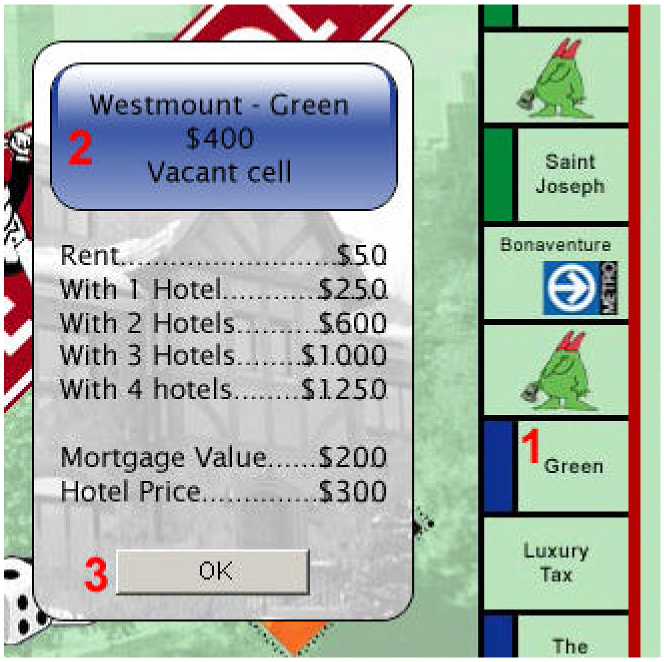
View 1.2 – Main board

a. Cells

The board game is composed of 40 cells, with 5 types of different cells. At any time, players can click on a cell in order to view detailed information about the cell. The different types of cells are described below.

i. Street cell

The board is composed of 22 street cells grouped in 8 colors. Each street cell contains a color, and its name. When a user clicks on a cell, the title deed of this cell will appear, displaying the current status of this street. The title deed will be explained in detail in section 3, Pop-up Windows. The street cell also displays who owns the street, whether or not the street has been mortgaged, the amount of hotels on the street.


Westmount - Green
$400
Vacant cell

Rent	$50
With 1 Hotel	$250
With 2 Hotels	$600
With 3 Hotels	$1.000
With 4 hotels	$1250
Mortgage Value	$200
Hotel Price	$300

View 1.2.1 – User clicks on a cell

<u>User interactions</u>

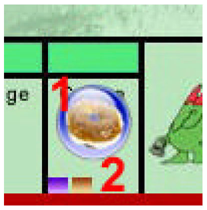
1: User clicks on a cell. This action provides the user information about the street cell that was clicked.

2: Title deed card corresponding to the cell appears.

3: The player clicks on the "OK" button when finished viewing the title deed card.

Special street cell cases

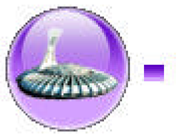
- Several players on the same cell: at times, there can be several players on the same cell, making it difficult to see who is on the cell. To resolve this problem, the token of the current player is displayed and the rest of the tokens are represented by little colored squares.



View 1.2.2 – Several users on the same cell

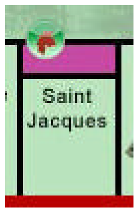
1: The token of the current player on the cell is displayed.

2: The colored squares represent the other users on the same cell.



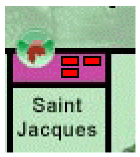
View 1.2.3 – A token and its associated colored square

- Cell owned by a player: when a player is the owner of a street cell, his token is displayed on the top-left corner of the cell.



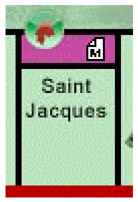
View 1.2.4 – Saint-Jacques street is owned by the player with the Canadian leaf token

- Hotels built on a street: little rectangles represent hotels and they are displayed in the top-right corner of a street cell.



View 1.2.5 – Saint-Jacques street has 3 hotels build on it

- Cell mortgaged: when a property is mortgaged, a mortgage icon is displayed on the top-right corner of the cell.



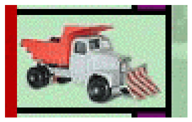
View 1.2.6 – Saint-Jacques street with mortgage icon

ii. Metro/Utility cell

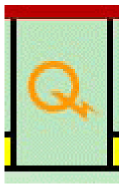
The board contains 4 metro cells and 2 utility cells, which can be purchased like street cells. These cells are similar to the street cells, except that no hotels can be built on them. Clicking on a metro/utility cell will bring up the cell's title deed.

METRO
Guy-Concordia

View 1.2.7 – A metro cell: Guy-Concordia



View 1.2.8 – An utility cell: snow dump



View 1.2.10 – An utility cell: Hydro Quebec

Metros/utilities cells behave in similar ways to the street cells.

- if a token is displayed in the top-left corner of the cell, then this cell is owned
- if a mortgage icon is displayed, then the cell is mortgaged

iii. Corner cell

The board contains 4 corner cells, which no player can own. The Go cell adds $200 to the player's account each time they land on or pass this cell. The Olympic Parc and Bordeaux Jail cells are resting cells where nothing happens when the player lands on them. The Go To Jail cell sends the player directly to jail. Clicking on the cell will display information about the cell. The Bordeaux Jail cell will display who currently is in jail and who is just visiting, when clicked on.

View 1.2.11 – Go Cell (bottom-right corner)

JAIL
Bordeaux

View 1.2.12 –Bordeaux Jail Cell (bottom-left corner)

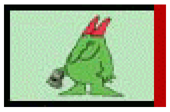
Parc
Olympique

View 1.2.13 – Olympic Park Cell (top-left corner)

View 1.2.14 – Go to jail Cell (top-right corner)

iv. JFL Cell (Just For Laughs Cell)

The board contains 6 JFL cells. When a user lands on a JFL cell, a JFL Card will be automatically displayed via a pop-up window. The user will then have to do as instructed as per the JFL card.

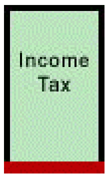


View 1.2.15 – JFL Cell

v. Tax Cell

There are 2 tax cells on the board, a Luxury Tax and Income Tax.

Clicking on one of theses cells will display the tax that they must pay to the bank.

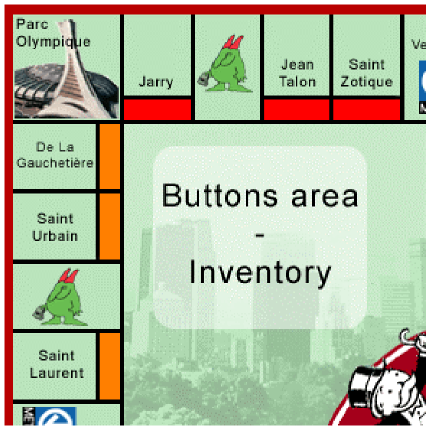


View 1.2.16 – Income Tax cell

View 1.2.17 – Luxury Tax cell

b. Buttons and inventory area of the board game

Most of the actions during a game will be performed in this area. It is the central point of the game as users will be able to perform several actions in this area (such as paying a fine if they are in jail, etc). The buttons area of the board game is located in the top-left corner of the application.



View 1.2.17 – Buttons area placement

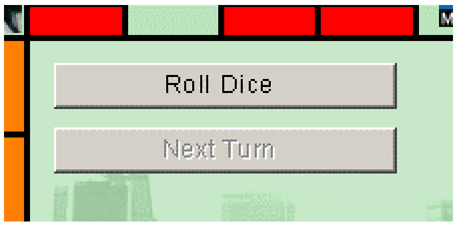
Depending on the state of the game, different buttons will appear here.

i. Roll dice button

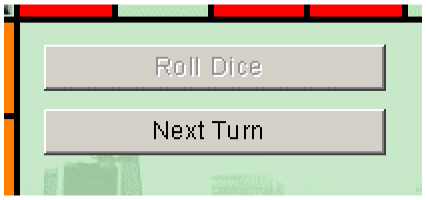
At the beginning of each turn, the player will have the option to roll the dice by clicking on the “Roll Dice” button. The player token will then be automatically moved the amount of steps the player rolled. Then the button is disabled except if the player has rolled double, then the button is still enabled and the player can roll the dice a second.

ii. Next Turn button

Once the player is finished his turn, he will be able to pass the play onto the next player by clicking on this button. This button will be disabled if the player is in debt to another player and will only be enabled once the debt has been paid.



View 1.2.18 – Roll Dice button



View 1.2.19 – Next Turn button

iii. Pay Fine button

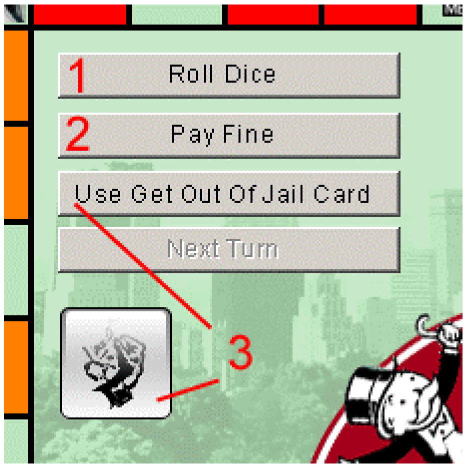
This button allows a player to pay the fine to get out of jail and will appear only when a player is in jail. After being in jail for three turns, the "Roll Dice" button will be disabled and the player will have only 2 solutions to get out of jail by clicking on "Pay Fine" or "Use Get out of Jail Card" (if he has the card).

iv. "Use Get out of Jail Card" button

If a player is in jail and has previously picked up a "Get out of Jail Card" from the JFL deck (and has yet to use it), then this button will appear on the buttons area. This button will allow the player to get out of jail for free. Once the card is used, the card will be returned to the JFL deck. This button will appear if the player is in jail and has the "Get out of Jail" card.

v. Inventory area

When a player has a "Get out of jail card", an icon will be displayed, reminding the player that he has a "Get out of Jail" Card. The icon is for display purposes only, and no actions can be performed by clicking on the icon.



View1.2.20 – Button / Inventory Area

<u>User interactions</u>

When a user is in jail, he will have these different possibilities to get out of jail.

1: If the user has not spent more than 3 turns in jail, he can roll dice and get out of jail if doubles were rolled.

2: The user can pay the fine to get out of jail.

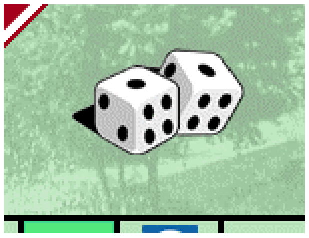
3: If the user has a "Get Out Of Jail Card" he can then click on the "Use Get Out Of Jail Card" button, if he has one.

vi. Declare bankruptcy

When a user does not have enough money in bank to continue playing the game, then the "Declare Bankruptcy" button appears. Pressing this button will terminate the game for the current player.

c. Dice area

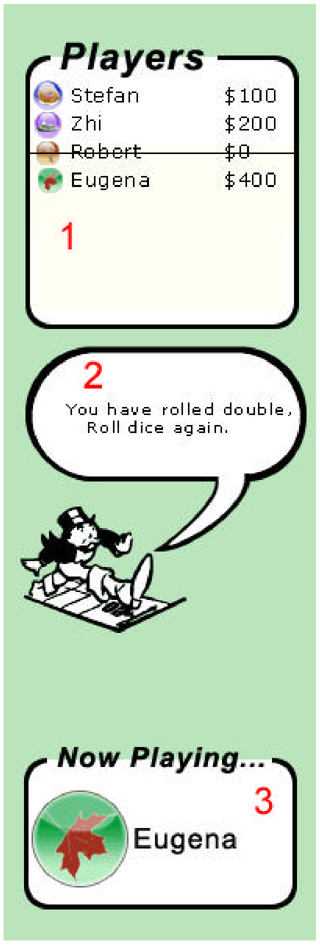
This is the place where users see what they have rolled and is for display purposes only.



View 1.2.21 – Dice area

d. Side bar

The sidebar displays the players in the game, the current player and a message area.



You have rolled double, Roll dice again.

View 1.2.22 – Side Bar

Components of the side bar

1. Player List

All the players currently in the game are displayed here along with their token. A computer player will have a '#' symbol in front of their name. A player who has declared bankruptcy will have their name striked-out.

2. Message area

The message area will prompt the current player with the current state of the game. The message area display the amount that he rolled, prompt the player that he is in debt, etc.

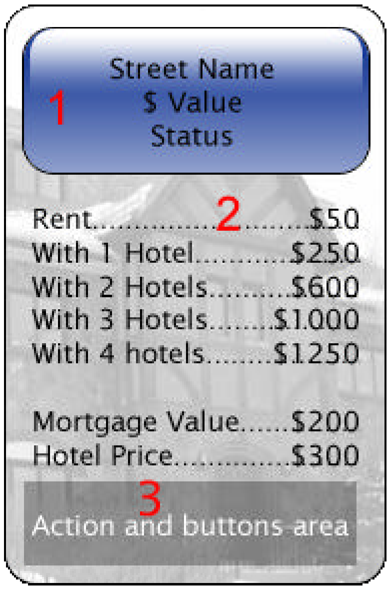
3. Current player

The current player along with their token will be displayed here.

3.1.1.3 Pop-Up Windows

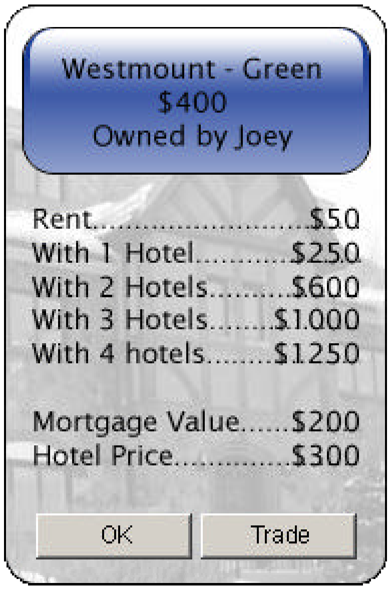
a. Title deed card

Clicking on a street/metro/utility cell will produce a pop-up with the property's title deed card. Depending on the status of the game and who clicked on the title deed, different buttons will appear on it. All title deeds have the following basic look:


Street Name
$ Value
Status
1
Rent.................2........$50
With 1 Hotel............$250
With 2 Hotels..........$600
With 3 Hotels........$1.000
With 4 hotels........$1250
Mortgage Value......$200
Hotel Price..............$300
3
Action and buttons area

View 1.3.1 – A title deed card

1: this area shows the name of the street, its price and its status (vacant or owned by)
2: this area displays the different costs: the rent, with or without hotels, the mortgage value and the price to build a hotel
3: this area shows several buttons, depending on the status of street and the player who clicked on the title deed.



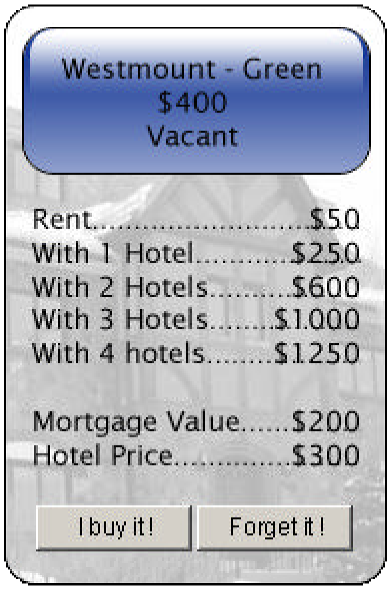
Westmount - Green
$400
Owned by Joey

Rent..........................$50
With 1 Hotel...........$250
With 2 Hotels..........$600
With 3 Hotels........$1.000
With 4 hotels........$1250

Mortgage Value......$200
Hotel Price.............$300

View 1.3.2 – Title deed info

At any time during the player's turn, the player may view a Title Deed Card by clicking on its corresponding cell. The information pertaining to the property will appear and a click on the "OK" button will close the title deed.
If the player clicks on the "Trade" button, then a "Trade Card" will appear (see topic b – Trading Card for further details).



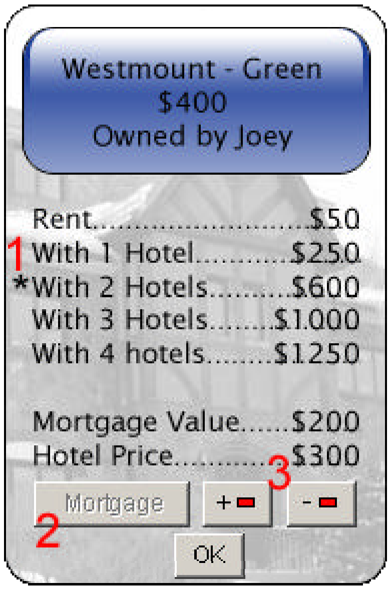
Westmount - Green
$400
Vacant

Rent..........................$50
With 1 Hotel...........$250
With 2 Hotels..........$600
With 3 Hotels........$1.000
With 4 hotels........$1250

Mortgage Value......$200
Hotel Price.............$300

View 1.3.3 – Title deed vacant lot

Landing on a vacant property will produce a Title Deed Card with two buttons that displaying two options. One button will say "I want it!" and by clicking on it will allow the player to purchase the property. The other button will say "Forget it!" and the player can choose not to purchase the property.



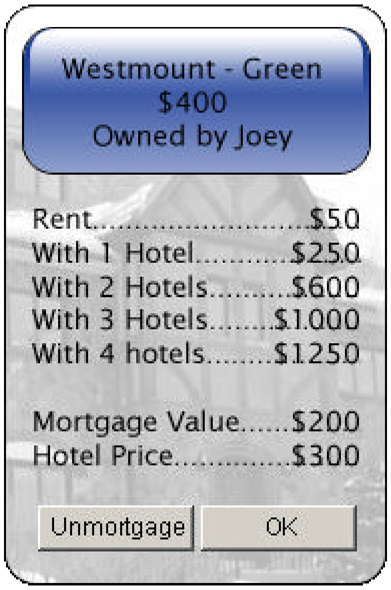
Westmount - Green
$400
Owned by Joey

Rent..........................$50
With 1 Hotel...........$250
*With 2 Hotels..........$600
With 3 Hotels........$1.000
With 4 hotels........$1250

Mortgage Value......$200
Hotel Price.............$300

View 1.3.4 – Title deed when you owns this street

Once the player is the owner of a property and clicks on the corresponding cell, the action buttons displayed will allow the player to mortgage / unmortgage / buy / sell



Westmount - Green
$400
Owned by Joey

Rent..........................$50
With 1 Hotel...........$250
With 2 Hotels..........$600
With 3 Hotels........$1.000
With 4 hotels........$1250

Mortgage Value......$200
Hotel Price.............$300

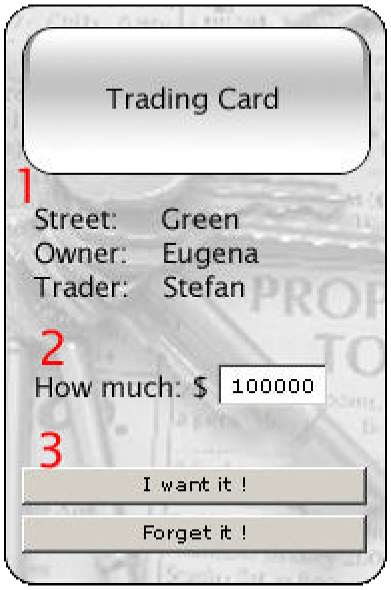
View 1.3.5 – Title deed when a street is mortgaged

If a property is mortgaged, the Title Deed Card will have action buttons to "Unmortgage" the property and the "OK" button to close the Title Deed Card.

hotels. The details of the buttons are as follows:
2: “Mortgage”: mortgage the property and will appear if the property is unimproved.
2: “Unmortgage”: pay back the mortgage to the bank.
3: “+Hotel”: to build a hotel and appears only if there are sufficient funds.
3: “-Hotel”: to sell a hotel back to the bank.
“OK” to close the Title Deed Card
1: (*)Number of hotels built on the street

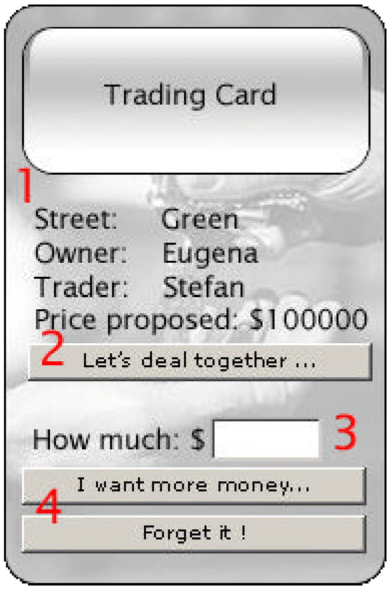
b. Trading card

If a player wants to purchase a property from another player, he must click on the property that he wants, then click on the trade button. The trading card appears; it shows information about the trade (1), a text box to enter the amount that the player wants to pay for it (2). By clicking on the “I Want It!” button will initiate the negotiations for the property and clicking on “Forget It!” cancels the offer (3).



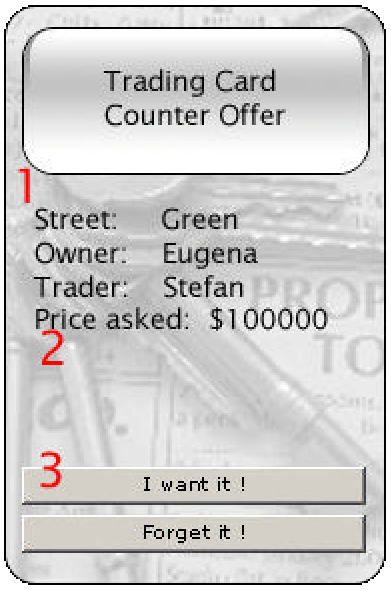
View 1.3.6 – Trading card interface

The owner of the property receives a pop-up with the proposal (1), after which the owner has several options. If the owner is interested in the trade he clicks on the “Let’s Deal Together” button (2). On the other hand, if the owner wants more money, he enters the amount desired in the “How much” box (3) and then clicks on the “I Want More Money…” button (4), which is a counter-offer. The offer can also be rejected by clicking on the “Forget about it!” button (4).



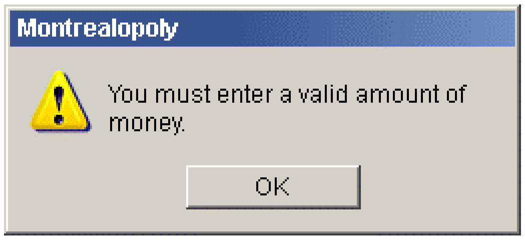
View 1.3.7 – Trading card interface when replying

If a counter-offer is made the buyer will receive a pop-up displaying the counter-offer proposal (1) and the price asked (2). The buyer then can accept or reject the offer (3).



View 1.3.8 – Trading card interface when counter-offering

If a proposal is accepted by the owner of a street, the street changes owner and a message is displayed in the message area (see view 1.2.22). If the trade proposal has been rejected, then another message is displayed in the message area (see view 1.2.22). If the players enter invalid amounts of money, an error message will prompt the user to enter a valid amount.



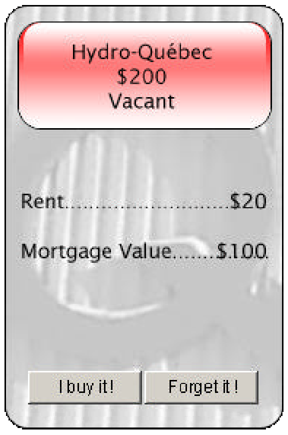
Montrealopoly

You must enter a valid amount of money.

View 1.3.9 – Error message box: not a valid amount of money

c. Metro / Utility card

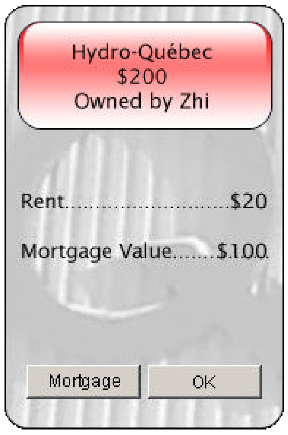
Like streets, Metro and Utility cells have properties like rent and mortgage. When a user lands on a Metro or Utility cell for the first time, he has the possibility to buy it. Otherwise, when a user clicks on a metro or utility cell, a card will appear and show information about the cell. An example is shown with the Hydro-Quebec utility below.



Hydro-Québec
$200
Vacant

Rent..........................$20

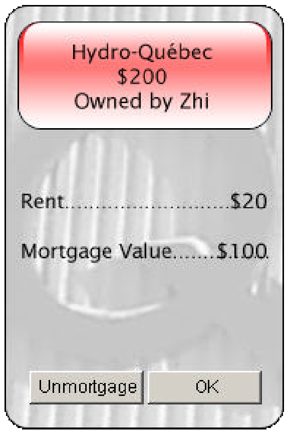
Mortgage Value.......$100



Hydro-Québec
$200
Owned by Zhi

Rent..........................$20

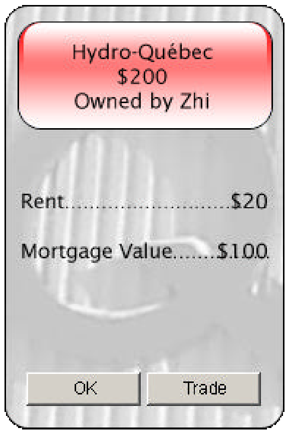
Mortgage Value.......$100



Hydro-Québec
$200
Owned by Zhi

Rent..........................$20

Mortgage Value.......$100



Hydro-Québec
$200
Owned by Zhi

Rent..........................$20

Mortgage Value.......$100

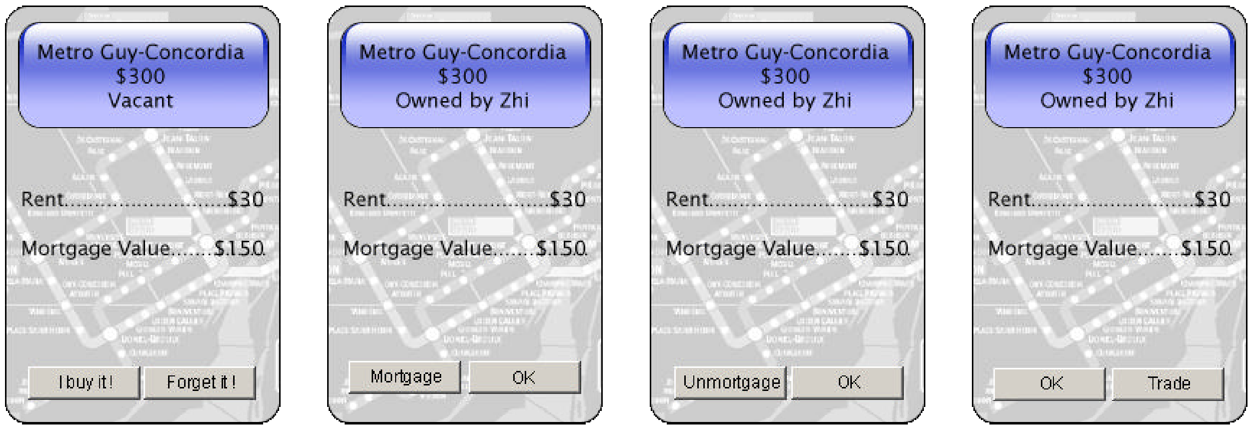
View 1.3.10-13

When a user lands on the Hydro-Québec cell and this cell is vacant, this card appears when the cell is clicked on and the player has the possibility to buy this property by clicking the "I buy it" button.
If user clicks on the "Forget it" button, then the property is still vacant.

When an owner clicks on their own utility, they have the option to mortgage it or simply view the information about the utility.

If the Hydro-Québec utility has been mortgaged, the owner can unmortgage the property if they have enough funds to do so by clicking on the "Unmortgage" button when they click on their own cell.

If a player clicks on a cell that is owned by another player, they will have to opportunity to initiate a trade for the property by clicking on the "Trade" button.



Metro Guy-Concordia
$300
Vacant
Rent.........................$30
Mortgage Value.......$150

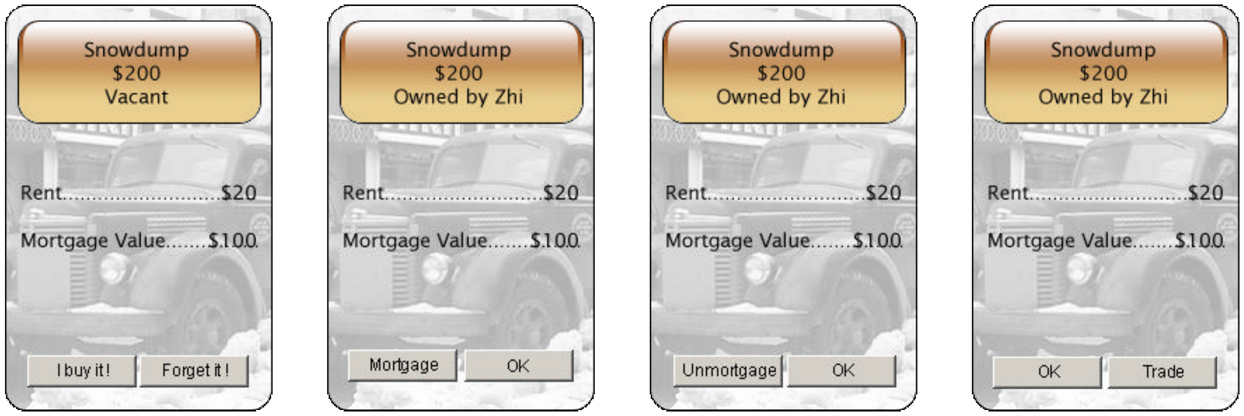
Metro Guy-Concordia
$300
Owned by Zhi
Rent.........................$30
Mortgage Value.......$150

Metro Guy-Concordia
$300
Owned by Zhi
Rent.........................$30
Mortgage Value.......$150

Metro Guy-Concordia
$300
Owned by Zhi
Rent.........................$30
Mortgage Value.......$150

View 1.3.14-17

The metros have the same characteristics as the Hydro-Québec card.



Snowdump
$200
Vacant
Rent.........................$20
Mortgage Value.......$100

Snowdump
$200
Owned by Zhi
Rent.........................$20
Mortgage Value.......$100

Snowdump
$200
Owned by Zhi
Rent.........................$20
Mortgage Value.......$100

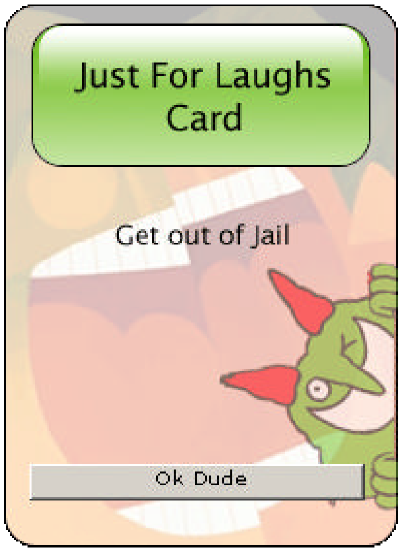
Snowdump
$200
Owned by Zhi
Rent.........................$20
Mortgage Value.......$100

View 1.3.18-21

Same as the Hydro-Québec card.

d. JFL Card

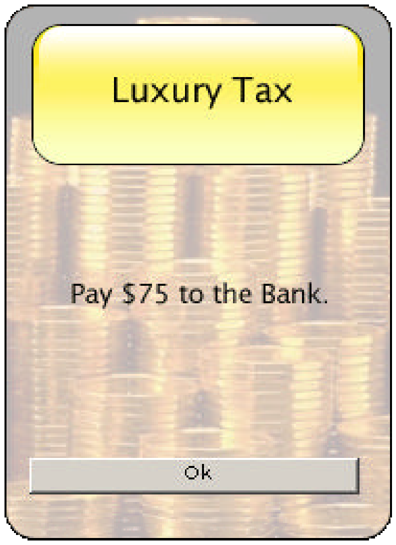
Once a player lands on a JFL cell, a JFL Card will pop-up. A JFL Card will give the player instructions that the player must follow. When the player clicks on the “OK Dude” button, the action based on the instructions will be performed. However, if the card is a “Get Out of Jail” Card, the player gets to keep the card for future use (in that case a special icon will be placed in the inventory area – see view 1.2.20). The only action required from the user is a click on the “OK Dude” button.



View 1.3.22 – Just For Laughs Card

e. Other pop-ups

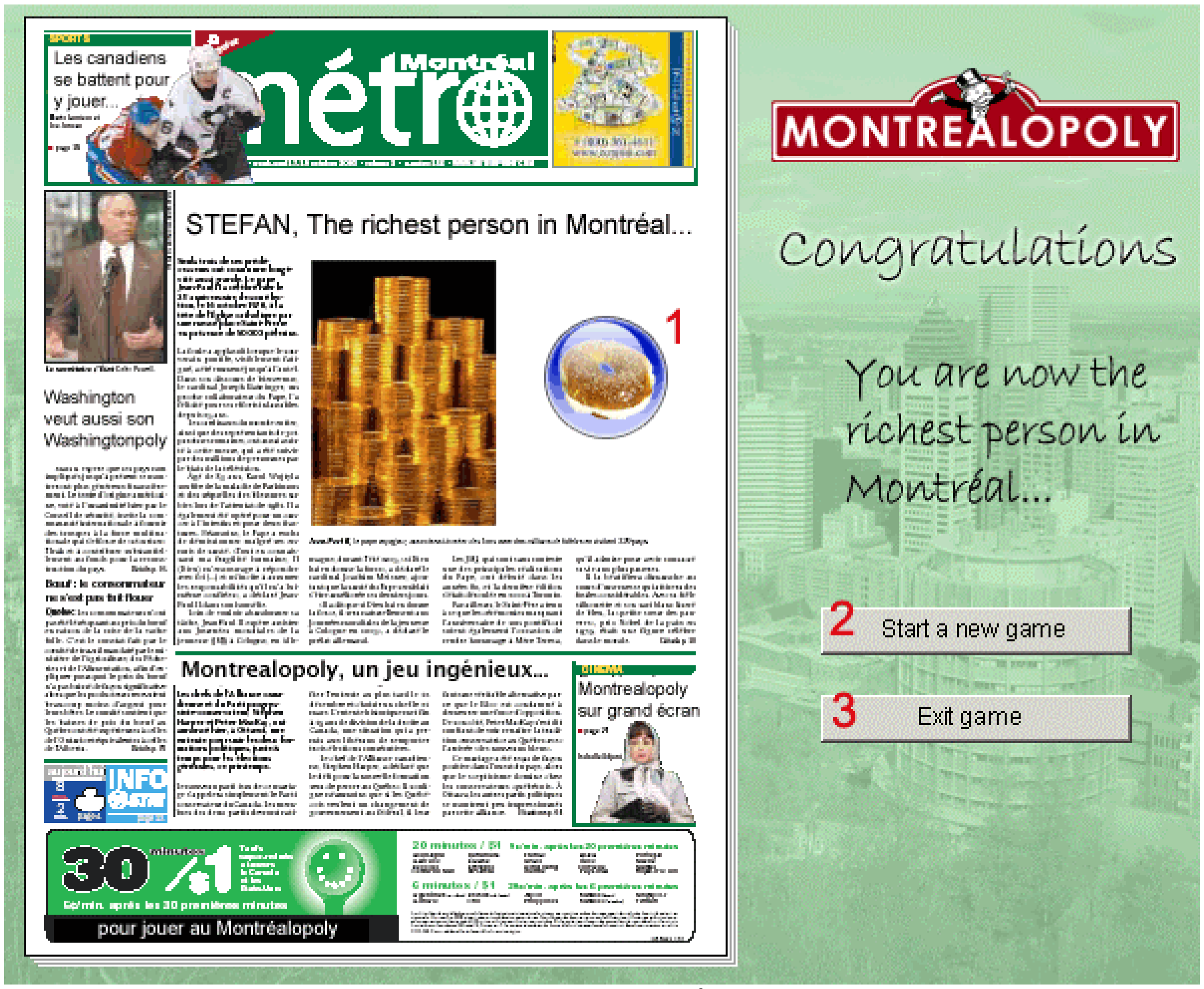
The other pop-ups of the game (Jail, Olympic, Go To Jail, Luxury Tax and Income Tax cells), like the JFL Cards, are only be for information purposes. When a player clicks on one of these cells (see view 1.2 Main Board to see where they are located), a Pop-Up window will appear, displaying information about the cell. User won’t have any interaction with these cards except for the “Ok” button to close them.



View 1.3.23 – Luxury Tax example.

3.1.1.4 Winner interface

When there is only one player left in the game, that player is declared the winner. A pop-up will display the name of the winning player. The winner will then have two options, either to start a new game or to exit the game.

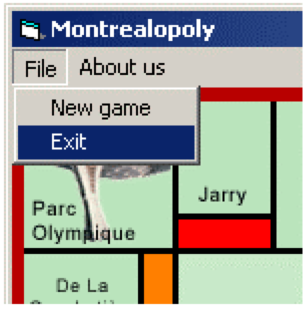

SPORTS
Les canadiens se battent pour y jouer...
Montréal
métro
STEFAN, The richest person in Montréal...
1
Washington veut aussi son Washingtonpoly
Montrealopoly, un jeu ingénieux...
CINÉMA
Montrealopoly sur grand écran
INFO
30 minutes
pour jouer au Montréalopoly
MONTREALOPOLY
Congratulations
You are now the richest person in Montréal...
2
Start a new game
3
Exit game

View 1.4.1 – Winner interface

<u>User interactions</u>

1: No interaction needed. The nickname and the token of the winner is displayed on the front page of the newspaper.
2: If a player wants to play a new game, he can do so by pressing the “Start a new game” button.
3: If a player wants to exit the game, they can do so by clicking “Exit game”.

3.1.1.5 Menu interface



View 1.5.1 – Menu bar

User interactions

By clicking on "File", a sub-menu appears and the player can start a new game by clicking "New game" or can exit the game by clicking on "Exit".

If the users want to have more information about the team which developed this game, they can click on "About us". A pop-up will provide information about game's developers.

3.1.2 Software Interface

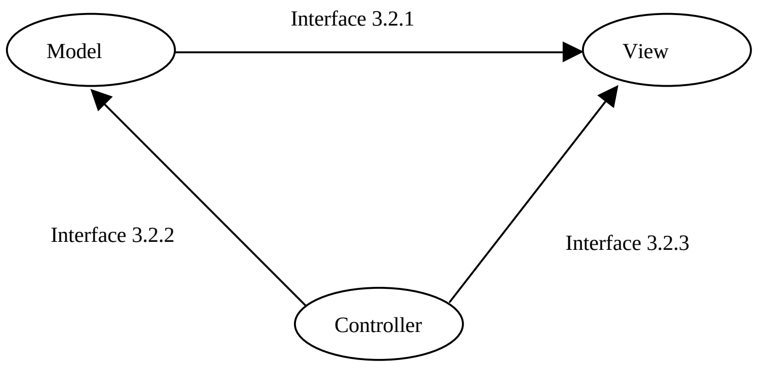
This is not applicable as the Montrealopoly game does not interact with any system software, besides the OS. The Montrealopoly game is designed as a stand-alone system; therefore, there is no interaction (and thereby no interface) between the Montrealopoly system and any other software system.

3.1.3 Hardware Interface

This is not applicable, as the Montrealopoly game does not interact with any specific hardware (other than the hardware of the computer running the software). Therefore, as part of our design, there is no interactions (and thereby no interface) between the Montrealopoly system and any other hardware system.

3.2 Module Interface Diagrams

In the MVC model, there are interfaces between the Model, View and Controller. The methods in one module are accessed by another module via an interface between the two modules, as seen in interfaces 3.2.1 - 3.2.3.



The three interfaces between the MVC model modules.

3.2.1 View Interface

The interface between the model and view is called by the model whenever there is a change in its internal data structures.. The following methods are part of the interface:

3.2.1.1 GameStart

The gameStart class is called whenever a new game is launched.

The following method is available to this interface:

- ShowGameStartWindow(): The game start panel is launched at the start of a new game and is used to gather information about the players in the game.

3.2.1.2 MainWindow

The MainWindow class is the main playing area of the game. It consists of the board image, all the cells on the board, the current position of the player’s tokens, and the list of players in the game.

The following methods are available to this interface:

- showStreetCell(): This cell gets refreshed whenever the state of the cell changes. The cell displays its owner, the current player’s token, colored cells representing the other players on the cell, the number of hotels and whether the street is mortgaged.
- showMetroCell(): This cell gets refreshed whenever the state of the cell changes. The cell displays its owner, the current player’s token, colored cells representing the other players on the cell, and whether the metro is mortgaged.
- showUtilityCell(): This cell gets refreshed whenever the state of the cell changes. The cell displays its owner, the current player’s token, colored cells representing the other players on the cell, and whether the utility is mortgaged.
- showInfoMessage(): All messages regarding the games state are displayed in the message window on the board .
- showErrorMessage(): Input errors detected by the controller are displayed with this method.

The following methods all act in a similar manner whenever the state of the cell changes. The cell displays the current player’s token and colored boxes representing the other players on the cell.

- showGoCell()
- showGoToJailCell()
- showOlympicParkCell()
- showJailCell()
- showIncomeTaxCell()
- showLuxuryCell()
- showJFLCell()

3.2.1.3 JFL CardWindow

Whenever a player lands on a JFL cell, a pop-up with a JFL card appears.

The following method is available to this interface:

- ShowJFLCard(): The JFL pop-up displays a JFL card with the instructions that the player must follow.

3.2.1.4 TradeWindow

Whenever a player wants to initiate a trade, this window pops-up to perform the trade.

The following method is available to this interface:

- ShowTradeWindow(): The pop-up window prompts the user for the amount of money that the player wants to pay for the property. The owner of the property then receives a pop-up where he can then accept the offer, refuse it or ask for more money. The initiator of the trade will then get a pop-up with the result of the offer, which he can accept or decline.

3.2.1.5 CellInfoWindow

Whenever a player clicks on a cell, a pop-up displays the information about the particular cell.

The following method is available to this interface:

- showCellInfo(): Displays the information about the Jail, Olympic park, Go To Jail, go, JFL, Income Tax and Luxury Tax cells when clicked on.
- showStreetInfo(): Displays the owner of the street, the rent, mortgage value, cost of the hotels and buttons area.
- showMetroInfo(): Displays the owner of the metro, the rent, mortgage value and buttons area.
- showUtilityInfo():Displays the owner of the utility, the rent, mortgage value and buttons area.

3.2.1.6 GameEnd

The gameEnd class is called whenever there is only one player left in the game, which is also the games winner.

The following method is available to this interface:

- ShowWinner(): The winner of the game is displayed along with the option to start and new game or exit the application.

3.2.2 Model Interface

The interface between the controller and model is called whenever the controller receives input from the player. The following methods are part of the interface:

3.2.2.1 Player

The player class contains all the data of each player in the game. The methods called in this class by the interface are to query the player's status or to perform an action that the player requested.

The following methods are available to this interface:

- IsBankrupt(): Called whenever the next turn button is clicked on to determine if the next player is still in the game.
- declareBankrupcy(): A player declares bankruptcy when he cannot raise enough funds to pay all his debts to another player. Declaring bankruptcy removes the player from the game and any assets are transferred to the player he is in debt to.
- buyProperty(): If the property the player lands on is vacant, the player can purchase the property by clicking on the buy button on the title deed card.
- offerTrade(): If a player is interested in acquiring a property from another player, he may initiate trade negotiations by clicking on the trade button.
- pay50GOJ(): If a player is in jail and wants to get out of jail by paying the $50 fine, he may do so by clicking on the pay fine button.
- useGOJFC(): If a player is in jail and has a get out of jail for free card, he may use it by clicking on the get out of jail for free card.

3.2.2.2 Board

The board class contains the methods which control the flow of the game. Methods are called whenever a new game is started and when control of the game needs to be passed on to the next player to start his turn.

The following methods are available to this interface:
addPlayer(): Whenever the add player button is clicked on, the player is entered into the game.
shufflePlayers(): When all the players have been entered into the game, all the players are randomly shuffled inorder to set the order of play.
playTurn(): Rolls the dice and gets value of the dice rolled.
rollDice(): At the start of a player's turn, he may roll the dice to advance on the board by clicking on the roll dice button.
endTurn(): When a player has completed his turn, play is passed on to the next player by clicking on the next turn button.
getCurrentPlayer(): Gets the player object in order to call other methods on the object.
getUtility(): Gets the utility object in order to call other methods on the object.
getStreet(): Gets the street object in order to call other methods on the object.
getMetro(): Gets the metro object in order to call other methods on the object.

3.2.2.3 Property

Methods from the property class are called whenever a player clicks on a street, metro or utility cell on the board.

The following methods are available to this interface:
getOwners(): Gets the owner of the property.
mortgage(): Mortgages the property if the owner clicks on the mortgage button.
unmortgage(): Unmortgages the property if the owner clicks on the unmortgage button and has sufficient funds.

3.2.2.4 Jail

The jail class contains a method to show information about the cell when clicked on.

The following method is available to this interface:
showInfo(): Displays a pop-up with information with which players are in jail and which players are just visiting.

3.2.2.5 Utility

The utility class contains a method to show information about the cell when clicked on.

The following method is available to this interface:
showInfo(): Displays a pop-up with information about the utility and along with who owns it.

3.2.2.6 Street

The street class contains methods that allows the player to build and sell hotels and displays information about the street when the cell is clicked on.

The following methods are available to this interface:
buildHotel(): Builds a hotel on the property whenever the build hotel button is clicked on. The player must have sufficient funds and own the whole district to do so.
sellHotel(): Sells the hotels back to the bank at half the purchase price.
showinfo(): Displays a pop-up with information about the street and along with who owns it.

3.2.2.7 Metro

The metro class contains a method to show information about the cell when clicked on.

The following method is available to this interface:
showInfo(): Displays a pop-up with information about the metro and along with who owns it.

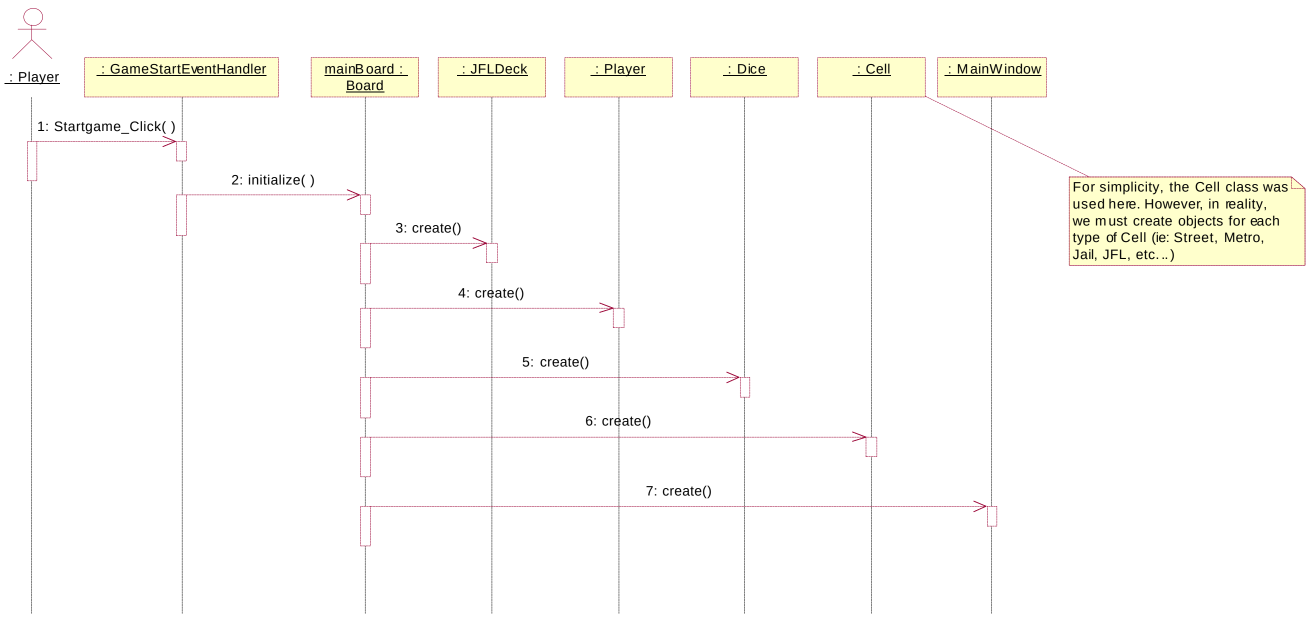
3.2.3 Controller Interface

The controller accepts input from the player and performs simple validations on it. Through the use of Visual Basic’s event handler, any errors that are detected update the view by calling the showErrorMessage method in the view. This method informs the player of the error by displaying a pop-up.

3.3 Dynamic Models of System Interface

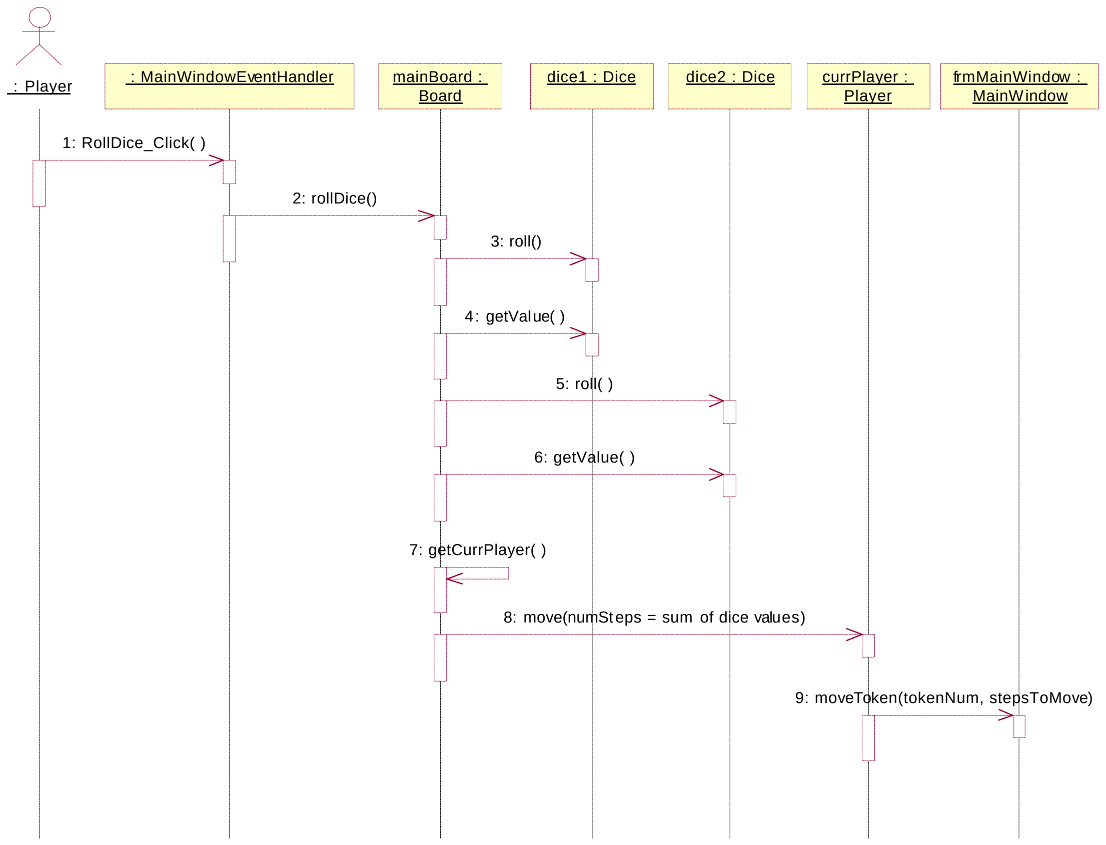
In order to better portray the interactions between the system modules, we have chosen some scenarios, or major functionalities of the system and will explain and depict them using sequence diagrams. We have elected to use sequence diagrams because they depict the interaction between the classes (or objects) of the system and also show the sequence of calls (or messages) that occur.

3.3.1 Start Game Scenario



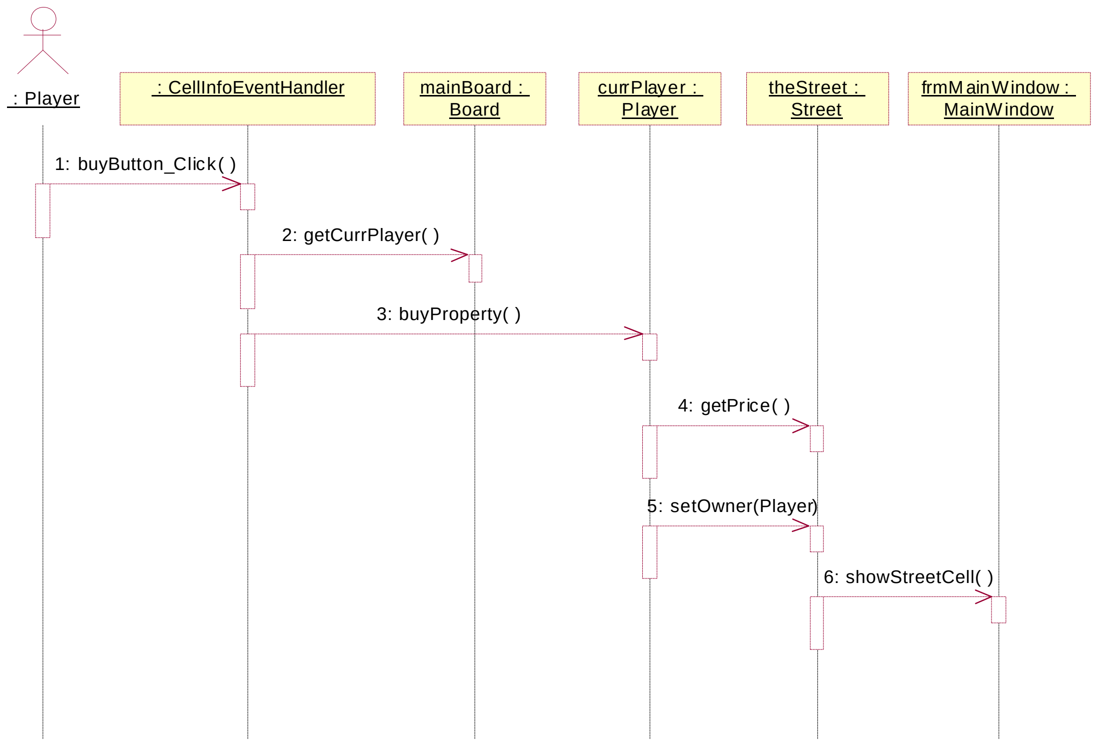
3.3.2 Roll Dice Scenario

The following scenario depicts the actions that occur when a user clicks on the Roll Dice button. First, the event handler (in the Controller) is called upon to handle the click event. Next, the controller calls the rollDice() in the Board class (in the Model). The Model then processes the dice roll and updates itself. Finally, the player's token has to be moved, so a message is passed to the MainWindow form (in the View) to move the token. This is a typical example of the interaction between the three modules of the system.



3.3.3 Buy Property Scenario

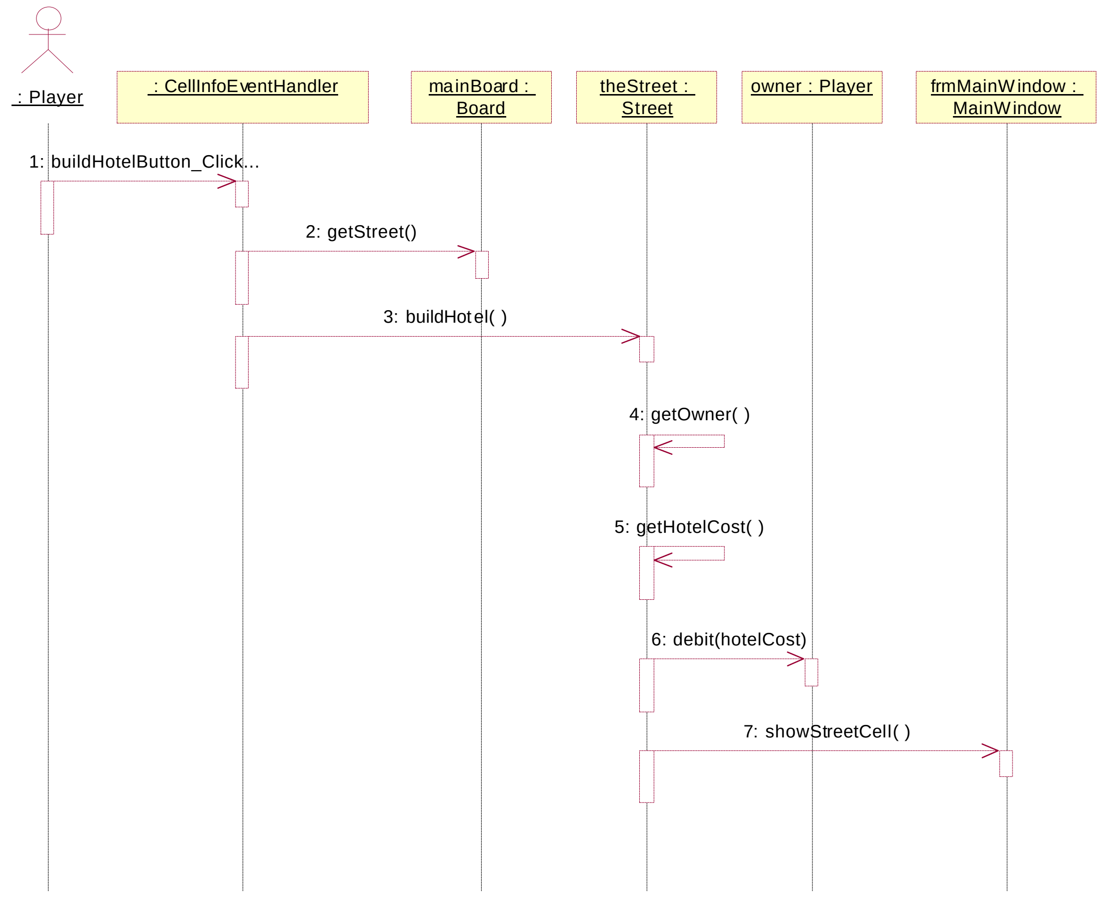
Similarly, one can easily describe the sequence of actions that occur when the player buys a property. First, the user clicks on the BuyIt button. This event is handled by the Controller. The event handler then calls the getCurrPlayer() method in class Board (in Model) to retrieve the current Player object. Then, it will call the buyProperty() method on this object. From this point on, the Model takes over, updating its data structures to keep track of the new property owner. Finally, the model will send a message to the View, to update it's cell display on the board, to identify the new owner.



3.3.4 Build Hotel Scenario

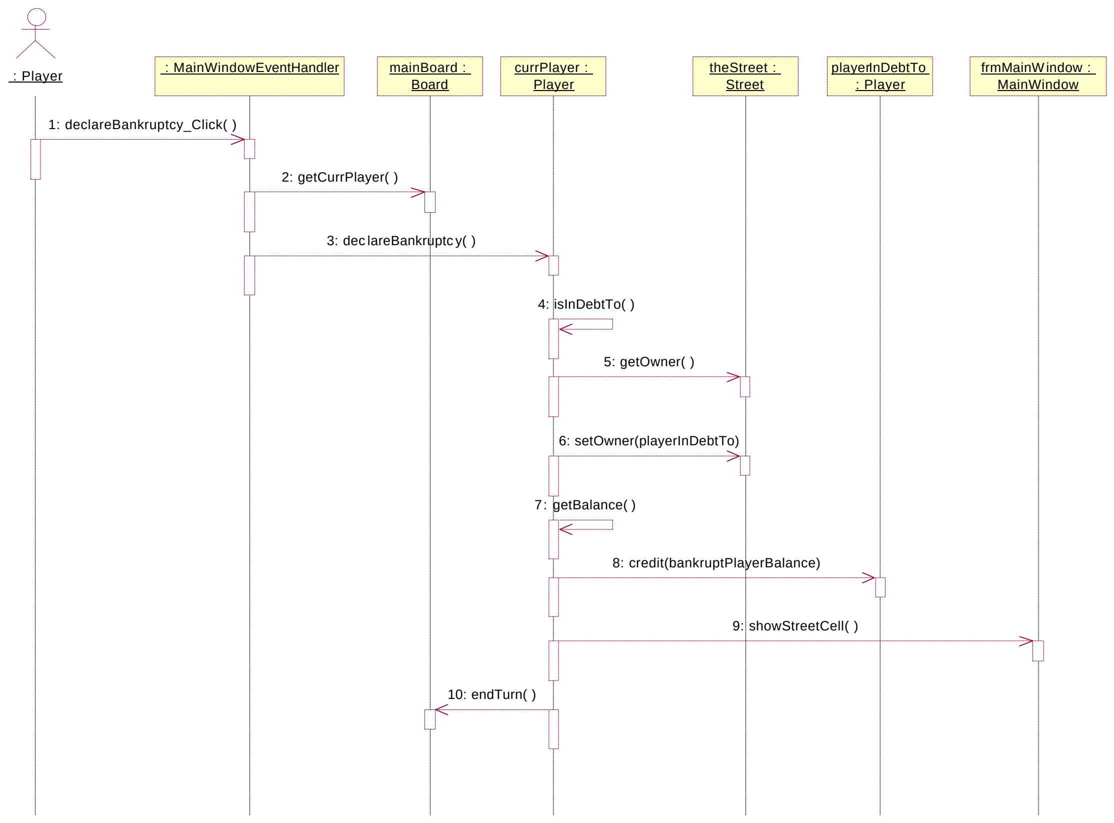
In this scenario, the player builds a hotel on a property he owns by first accessing the CellInfoWindow and then clicking on the Build Hotel button. This event is handled by the Controller, which first calls the getStreet() method in the Board class to get the street object and then calls the buildHotel() method on that Street object. From this point, the Model takes over, and updates its internal data structures. Finally, the model then updates the View, by calling the method showStreetCell(), which will update the cell display on the board, with the new number of hotels.

Similarly, the Sell Hotel scenario can be depicted using approximately the same sequence diagram as shown below.



3.3.5 Declare Bankruptcy Scenario

When a user decides to declare bankruptcy, the following sequence of actions occur. First, the player clicks on the declareBankruptcy button. This event is handled by the controller. The event handler first calls the getCurrPlayer() method on the Board class (in the Model) to get the current Player object. Then, it will call the declareBankruptcy() method on that Player object. From this point, the Model takes over, updating it's internal data structures accordingly. This includes finding out which player the current player is in debt to, and transferring the assets and balance to that player. Finally, it will update the view on the MainWindowView, by updating the cell displays on the board (since the owners may have changed).



4. Internal Module Design

As explained before, the system is divided into three modules, the Model, the View and the Controller. The modules interact together using their respective interfaces, which have been described in detail in section 3.2. Now, we turn our attention to each module, separately. In this section, we will describe the detailed design of the three modules. First, a textual description of the classes will be given. This will be followed by a class diagram, which will describe the relationships between the classes. Subsequently, a detailed description of each class, its methods and attributes will be given. Finally, an important feature, Artificial Intelligence, will be discussed separately in a sub-section.

Before proceeding with the detailed internal module design, it is important to identify and discuss some issues relate to this design. Visual Basic (VB) is the software development tool of choice for our project. This programming language was chosen because it is remarkably easy to develop a graphical user interface (GUI) quickly and efficiently. This will allow us to better deal with the strict time restrictions that this project must meet.

As the design was in progress, an important question was raised concerning VB and inheritance. In our design, we rely on inheritance to solve certain problems like the different types of cells. It is important to note that we tested Visual Basic, and determined that it does support inheritance. We also tested several scenarios with regards to this design and were able to properly implement them using Visual Basic. This was a clear indication that the design was going in the right direction.

It is also quite interesting to note that the CASE tool of choice for this project, Rational Rose, supports VB integration. Rational Rose allows you to use VB-specific data types in a class diagram, and even generate source code for those classes. After the initial design, we were able to fine-tune the Rose software to make it generate VB code. This source code can then be modified by our implementation team; at that point we can, at the click of a button, update the Rose model from the modified source code. This will be greatly helpful in accelerating the implementation phase.

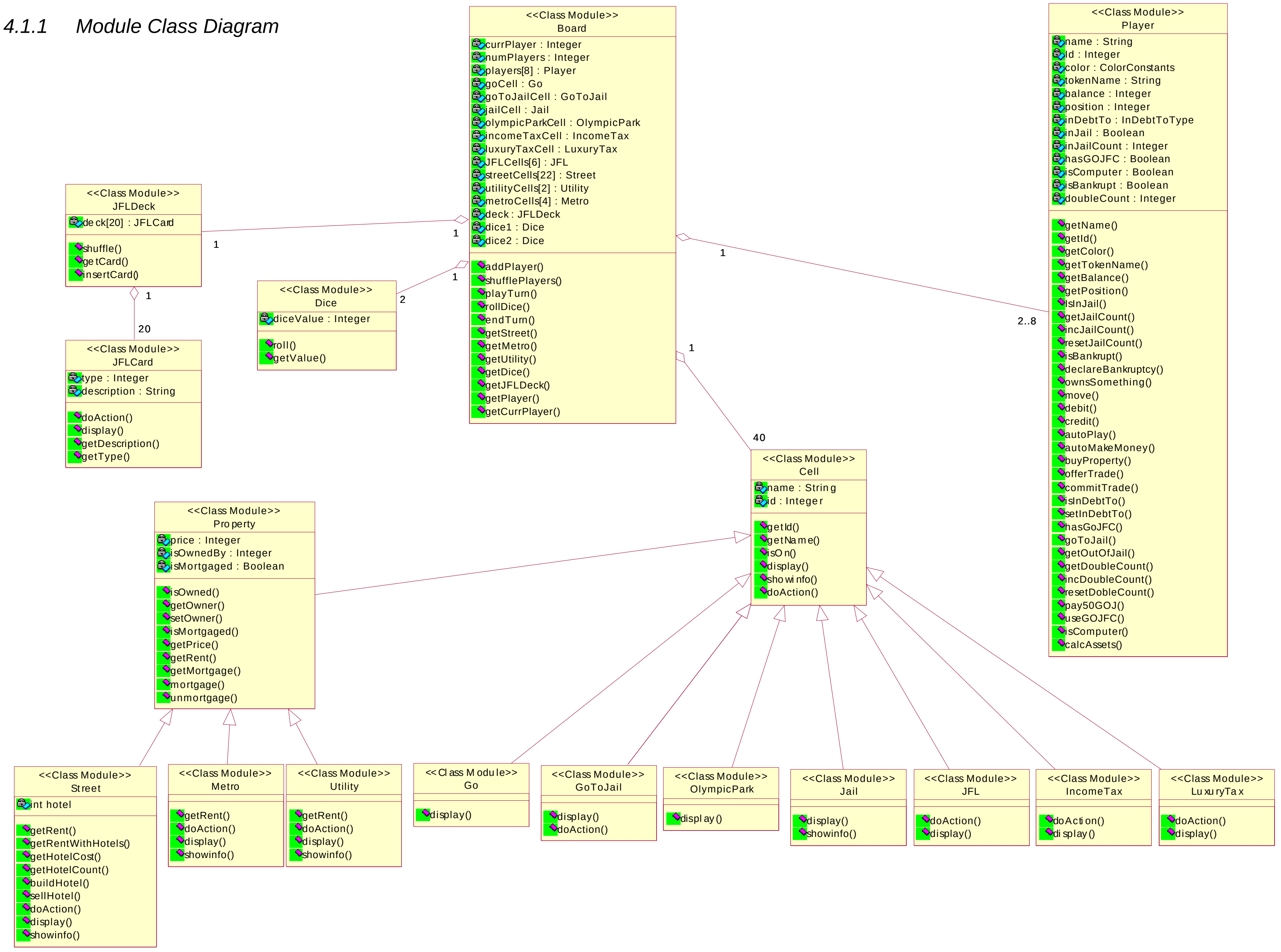
4.1 Module <Model>

The most important module in the Montrealopoly game design is the model module. This module is used to represent the state and actions that occur in the game. This includes player states, cell information and the actions that must occur in the game.

The model consists of several classes, which are all contained within a container calls called Board. The Board class is composed of 1-8 Player objects, 40 Cell objects, 2 dice objects and a JFLDeck object. The JFLDeck object, in turn, is composed of 20 JFLCard objects. Each class contains the attributes and methods necessary to meet its requirements.

The following sections will describe the classes, their relationships and will provide a detailed description of each method listed below.

4.1.1 Module Class Diagram



4.1.2 Class <Board>

Class Name	Board			
Inherits from	None			
Description	The biggest container of the game. All object in a game will be in the board.			
Attributes	Visibility	Data type	Name	Description
	Private	Player	currPlayer	Current player id
	Private	Integer	numPlayers	Total number of the players, from 2 to 8
	Private	Player	players[8]	Array of player objects
	Private	Go	goCell	Object representing the Go cell
	Private	GoToJail	GoToJailCell	Object representing the GoToJail cell
	Private	Jail	JailCell	Object representing the Jail cell
	Private	OlympicPark	OlympicParkCell	Object representing the OlympicPark cell
	Private	IncomeTax	IncomeTaxCell	Object representing the IncomeTax cell
	Private	LuxuryTax	LuxuryTaxCell	Object representing the LuxuryTax cell
	Private	JFL	JFLCells[6]	Objects representing the JFL cells
	Private	Street	StreetCells[22]	Objects representing the street cells
	Private	Utility	UtilityCells[2]	Objects representing the utility cells
	Private	Metro	MetroCells[4]	Objects representing the metro cells
	Private	JFLDeck	Deck	The JFLDeck object
	Private	Dice	dice1	The first dice
	Private	Dice	dice2	The second dice

Methods	Visibility	Method Name	Description
	Public	Board()	Constructor.
	Public	endTrun()	Transfer the control to the next player.
	Public	shufflePlayers()	To make the players turns randomly.
	Public	addPlayer (Player player)	Add a player to the game.
	Public	rollDice()	Roll two dices; get the two numbers, get the steps to move; set the doubleCount of a player if it is double.
	Public	Street getStreet(Integer streetNo)	Return a street object no:1—22
	Public	Utility getUtility(Integer utilitytNo)	Return a utility object no:1—2
	Public	Metro getMetro(Integer metroNo)	Return a metro object no:1—4
	Public	Dice getDice(Integer diceNo)	Return a Dice object No: 1-2
	Public	JFLDeck getJFLDeck()	Return a JFLDeck object
	Public	Player getPlayer(Integer playerNo)	Return a player object playerNo:from 1to numPlayer
	Public	Player getCurrPlayer()	Return the current player object

4.1.2.1 Method Descriptions

Method name	EndTurn()
Class Name	Board
Functionality	Transfer the control to the next player.
Input	None
Output	None

Pseudo Code	Begin currPlayer = (currPlayer +1) Mod numPlayers End

Method name	shufflePlayers()
Class Name	Board
Functionality	Randomize the order of the players. Call this after adding all players
Input	None
Output	None
Pseudo Code	Begin Provide a random seed to the random number generator Generate 8 unique random numbers from 0 to 7 Re-order the array of players using sequence generated in previous step End

Method name	addPlayer(Player player)
Class Name	Board
Functionality	Add a player to the board
Input	A player
Output	None

Pseudo Code	Begin
	    numPlayers = numPlayers + 1
	    players[numPlayers] = Player(player) // Construct a new Player object
	End

Method name	rollDice()
Class Name	Board
Functionality	Roll the two Dices
Input	None
Output	None
Pseudo Code	see below

```
Begin
        dice1.roll()
        dice2.roll()
        If dice1.getValue() = dice2.getValue()
                currPlayer.incDoubleCount()
        Else
                currPlayer.resetDoubleCount()
        If currPlayer.getDoubleCount() = 3
                currPlayer.goToJail()
        Else
                currPlayer.move(dice1.getValue() + dice2.getValue() )
End
```

Method name	Street getStreet(Integer streetNo)
Class Name	Board
Functionality	Return a street object no:1—22
Input	None
Output	Street object
Pseudo Code	Begin Return street[streetNo] End

Method name	Utility getUtility(Integer utilitytNo)
Class Name	Board
Functionality	Return a utility object no:1—2
Input	None
Output	Utility object
Pseudo Code	Begin Return utility[utilityNo] End

Method name	Metro getMetro(Integer metroNo)
Class Name	Board
Functionality	Return a metro object no:1—4
Input	None
Output	Metro object
Pseudo Code	Begin Return metro[utilityNo] End

Method name	Dice getDice(Integer diceNo)
Class Name	Board
Functionality	Return a metro object no:1—2
Input	None

Output	Dice object

Pseudo Code	
	```
	Begin
	    If (diceNo==2) Return dice2
	    Return dice1
	End
	```

Method name	JFLDeck getJFLDeck()
Class Name	Board
Functionality	Return a JFLDeck object
Input	None
Output	JFLDeck object

Pseudo Code	
	```
	Begin
	    Return Deck
	End
	```

Method name	Player getPlayer(Integer playerNo)
Class Name	Board
Functionality	Return a players object
Input	None
Output	Player objects

Pseudo Code	
	```
	Begin
	    Return player[playerNo]
	End
	```

Method name	Player getCurrPlayer()
Class Name	Board
Functionality	Return the current players objects
Input	None
Output	Player objects

Pseudo Code	
	```
	Begin
	    Return currPlayer
	End
	```

4.1.3 Class <Player>

Class Name	Player			
Inherits from	None			
Description	The actor of this game. It may be a cyber player or human. It has all the attributes and actions to make game goes well.			
Attributes	Visibility	Data type	Name	Description
	Private	String	name	The name of the player
	Private	Integer	id	id of the player.
	Private	Color	color	The color associated with the player.
	Private	String	TokenName	File name if the token image
	Private	Integer	Balance	The balance (amount of money) the player has
	Private	Integer	Position	The number of the cell he is on

	Private	Integer	InDebtTo	This is indicated by the inDebtTo field which can be, 0: Not in Debt; 1 to 8 - In debt to the specified player ID; 9: In debt to Bank
	Private	Boolean	inJail	In jail: true; otherwise: false
	Private	Integer	inJailCount	Number of turns the player has been in jail
	Private	Boolean	iscomputer	If the player is a human player, this is false; true otherwise
	Private	Boolean	HasGoJFC	If the player has a GOJFC, this is true; false otherwise
	Private	Boolean	isBankrupt	True if the player is bankrupted.
	Private	Integer	doubleCount	Number of times a player has rolled doubles

Methods	Visibility	Method Name	Description
	Public	Player()	The constructor
	Public	getName()	To get the name of the player
	Public	Integer getID()	To get the ID of the player
	Public	Integer getColor()	To get the color of the player
	Public	String getTokenName()	To get the name of the player’s token
	Public	Integer getBalance()	To get the player’s balance
	Public	Integer getPosition()	To get the current position of the player
	Public	Boolean isInJail()	To see if the player is in jail: true: in jail
	Public	Integer getJailCouunt()	To see how many turns the player is in the jail.
	Public	incJailCount()	To increase the jailCount by one.
	Public	resetJailCount()	To reset the jailCount to zero
	Public	Boolean isBankRupt()	To see if the player is bankrupted: true: yes
	Public	declareBankruptcy()	The player declares bankruptcy.
	Public	Boolean ownsSomething()	To see if the player owns at least a property.
	Public	move(Integer moveStep)	The token moves cell by cell for moveStep steps, if pass GO, then collect $200!
	Public	debit(Integer amount)	Pay the specified amount
	Public	credit(Integer amount)	Collect the specified amount
	Public	autoMakeMoney()	Called if Computer player is in debt, to make money.
	Public	autoPlay()	This is the AI for a computer player.
	Public	buyProperty (Property theProperty)	To buy an unowned property.
	Public	offerTrade (Property theProperty, Integer amount)	Makes an offer (proposal) to buy another player’s property.
	Public	commitTrade(Property theProperty, Integer amount)	Changes ownership of properties and transfers the amounts of the trade. This actually commits the trade.
	Public	Integer isInDebtTo()	The function getInDebtTo() just returns the inDebtTo attribute
	Public	setInDebtTo(Integer p)	P=0 no debit, 1-8 with player 1-9; 9:with bank
	Public	Boolean hasGoJFC()	True if has a go out of jail free card.
	Public	gotoJail()	Move the token directly, in a straight line, to the jail cell .
	Public	getOutOfJail()	A player is freed from the jail.
	Public	Integer getDoubleCount()	How many double has the player rolled?
	Public	incDoubleCount()	Increase the doubleCount by one.
	Public	resetDoubleCount()	Set the doubleCount to zero.
	Public	Pay50GOJ()	To pay $50 to get out of jail.
	Public	useGOJFC(JFLDeck deck)	Use Go Out Of Jail Free card to get out jail

	Public	Boolean isComputer()	True if a player is a cyber.

4.1.3.1 Method Descriptions

Method name	String getName()
Class Name	Player
Functionality	Get player's name
Input	None
Output	Player's name
Pseudo Code	Begin Return name End

Method name	Integer getID()
Class Name	Player
Functionality	Get player's ID
Input	None
Output	Player's ID
Pseudo Code	Begin Return ID End

Method name	Integer getColor()
Class Name	Player
Functionality	Get player's color
Input	None
Output	Player's color
Pseudo Code	Begin Return color End

Method name	String getTokenName()
Class Name	Player
Functionality	Get player's TokenName
Input	None
Output	Player's TokenName
Pseudo Code	Begin Return tokenName End

Method name	Integer getBalance()
Class Name	Player
Functionality	Get player's balance
Input	None
Output	Player's name
Pseudo Code	Begin Return balance End

Method name	Integer getPosition()
Class Name	Player
Functionality	Get player's current position (the number of cell i.e., cellID)
Input	None
Output	Player's position
Pseudo Code	Begin Return position End

Method name	Boolean isInjail()
Class Name	Player
Functionality	Get player's current inJail (true if in the jail)
Input	None
Output	Player's inJail
Pseudo Code	Begin Return inJail End

Method name	Integer getJailCount()
Class Name	Player
Functionality	Get player's current jail count (how many turns has the player been in the jail)
Input	None
Output	Player's position
Pseudo Code	Begin Return jailCouunt End

Method name	incJailCount()
Class Name	Player
Functionality	Increase the jailCount by one
Input	None
Output	None
Pseudo Code	Begin jailCount = jailCount + 1 End

Method name	resetJailCount()
Class Name	Player
Functionality	Reset the jailCount to zero
Input	None
Output	None
Pseudo Code	Begin JailCount=0 End

Method name	Boolean isBankrupt()
Class Name	Player
Functionality	Get player's isBankrupt value (true if he is bankrupted and will not play.)

Input	None
Output	Player's bankrupt status
Pseudo Code	Begin Return isBankrupt End

Method name	declareBankruptcy()
Class Name	Player
Functionality	Set player's isBankrupt value (true if he is bankrupted and will not play.)
Input	None
Output	None
Pseudo Code	Begin isBankRupt = True If isInDebtTo = 0 to 7 (in debt to a player) Transfer ownership of all properties to the other player Transfer the balance (even if negative) to the other player Else (in debt to bank) All properties become un-owned Balance disappears (given to bank) End

Method name	Boolean ownsSomething(Board board)
Class Name	Player
Functionality	If player owns assert, return true (go over all properties: street, utilities, metro)
Input	Board object
Output	Boolean value

Pseudo Code	Begin
	    For(all cell objects){
	        If (anycell.getOwner()=id) return true;
	        Else return false
	End

Method name	move(Integer moveSteps)
Class Name	Player
Functionality	The player move for moveStep cells(cell by cell). If he pass GO, just collect $200!
Input	Integer moveStep
Output	Display player's position

Pseudo Code	Begin
	    For (moveStep){
	        Position = (Position + 1) Mod 40
	        Move token image to next cell
	        Sound // make a sound
	        If (CellID==GO_CELL_ID) { // if pass GO
	            Credit(200) // collect $200
	        }
	    }
	    Cell.doAction() // Landed on cell, call doAction
	End

Method name	debit(Integer amount)

Class Name	Player
Functionality	To pay $amount.
Input	Integer amount
Output	Decrease balance

Pseudo Code	
	```
	Begin
	Balance-=amount
	If (balance<0){
	    If (isComputer==true) call autoPlay()
	    Message "Your balance is negative. You owe money. You must increase your funds to a positive balance to continue the game, or declare bankruptcy."
	End
	```

Method name	credit(Integer amount)
Class Name	Player
Functionality	To collect $amouny.
Input	Integer amount
Output	Increase balance

Pseudo Code	
	```
	Begin
	    Balance+=amount
	End
	```

Method name	autoPlay(Board board)
Class Name	Player
Functionality	This is the AI for the computer player.
Input	None
Output	None
Pseudo Code	This will be described in detail in section 4.1.19.

Method name	autoMakeMoney(Board board)
Class Name	Player
Functionality	To make money by sell(buy) or mortgage(unmortgage) property automatically. If nothing to sell and balance is les than 0: declare bankruptcy
Input	None
Output	None
Pseudo Code	This will be described in detail in section 4.1.19.5

Method name	buyProperty(Property theProperty)
Class Name	Player
Functionality	To buy a property. Note: This function may need to be overloaded, to supply 3 different functions, one for each of: Street, Metro and Utility classes (due to Visual Basic restrictions).
Input	A property object
Output	Change the color and owner of the property.

Pseudo Code	
	```
	Begin
	    If (theProperty.getOwner<>0) "You cannot buy it. It is owned by a player":exit
	    If (Balance<cell[cellID].cost) "you do not have enough money";exit;
	    theProperty.setOnwer(playerID)
	End
	```

Method name	offerTrade(Property theProperty, Integer amount)
Class Name	Player
Functionality	Want to trade a property with another player. Note: This function may need to be overloaded, to supply 3 different functions, one for each of: Street, Metro and Utility classes (due to Visual Basic restrictions).
Input	Property theProperty, Integer amount
Output	Change the color and owner of the property
Pseudo Code?	Begin If (Balance<theProperty.getCost()) “you do not have enough money”;exit; Call trade pop-up window. End

Method name	commitTrade(Property theProperty, Integer amount)
Class Name	Player
Functionality	Want to trade a property with another player.
Input	Property theProperty, Integer amount
Output	Change the color and owner of the property

Pseudo Code?	Begin
	    If (Balance< theProperty..getCost()) “you do not have enough money”;exit;
	    Utility.display()
	    Player.debit(amount)
	    Player owner = theProperty.getOwner()
	    owner.credit(amount)
	    Player.debit(amount)
	End

Method name	resetDoubleCount()
Class Name	Player
Functionality	Set double count to zero
Input	None
Output	None

Pseudo Code?	Begin
	    If (doubleCount<3) “you can not”;exit;
	    DoubleCount=0
	End

Method name	setInDebtTo (Integer inDebtToPlayer)
Class Name	Player
Functionality	To set whom does the player is debt to
Input	Integer inDebtToPlayer
Output	None
Pseudo Code	Begin IsInDebtTo= inDebtToPlayer End

Method name	Integer isInDebtTo()
Class Name	Player
Functionality	To see whom does the player is debt to, return isDebtTo
Input	None

Output	isInDebtTo
Pseudo Code	Begin Return isInDebtTo End

Method name	hasGOJFC()
Class Name	Player
Functionality	If a player gas GOJFC
Input	None
Output	hasGOJFC
Pseudo Code	Begin Return hasGOJFC End

Method name	goToJail()
Class Name	Player
Functionality	The player land on GO TO Jail cell. He will be sent to jail. He will go in straight ling rather than cell by cell. So there is no action on the way.
Input	None
Output	None
Pseudo Code	Begin inJail = true Move the player’s token in a straight line to the “in jail” cell. End

Method name	getOutofJail()
Class Name	Player
Functionality	The player get out of jail
Input	None
Output	None

Pseudo Code	Begin
	    IsInJail =false
	    Move player’s token from “in jail” to “just visiting” (part of Jail cell)
	End

Method name	pay50GOJ()
Class Name	Player
Functionality	The player pays $50 and gets out of jail
Input	None
Output	None

Pseudo Code	Begin
	    debit(50)
	    getOutofJail()
	End

Method name	useGOJFC(JFLDeck deck)
Class Name	Player
Functionality	The player use GOJEF of get out of jail (yes, without payment)

Input	JFLDeck deck
Output	None

Pseudo Code	Begin
	    getOutofJail()
	    hasGoJFC = false
	    JFLCard card = new JFLCard(GOJFLCardType)
	    deck.insertCard(card)
	End

Method name	Boolean isComputer()
Class Name	Player
Functionality	Is the player is a computer player?
Input	
Output	Boolean value
Pseudo Code	Begin Return isComputer End

4.1.4 Class <JFLDeck>

Class Name	JFLDeck			
Inherits from	None			
Description	Class JFLDeck is the deck of all the Just For Laugh Cards. There are twenty cards. Note: • The deck is implemented as a queue. • A player takes a card from the deck, follows the card's instructions (via the doAction() method in JFLCard) and returns the card to the deck. • If the card withdrawn is a GOJFC (Get Out of Jail Free Card), the he does not insert it back. The player keeps the card. This is taken care of by the player attribute hasGOJFC. If the player is in jail and decides to use his GOJFC, the card will be re-inserted into the deck, and the player's hasGOJFC will be set to False.			
Attributes	Visibility	Data type	Name	Description
	Private	JFLCard	deck[20]	Array of 20 JFL Cards.
Methods	Visibility	Method Name	Description	
	Public	shuffle()	To make the deck random order	
	Public	getCard()	Return a JFL card(dequeue).	
	Public	insertCard()	After a card is used, put it back to the card queue(enqueue).	

4.1.4.1 Method Descriptions

Method name	shuffle()
Class Name	JFLDeck
Description	Randomize the order of the cards in the deck. Note, this method runs only once for each game.
Input	None
Output	None

Pseudo Code	Begin Provide a random seed to the random number generator Generate 20 unique random numbers from 0 to 19 Re-order the array of JFL cards using sequence generated in previous step End

Method name	JFLCard getCard()
Class Name	JFLDeck
Description	Get a JFLCard from the Deck (dequeue it)
Input	Deck[20] in random order
Output	A JFLCard
Pseudo Code	Begin Dequeue the first card; Return this card End

Method name	insertCard(JFLCard)
Class Name	JFLDeck
Description	Put a used JFLCard back the Deck(into bottom, enqueue it)
Input	A JFLCard
Output	None
Pseudo Code	Begin Put the card to the last position; End

4.1.5 Class <JFLCard>

Class Name	JFLCard			
Inherits from	None			
Description	Just For Laugh card object. Each card has it own type and methods There should be no action for this class.			
Attributes	Visibility	Data type	Name	Description
	Private	Integer	cardType	From 0 to 19
	Private	String	description	
Methods	Visibility	Name	Description	
	Public	display()	Display the card.	
	Public	Integer getType()	Get the card type	
	Public	String getDescription()	Get the card description	
	Public	doAction()	For different type of card, do different actions.	

4.1.5.1 Method Descriptions

Method Name	display()
Class Name	JFLCard
Description	Display a Card in a pop-up window
Input	None
Output	None

Pseudo Code	Begin Pop-up a window; Display card.description on the window. End

Method Name	getType()
Class Name	JFLCard
Description	Returns the type of the JFL Card.
Input	None
Output	CardType
Pseudo Code	Begin Return type End

Method Name	String getDescription()
Class Name	JFLCard
Description	Get the card description string
Input	None
Output	Return the card description string.
Pseudo Code	Begin Return description End

Method Name	doAction(Player player)
Class Name	JFLCard
Functionality	Performs the actions described by the card on the player.
Input	Player
Output	None

Pseudo Code

```
Message getDescription()
Switch (getType())
Case 0:  // Go to jail
         Player.goToJail()
Case 1:  // Get Out of Jail Free Card
         Player.setHasGOJFC(True);
Case 2:  // Pass Go, collect 200$
         Player.move( 40 – Player.getPosition() );
Case 3:  // Collect $250 (from bank)
         Player.credit(250);
Case 4:  // Collect $200 (from bank)
         Player.credit(200);
Case 5:  // Collect $150 (from bank)
         Player.credit(150);
Case 6:  // Collect $100 (from bank)
         Player.credit(100) ;
Case 7:  // Collect $50 (from bank)
         Player.credit(50) ;
Case 8:  // Pay $250 (from bank)
         Player.debit(250) ;
Case 9:  // Pay $200 (from bank)
         Player.debit(200) ;
Case 10: // Pay $150 (from bank)
         Player.debit(150);
Case 11: // Pay $100 (from bank)
         Player.debit(100) ;
Case 12: // Pay $50 (from bank)
         Player.debit(50)  ;
Case 13: // Advance to Green street
         Integer moveStepGo = ( cell_id_green_street – player.getPosition() ) Mod 40
         Player.move(moveStepGo)
Case 14: // Advance to St-Laurent street
         Integer moveStepGo = ( cell_id_st_laurent_street – player.getPosition() ) Mod 40
         Player.move(moveStepGo);
Case 15: // Move forward 7 cells
         Player.move(7);
Case 16: // Move forward 11 cells
         Player.move(17);
Case 17: // Move forward 5 cells
         Player.move(5);
Case 18: // Move back 3 cells
         Player.move(-3);
Case 19: // Do Nothing
```

4.1.6 Class <Dice>

Class Name	Dice			
Inherits from	None			
Description	A dice object can roll and give value.			
Attribute	Visibility	Data type	name	Description
	Private	Integer	diceValcue	
Methods	Visibility	Method Name	Description	
	Public	roll()	To roll the dice to produce dice value	
	Public	Integer getValue()	To get the dice value	

4.1.6.1 Method Descriptions

Method name	roll()
Class Name	Dice
Functionality	General a random number from 1-6.
Input	None
Output	DiceValue

Pseudo Code	Begin
	    Radom seed;
	    DiceValue = (radom number)%6 +1
	End

Method name	Integer getValue()
Class Name	Dice
Functionality	Get diceValue
Input	None
Output	DiceValue
Pseudo Code	Begin Return diceValue End

4.1.7 Class <Cell>

Class name	Cell			
Description	Define functions for the inherited classes as a abstract class			
Attributes	Visibility	Data Type	Name	Description
	Private	Integer	id	Id of the cell.
	Private	String	name	Name of the cell.
Methods	Visibility	Method Name	Description	
	Public	getId()	Get the ID of a cell.	
	Public	getName()	Get the name of a cell	
	Public	isOn()	Check whether a player on a cell	
	Public	display()	Define a pure virtual function for its inherited classes	
	Public	showInfo()	Define a pure virtual function for its inherited classes	
	Public	doAction()	Define a pure virtual function for its inherited classes	

4.1.7.1 Method Descriptions

Method name	getId()
Description	Get the value of cell's id
Input	None
Output	Id
Return Type	Integer
Pseudo Code	Begin Return value of id End

Method name	getName()
Description	Get the name of a cell
Input	None
Output	Name
Return Type	Integer
Pseudo Code	Begin Return name of cell End

Method name	isOn()
Description	Get to know whether the token is landed on a cell
Input	Player P
Output	IsOn
Return Type	Bool
Pseudo Code	Begin If (player's position == cell's id) then//a player is on cell Return isOn= true Else //a player is not on cell Return isOn= False Endif End

Method name	display()
Description	Called whenever the state of the cell changes. This function will call the showCell() method in the View module, and pass it all the information (parameters) it needs to refresh the cell on the board.
Input	None
Output	None
Return Type	None

Pseudo Code	Begin
	          // Do nothing. This function will be overloaded by all the sub-classes
	          // This is like a pure virtual function, however, VB does not support this
	End

Method name	showInfo()
Description	Called whenever a user clicks on a cell. This function will call the showInfo() function in the View module, and pass it all the information (parameters) it needs to display on the pop-up window.

Input	None
Output	Pop up a info window
Return Type	Void
Pseudo Code	Begin showCellInfo(this.getName(), this.getType(), this.getDescription()) End

Method name	doAction()
Description	The doAction() method is called whenever a player lands on this cell. It contains the actions that must occur when a player lands on it.
Input	None
Output	None
Return Type	Void
Pseudo Code	Begin // By default, there are no actions to be done End

4.1.8 Class <Property>

Class name	Property			
Description	Inherites from Cell class			
Attributes	Visibility	Data Type	Name	Description
	Private	Integer	price	The price of property.
	Private	Integer	isOwnedBy	Owner of the property.
	Private	Bool	isMortgaged	Flag for mortgage property or not.
Methods	Visibility	Method Name	Description	
	Public	getPrice()	Get price of property	
	Public	getRent()	Define a function to get rent for its inherited classes.	
	Public	setOwner()	Set owner of the property	
	Public	getOwner()	Get owner of the property	
	Public	isMortgage()	Know whether the property is mortgaged of not	
	Public	mortagage()	Sell or mortgage Property	
	Public	unMortagage()	Buy or un-mortgage Property	

4.1.8.1 Method Descriptions

Method name	getPrice()
Description	Get the value of property value
Input	None
Output	Price
Return Type	Integer
Pseudo Code	Begin Return value of price End

Method name	getRent()
Description	Define a same function for its inherited classes
Input	None
Output	None

Return Type	Integer

Pseudo Code	
	```
	Begin
	        Define a virtual function of getRent()
	End
	```

Method name	setOwner()
Description	Set the property to a player
Input	Player P
Output	isOwnedBy
Return Type	Void

Pseudo Code	
	```
	Begin:
	    Set value of isOwnedBy
	End
	```

Method name	getOwner()
Description	Get owner of the property
Input	Player P
Output	None
Return Type	Int

Pseudo Code	
	```
	Begin
	     Return the value of isOwnedBy
	End
	```

Method name	isMortagated()
Description	Get to know whether the property is mortgaged
Input	Player P
Output	IsMortagaged
Return Type	Bool

Pseudo Code	
	```
	Begin
	     Return value of ismortgaged
	End
	```

Method name	mortgage()
Description	Mortgage the property
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	    If(isMortgaged ==TRUE )then
	        Error message(“This property has been mortgaged!”)
	        Exit
	    Else
	        If ( Int mortgagePrice = getMortage() ) then
	            P.Credit( mortagagePrice);//Mortgage the property
	            isMortgaged =TRUE
	     Endif
	End
	```

Method name	UnMortagate()

Description	UnMortgage the property
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	    If(isMortgaged ==FALSE)then
	        Error message("This property has not been mortgaged!")
	        Exit
	    Else  //un-mortgage the property
	        Int un-mortgagePrice = 1.10 * getPrice()
	        P.debit(un-mortgagePrice);
	        isMortgaged =FALSE
	    Endif
	End
	```

4.1.9 Class <Street>

Class name	Street			
Description	Inherits from Property class			
Attributes	Visibility	Data Type	Name	Description
	Private	integer	hotelCount	The number of hotel build by a player
Methods	Visibility	Method Name	Description	
	Public	getRent()	Get rent of street.	
	Public	getHotelPrice()	Get the cost of hotel(s)	
	Public	buildHotel()	Build hotel(s)	
	Public	sellHotel()	Sell hotel(s) to bank	
	Public	display()	Display the token is landed on Street cell	
	Public	showInfo()	Show the info of Street	
	Public	doAction()	Pay the rent to the owner of Street	

4.1.9.1 Method Descriptions

Method name	getRent()
Description	Get rent of street
Input	None
Output	None
Return Type	Integer

Pseudo Code	
	```
	Begin
	    float rate=0.1
	    If (hotelCount ==1) rate=0.5
	    If (hotelCount ==2) rate=1.2
	    If (hotelCount==3) rate=2.2
	    If (hotelCount==4) rate=2.50
	    Return price*rate     // for rent
	End
	```

Method name	getHotelPrice()
Description	Get the cost of hotel(s)
Input	Player P
Output	None

Return Type	Void

Pseudo Code	
	```
	Begin
	    Int rate = 0.6;
	    Return price*rate;
	End
	```

Method name	buildHotel()
Description	Build the hotel on Street
Input	Player P
Output	Pop up a info window
Return Type	Viod

Pseudo Code	
	```
	Begin
	   If( hotelCount < 4)
	      If (getBalance() < getHotelCost()) then
	         Error Message(“Not enough money to build”)
	         exit
	      endif
	   endif
	   P.debit( Property.getHotelPrice())
	   HotelCount = HotelCount + 1
	End
	```

Method name	sellHotel()
Description	Sell the hotel on Street to bank
Input	Player P
Output	None
Return Type	Integer

Pseudo Code	
	```
	Begin
	   GetOwner().credit(getHotelPrice())
	   HotelCount = HotelCount -1
	End
	```

Method name	display()
Description	Show the token on Street cell
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	      This will be overloaded by the subclasses
	End
	```

Method name	showInfo()
Description	Display Window gives detail info on who owns Street or who is visiting on Street
Input	Player P
Output	Pop up a info window
Return Type	Void

Pseudo Code	Begin Pop display windows If (Street is not owned by any player) then Show the name and price of Street Show buy button Else Show the owner of Street Show the hotel on the Street. Show the rent of the Street. Endif End

Method name	doAction()
Description	Buy or sell Street or Pay rent
Input	Player P
Output	None
Return Type	Void

Pseudo Code	Begin
	    If (!isOwnedBy)
	        If (a player decided to buy the un-owned Street)then
	            P.buyProperty(amount)// buy Street
	        Endif
	    Else if (getOwner()==P)
	         If (a player decided to build Hotel and Hotel<4) then
	             buildHotel()
	    Else if(a player decided to sell Hotel to bank)
	         sellHotel()
	    Else if (a player decided to sell the owned Street) then
	         P.offerTrade()
	    Else if (a player decided to mortgage Street) then
	         Property.mortgage()
	    Else if(isMortgage==TRUE & a player decided to un-mortgage
	      Street)then
	          Property.unmortgage()
	    Else{payRent()}
	    Endif
	End

4.1.10 Class <Metro>

Class name	Metro			
Description	Inherits from Property class			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name	Description	
	Public	getRent()	Get rent of metro.	
	Public	display()	Display the token is landed on Metro	
	Public	showInfo()	Show the info of Metro	
	Public	doAction()	Deal with the transaction on metro	

4.1.10.1 Method Descriptions

Method name	getRent()
Description	Get rent of metro
Input	None
Output	None
Return Type	Integer

Pseudo Code	
	```
	Begin
	        Return Property.getPrice() *0.10 // get rent
	End
	```

Method name	display()
Description	Show the token on Metro cell
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	    Show the current player's token on this cell
	    While (any other players on this cell)
	        show the color of the player's token
	End
	```

Method name	showInfo()
Description	Display Window gives detail info on who owns Metro or who is visiting on Metro
Input	Player P
Output	Pop up a info window
Return Type	Void

Pseudo Code	
	```
	Begin
	    Pop up a window to display name of metro
	    if (there is a owner) then
	        show rent
	    show OK button
	    If ( Metro is not owned by any player) then
	        Show buy button
	    Else
	        if (owned by current player) then
	            show Mortgage/UnMortgage button
	        else
	            show Trade button
	        Endif
	    Endif
	End
	```

Method name	doAction()
Description	Buy or sell Metro
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	    If (!isOwnedBy)
	        Exit  // do nothing
	    Else if (getOwner()==P)
	        CurrentPlayer.debit (getRent());
	        P.credit (getRent());
	    Endif
	End
	```

4.1.11 Class <Utility>

Class name	Utility			
Description	Inherited class from Property			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name		Description
	Public	getRent()		Get rent of utility.
	Public	display()		Display the token is landed on Utility cell
	Public	showInfo()		Show the info on Utility
	Public	doAction()		Deal with the transaction with utility

4.1.11.1 Method Descriptions

Method name	getRent()
Description	Get rent of utility
Input	None
Output	None
Return Type	Integer

Pseudo Code	
	```
	Begin
	    float rate=0.1
	    Return price*rate// for rent
	End
	```

Method name	display()
Description	Show the token on Utility cell
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	    Show the current player's token on this cell
	    While (any other players on this cell)
	        show the color of the player's token
	End
	```

Method name	showInfo()
Description	Display Window gives detail info on owner and rent(if any) for other players or mortgage/unmortgage or Trade button for a current player.
Input	Player P
Output	Pop up a info window

Return Type	Void
Pseudo Code	

```
Begin
        Pop up a window to display name of utility
        if (there is a owner) then
              show rent
        show OK button
        If ( utility is not owned by any player) then
             Show buy button
        Else
             if (owned by current player) then
                   show Mortgage/UnMortgage button
             else
                   show Trade button
             Endif
       Endif
End
```

Method name	doAction()
Description	Buy or sell Utility
Input	Player P
Output	None
Return Type	Void
Pseudo Code	

```
Begin
        If (!isOwnedBy){
         If ( a player decided to buy the un-owned Utility)then
            P.buyProperty( int amount)// buy Utility
   }
     else if (getOwner()==P){
      if (a player decided to sell the owned Utility)then
             P.offerTrade()
     Else if (a player decided to mortgage Utility) then
         Property.mortgage()
     Else if(isMortgage==TRUE& a player decided to un-mortgage
          Utility)then
          Property.unmortgage()
               Endif
          }
          else{payRent()}
End
```

4.1.12 Class <Go>

Class name	Go			
Description	Inherits from Cell class			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name	Description	
	Public	display()	Show token on this cell	

4.1.12.1 Method Descriptions

Method name	display()
Description	show the token on Go cell
Input	Player P
Output	Token
Return Type	Void
Pseudo Code	Begin Show current player's token While (any non-current players landed on this cell) Display small color token on the edge of cell End

4.1.13 Class <Jail>

Class name	Jail			
Description	Inherits from Cell class			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name		Description
	Public	display()		Display the player's tokens landed on this cell
	Public	showInfo()		Show the info about token(s)

4.1.13.1 Method Descriptions

Method name	display()
Description	Show the token on Jail cell
Input	Player P
Output	None
Return Type	Void
Pseudo Code	Begin Show current player's token While (any non-current players landed on this cell) Display small color token on the edge of cell End

Method name	showInfo()
Description	Display a window for detail of the players who are currently in or visiting jail.
Input	Player P
Output	Pop up a info window
Return Type	Void
Pseudo Code	Begin Pop up a window While (any players is in jail) Show name and token of a player who is in jail now While (any player is visiting jail) Show name and token of a player who is visiting jail now End

4.1.14 Class <OlympicPark>

Class name	OlympicPark			
Description	Inherits from Cell class			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name	Description	
	Public	display()	show the token on this cell	

4.1.14.1 Method Descriptions

Method name	display()
Description	show the token on OlympicPark cell
Input	Player P
Output	Token
Return Type	Void
Pseudo Code	Begin Show current player's token While (any non-current players landed on this cell) Display small color token on the edge of cell End

```
Begin
    Show current player's token
    While (any non-current players landed on this cell)
        Display small color token on the edge of cell
End
```

4.1.15 Class <GoToJail>

Class name	GoToJail			
Description	Inherits from Cell class			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name	Description	
	Public	display()	Display the player's tokens landed on this cell	
	Public	doAction()	Move the player token to Jail	

4.1.15.1 Method Descriptions

Method name	display()
Description	Show the tokens on GoToJail cell
Input	Player P
Output	Token
Return Type	Void
Pseudo Code	Begin Show the tokens on GoToJail cell End

```
Begin
    Show the tokens on GoToJail cell
End
```

Method name	doAction()
Description	Directly move a player to Jail cell
Input	Player P
Output	None
Return Type	Void

Pseudo Code	
	```
	Begin
	    The player Call GoToJail()
	End
	```

4.1.16 Class <JFL>

Class name	JFL			
Description	Inherited class from Cell			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name	Description	
	Public	display()	Display the player's tokens landed on this cell	
	Public	doAction()	do something according to the JFLCard.	

4.1.16.1 Method Descriptions

Method name	display()
Description	Show the tokens on JFL cell
Input	Player P
Output	Token
Return Type	Void
Pseudo Code	Begin Show the token on JFL While (any players are visiting the JFL cell) Show color of a visiting player End

Method name	doAction()
Description	Draw a JFLcard. Do something according to the JFLCard. Return the JFLCard (except GoOutJailFree card)
Input	Player P
Output	None
Return Type	Void

Pseudo Code	Begin
	    P.move(int JFLid) // Move a token to cell
	    JFLCard.drawCard(Player P)//draw a card
	    PJFLCard.doAction(Player P)//do thing according to the card.
	    If (not a GoOutJailFree card) then
	        Return a JFLCard to deck
	    Endif
	End

4.1.17 Class <IncomeTax>

Class name	IncomeTax			
Description	Inherits from Cell class			
Attributes	Visibility	Data Type	Name	Description

	None	None	None	None
Methods	Visibility	Method Name		Description
	Public	display()		Display the player's tokens landed on this cell
	Public	doAction()		Debit the balance of a player for IncomeTax.

4.1.17.1 Method Descriptions

Method name	display()
Description	Show the tokens on IncomeTax cell
Input	Player P
Output	Token
Return Type	Void
Pseudo Code	Begin Show the current player's token on this cell While (any other players on this cell) show the color of the player's token End

```
Begin
    Show the current player's token on this cell
     While (any other players on this cell)
         show the color of the player's token
End
```

Method name	doAction()
Description	Debit the balance of a player for IncomeTax.
Input	Player P Board B
Output	None
Return Type	Void
Pseudo Code	Begin Find Total property of a player Calculate TotalAmount of assets plus balance Int amount = TotalAmount * tax rate If(amount > 200) then Amount = 200 endif P.debit(int amount)// reduce the incomeTax from a player End

```
Begin
    Find Total property of a player
    Calculate TotalAmount of assets plus balance
    Int amount = TotalAmount * tax rate
       If(amount > 200) then
          Amount = 200
       endif
     P.debit( int amount )// reduce the incomeTax from a player
End
```

4.1.18 Class <LuxuryTax>

Class name	LuxuryTax			
Description	Inherits from Cell class			
Attributes	Visibility	Data Type	Name	Description
	None	None	None	None
Methods	Visibility	Method Name		Description
	Public	display()		Display the player's tokens landed on this cell
	Public	doAction()		Debit the balance of a player for LuxuryTax.

4.1.18.1 Method Descriptions

Method name	display()
Description	Show the tokens on LuxuryTax cell

Input	Player P
Output	Token
Return Type	Void
Pseudo Code	Begin Show the current player's token on this cell While (any other players on this cell) show the color of the player's token End

Method name	doAction()
Description	Debit the balance of a player for luxury tax.
Input	Player P Board B
Output	None
Return Type	Void
Pseudo Code	Begin Int amount = 75 P.debit(int amount)// reduce the LuxuryTax from a player End

4.1.19 Artificial Intelligence (AI)

This section will deal with the artificial intelligence component of the game. As this is a feature of the system that needs to be dealt with separately, we chose to dedicate this section to it.

4.1.19.1 When is AI required?

The AI is required in the cases where a decision for a computer player needs to be made. One example is all activities that can make the player's balance less than zero. Second is that the player need to make decision in the pre-roll dice and post-roll dice scenarios. As shown in the diagram below, they are:

- Buy property
- Build Hotel
- Sell Hotel
- Mortgage Property
- Unmortgage Property
- Get out of Jail
- Get out of Jail free
- Offer Trade
- Accept Trade
- Reject Trade
- Declare Bankruptcy

4.1.19.2 The AI rules

In summary, these are the decisions that the computer player must make.

- Before rolling the dice, check if in jail
- After rolling the dice, check if the player has mortgaged properties and unmortgage them, if possible.
- If the player pays money, (ex: pay rent) and the balance becomes negative, the autoMakeMoney() function is called to attempt to increase funds.
- If the player lands on an un-owned property and the balance is greater than 120% of the cost of the property, buy it.
- If it is possible to build a hotel and balance is greater than 110% of the cost of the hotel, build it.
- If the balance is positive but less than $100, sell a hotel.
- If the balance is positive but less than $200, mortgage a property.
- If the balance is greater than $500, un-mortgage a mortgaged property/
- If the player is in jail and has enough money to pay penalty and get out of jail, pay it.
- If balance is greater than $800 and has two streets of one district, offer a trade to buy another street of the district. (Initiating a trade) This is an optional feature of the system.
- If a trade offer is made to a computer player, the computer must decide whether or not to accept the trade. This can be done in a clever manner. We calculate a ratio = offerAmount / propertyPrice. The higher the ratio, the higher the probability of the computer accepting the trade. We can calculate this probability using a simple formula. Finally, we use the probability in a random number generator to simulate the accepting/rejecting of the trade based on that probability. This algorithm can be seen below. [1]
- The function autoMakeMoney() will attempt to increase the player's funds in the following manner:
    - Sell hotel one by one if has
    - Mortgage utility if has one
    - Mortgage metro if has one
    - Mortgage street one by one if has
    - If cannot sell and cannot mortgage, then declare bankruptcy.

4.1.19.3 AI Testing

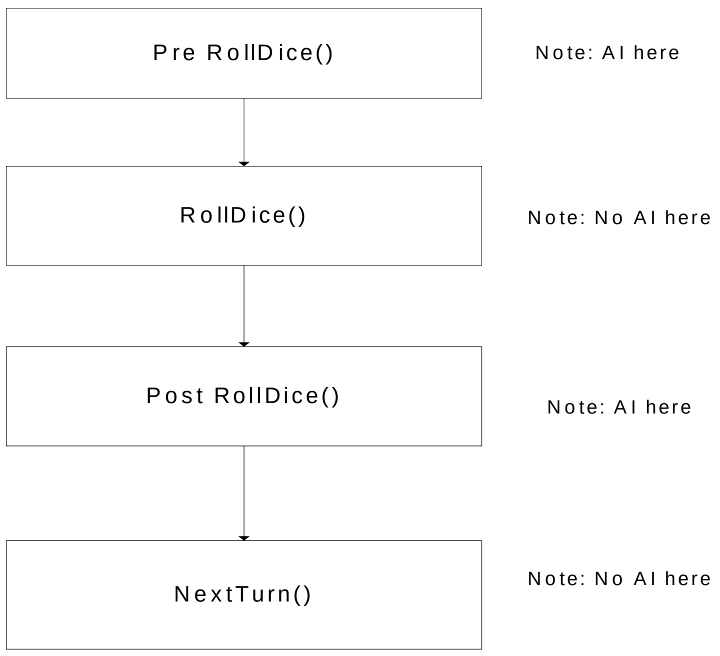
The previous description of AI may be changed during the implementation phase, particularly in the test phase. Testing AI is one of the test scenarios in Appendix B.

[1]
```
Calculate ratio = offerAmount / propertyPrice
If (ratio <= 0.5)
   p = 0
Else
   p = (ratio - 0.5) * 20

randNum = Random Number from 0 to 1

If (randNum <= p)
   Accept trade
Else
   Reject trade
```

4.1.19.4 Details of the AI: autoPlay()

As the following diagram shows, show the autoPlay() function has three steps, pre-rollDice, rollDice and post-rollDice. The AI decisions that need to be made are only on pre-rollDice and post-rollDice. Each will be described using decision trees.

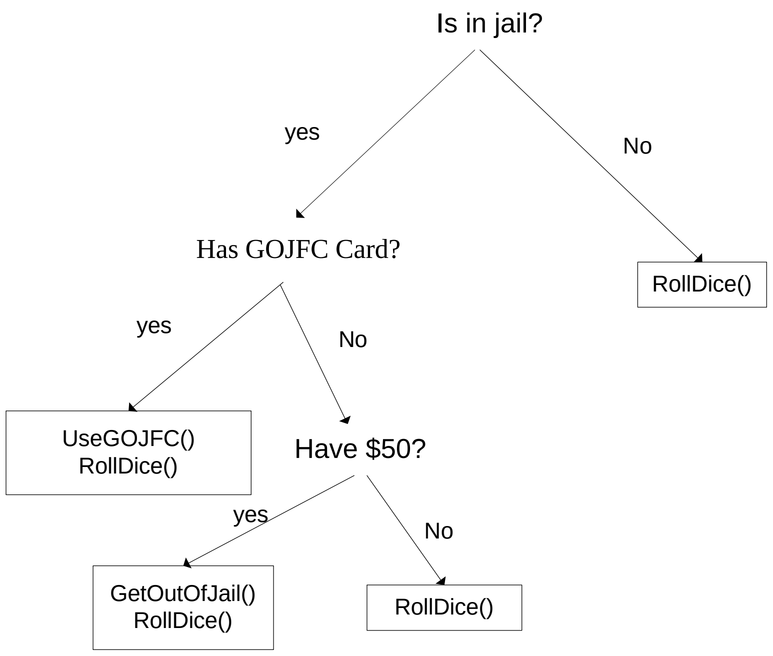
autoPlay() Diagram



Pre Roll Dice AI

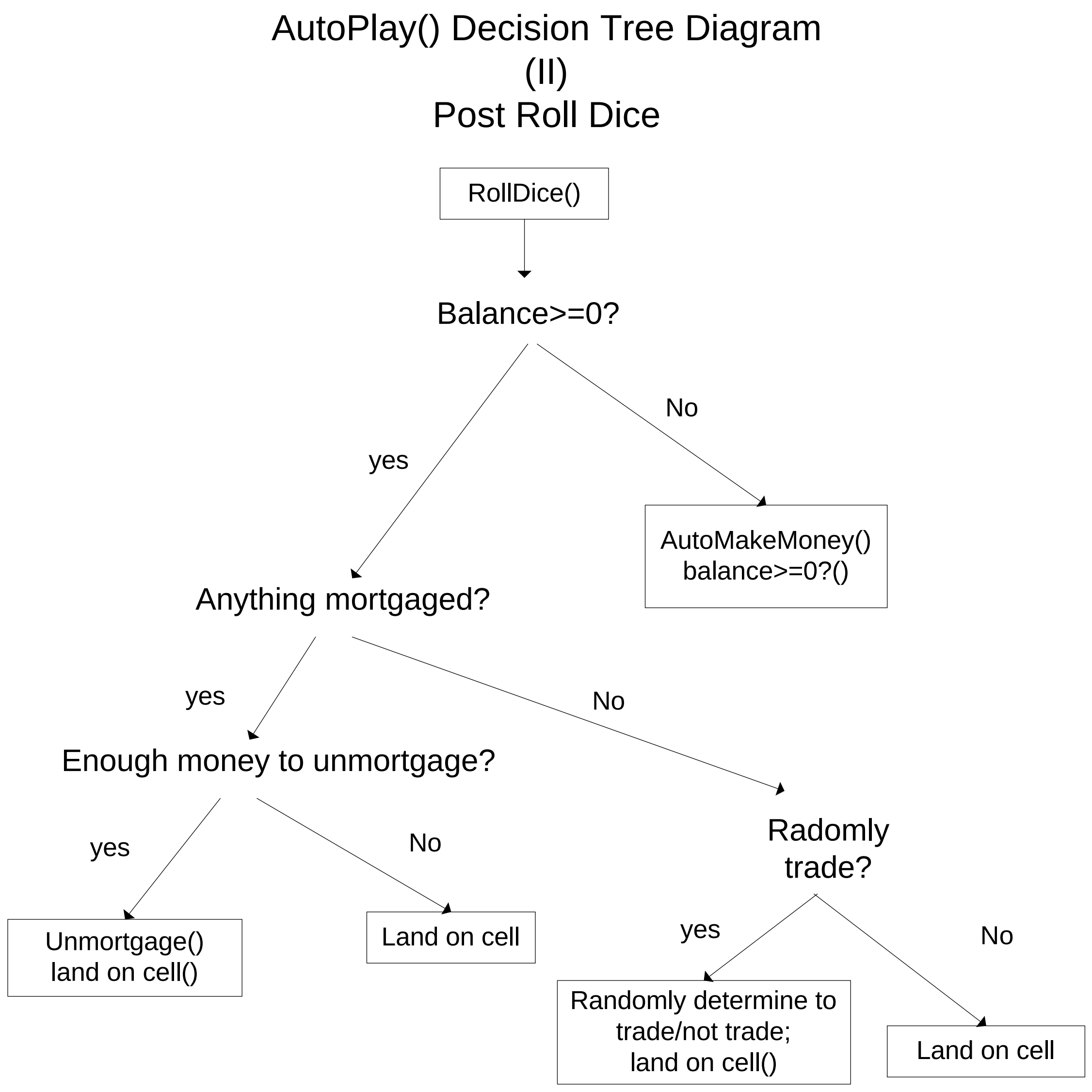
This the first step of the autoPlay() that happens before roll dice. The autoPlay() function only checks if the player is in jail or not. The exit point of this tree is rollDice().

AutoPlay() Decision Tree Diagram
(i)
Pre Roll Dice: Is In Jail



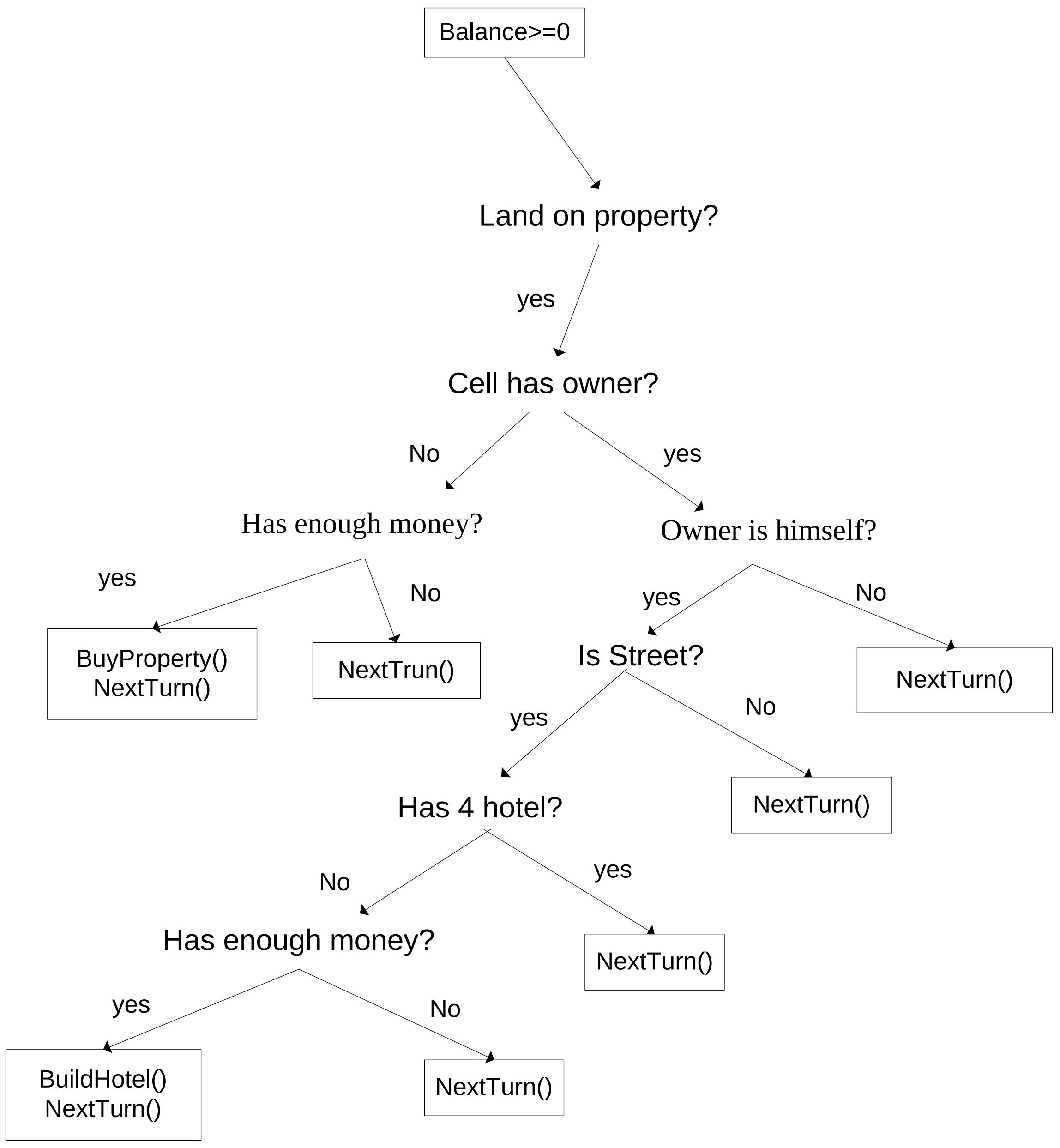
Post Roll Dice AI

This is the second step of the autoPlay() that happens after rolling the dice. First, the balance will be checked. Then Mortgaged situation will be checked. Concerning a computer initiating a trade, we just randomly determine if the computer player does trading and which street and which player the computer want to trade with (Note, this is an optional functionalities of this project). An important case is that if the balance of the player is less than zero, then the function autoMakeMoney() will be called.



This is the autoPlay() function of the computer player that happens after the player lands on a Cell. Here, the computer must decide whether or not to buy property and build hotels.

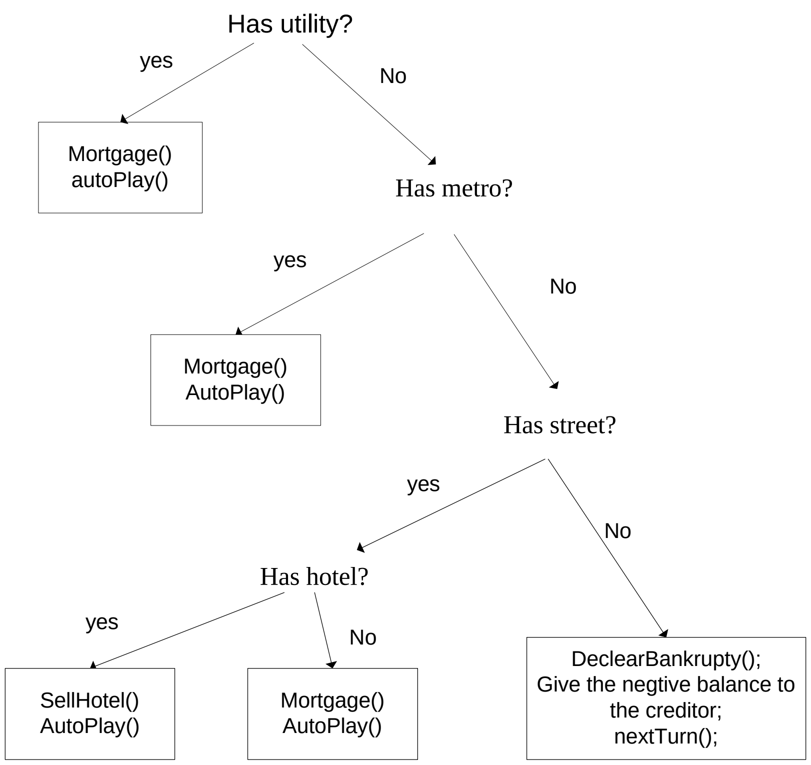
AutoPlay() Decision Tree Diagram (III) Post Roll Dice



4.1.19.5 Details of the AI: autoMakeMoney()

For autoMakeMoney() function, the basic idea is that a player can not sell a property other than selling a hotel. This is the rule that applies to the human player. To make the game fair, we apply the rule to all the computers too. So he can mortgage properties. This is the only change that a computer can declare bankruptcy.

AutoMakeMoney() Decision Tree Diagram



4.2 Module <View >

The view module consists of the different views that the user can see in the system. In Visual Basic terms, these are the VB forms. The following six views constitute the view module:

(i) The GameStart view is the first window that appears when the Montrealopoly game is started. It allows the user to select the names and tokens of the players of the game.

(ii) The MainWindow view is the main game play view. This is the view the players see when the game is in progress. It consists of the game board area, the action buttons area, the player list, the message area and the current player indicator.

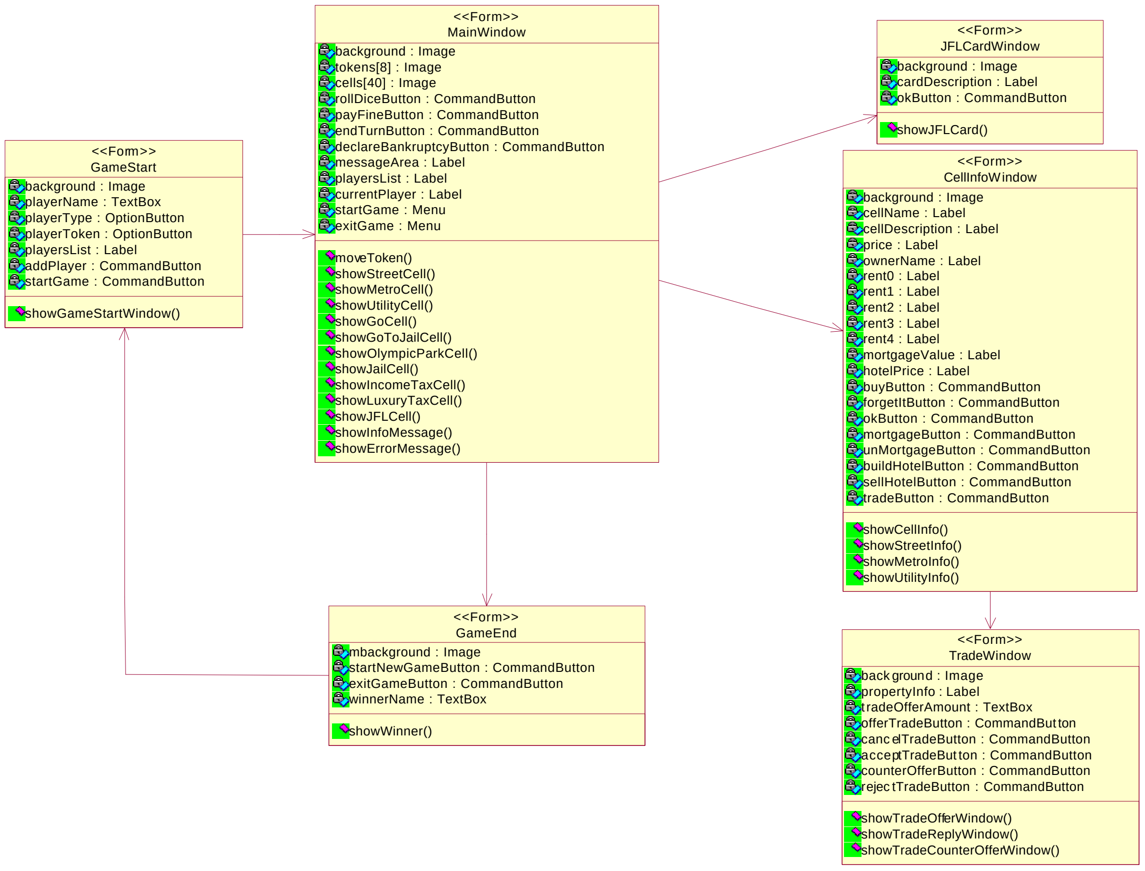
(iii) The JFLCardWindow is a view that is shown to the user if he lands on a JFL cell. It displays to the user the JFL card and what actions he must perform.

(iv) The CellInfoWindow view is a pop-up window that is displayed to the user when he clicks on a cell. From this window, the use can perform many actions on the cell such as: buy the cell, build/destroy hotel, mortgage/un-mortgage, trade, etc…

(iv) The TradeWindow view is used to perform property trades between users. It is designed to interact with the user in: making a trade offer, making a counter offer, accepting or rejecting a trade.

As per the MVC architecture, the View module can be removed, and replaced by a different module, with minimal impact on the system. In fact, this impact can be further minimized if the module's interface is defined clearly, while the internal mechanisms of the view can be different. For example, one can easily replace the current Montreal-themed view with another view that has a different theme, assuming that the new view module provides the same interface, and fulfills the requirements of that interface. This is a major advantage of using MVC architecture.

4.2.1 Module Class Diagram



View Class Diagram

The module interfaces consist of **all** the Public functions in these classes. In other words, the model module will interface with the view module using **all** of the Public functions defined in the above diagram.

4.2.2 Class <GameStart>

Class Name	GameStart (Form)			
Description	The first window that appears when the Montrealopoly game is started. It allows the user to select the names and tokens of the players of the game.			
Attributes	Visibility	Data Type	Name	Description
	Private	Image	background	The background image of the window.
	Private	TextBox	playerName	Allows the user to enter the player name.
	Private	OptionButton	playerType	This can be: Computer or Human player.
	Private	OptionButton	playerToken	Allows the user to select a player token.
	Private	Label	playerList	Displays the list of already added players.
	Private	CommandButton	addPlayer	Adds the player to the players list.
	Private	CommandButton	startGame	Starts the game with the current players.
Methods	Visibility	Method Name	Description	
	Public	showGameStartWindow()	Displays the current view to allow the user to specify the player's information.	

4.2.2.1 Method Descriptions

Method name	showGameStartWindow()
Description	Displays a window that allows the user to enter information about the players, and start the game.
Input	None
Output	None
Return Type	None

4.2.3 Class <MainWindow>

Class Name	MainWindow (Form)			
Description	This is the main game play view. This is the view the players see when the game is in progress. It consists of the game board area, the action buttons area, the player list, the message area and the current player indicator.			
Attributes	Visibility	Data Type	Name	Description
	Private	Image	background	Background image of the window.
	Private	Image	tokens[8]	Up to 8 tokens, one for each player.
	Private	Image	cells[40]	This is the image of the cell on the board. Varies depending on cell type, owner, players on the cell, number of hotels, etc..
	Private	CommandButton	rollDiceButton	Allows the user to roll the dice.
	Private	CommandButton	payFineButton	Allows a user in jail to pay a 50$ fine and get out of jail.
	Private	CommandButton	endTurnButton	Allows the user to indicate that he has finished his turn.
	Private	CommandButton	declareBankruptcy	Allows a user in debt to declare bankruptcy and withdraw from game
	Private	Label	messageArea	This displays different messages to the user, based on what occurs during the game.
	Private	Label	playersList	This displays a list of the players in the game, their balance and token.
	Private	Label	currentPlayer	Displays the name and token of the current player.
	Private	Menu	startGame	Menu to start a new game
	Private	Menu	exitGame	Menu to exit the game
Methods	Visibility	Method Name	Description	
	Public	moveToken()	Moves a player's token by a number of spaces.	
	Public	showStreetCell()	Updates a Street cell according to its state.	
	Public	showMetroCell()	Updates a Metro cell according to its state.	
	Public	showUtilityCell()	Updates a Utility cell according to its state.	
	Public	showGoCell()	Updates the Go cell according to its state.	
	Public	showGoToJailCell()	Updates the GoToJail cell according to its state.	
	Public	showOlympicParkCell()	Updates the OlympicPark cell according to its state.	
	Public	showJailCell()	Updates the Jail cell according to its state.	
	Public	showIncomeTaxCell()	Updates the IncomeTax cell according to its state.	
	Public	showLuxuryTaxCell()	Updates the LuxuryTax cell according to its state.	

	Public	showJFLCell()	Updates a JFL cell according to its state.
	Public	showInfoMessage()	Displays a message in the message area.
	Public	showErrorMessage()	Displays an error message in a pop-up window.

4.2.3.1 Method Descriptions

Method name	moveToken()
Description	Moves a player's token by the specified number of positions.
Input	playerId, numPositions
Output	None
Return Type	None
Pseudo Code	Begin Determine which token is to be moved by using the playerId Calculate the path the token image must take Move the token image along the path, step by step. End

Method name	showStreetCell()
Description	Updates a street cell, according to its state. A street cell can be owned/unowned, mortgaged/unmortgaged, have a certain number of hotels, and have player tokens landed on it.
Input	Integer cellId, Street theStreet, Player owner, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None
Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If cell is owned, display owner's token at the top of the cell If cell is mortgaged, display the "Mortgaged" sign If cell has hotels built on it, display a number of "hotel" icons on cell If currPlayer is on this cell Display currPlayer's token in center of cell For each player who's token is on this cell Display a square with the player's color in lower portion of cell End

Method name	showMetroCell()
Description	Updates a metro cell, according to its state. A metro cell can be owned/unowned, mortgaged/unmortgaged and have player tokens landed on it.
Input	Integer cellId, Metro theStreet, Player owner, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None

Pseudo Code	
	```
	Begin
	        Using cellId, determine the coordinates of the cell to be updated
	        Draw the cell background image
	        If cell is owned, display owner's token at the top of the cell
	        If cell is mortgaged, display the "Mortgaged" sign
	        If currPlayer is on this cell
	          Display currPlayer's token in center of cell
	        For each player who's token is on this cell
	          Display a square with the player's color in lower portion of cell
	End
	```

Method name	showUtilityCell()
Description	Updates a utility cell, according to its state. A street cell can be owned/unowned, mortgaged/unmortgaged and have player tokens landed on it.
Input	Integer cellId, Utility theStreet, Player owner, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None

Pseudo Code	
	```
	Begin
	        Using cellId, determine the coordinates of the cell to be updated
	        Draw the cell background image
	        If cell is owned, display owner's token at the top of the cell
	        If cell is mortgaged, display the "Mortgaged" sign
	        If currPlayer is on this cell
	          Display currPlayer's token in center of cell
	        For each player who's token is on this cell
	          Display a square with the player's color in lower portion of cell
	End
	```

Method name	showGoCell()
Description	Updates the Go cell, according to its state. A Go cell can have any number of players (up to 8) whose tokens are on it.
Input	Integer cellId, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None

Pseudo Code	
	```
	Begin
	        Using cellId, determine the coordinates of the cell to be updated
	        Draw the cell background image
	        If currPlayer is on this cell
	          Display currPlayer's token in center of cell
	        For each player who's token is on this cell
	          Display a square with the player's color in lower portion of cell
	End
	```

Method name	showGoToJailCell()
Description	Updates the GoToJail cell, according to its state. A GoToJail cell can only have one player token on it, momentarily, since any player who lands on it, will go to jail.
Input	Player currPlayer
Output	None
Return Type	None

Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If currPlayer is on this cell Display currPlayer’s token in center of cell End

Method name	showInfoMessage()
Description	Displays an informational message in the message area, which is on the right-hand side panel of the form.
Input	String message_in
Output	None
Return Type	None
Pseudo Code	Begin messageArea.text = messageArea.text + message_in End

Method name	showErrorMessage()
Description	Displays an error message to the user in a pop-up MessageBox window. The user can then click OK to continue.
Input	String message_in
Output	None
Return Type	None
Pseudo Code	Begin MessageBox(message_in) End

Method name	showOlympicParkCell()
Description	Updates the OlympicPark cell, according to its state. An OlympicPark cell can have any number of players (up to 8) whose tokens are on it.
Input	Integer cellId, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None
Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If currPlayer is on this cell Display currPlayer's token in center of cell For each player who's token is on this cell Display a square with the player's color in lower portion of cell End

Method name	showJailCell()
Description	Updates the Jail cell, according to its state. A Jail cell can have any number of players (up to 8) whose tokens are in the visiting area, and any number of players whose tokens are in the "In Jail" area.
Input	Integer cellId, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None
Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If currPlayer is on this cell If currPlayer is in jail Display currPlayer's token in the "In Jail" area Else Dislpay currPlayer's token in the "Just Visiting" area For each player who's token is on this cell If player is in jail Display a square with the player's color in the "In Jail" area Else Display a square with the player's color in the "Just Visiting" area End

Method name	showIncomeTaxCell()
Description	Updates the IncomeTax cell, according to its state. The IncomeTax cell can have any number of players (up to 8) whose tokens are on it.
Input	Integer cellId, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None
Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If currPlayer is on this cell Display currPlayer's token in the center of the cell. For each player who's token is on this cell Display a square with the player's color in the lower portion of the cell. End

Method name	showLuxuryTaxCell()
Description	Updates the LuxuryTax cell, according to its state. The LuxuryTax cell can have any number of players (up to 8) whose tokens are on it.
Input	Integer cellId, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None
Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If currPlayer is on this cell Display currPlayer's token in the center of the cell. For each player who's token is on this cell Display a square with the player's color in the lower portion of the cell. End

Method name	showJFLCell()
Description	Updates a JFL cell, according to its state. A JFL cell can have any number of players (up to 8) whose tokens are on it.
Input	Integer cellId, Player currPlayer, Player playersOnCell[8]
Output	None
Return Type	None
Pseudo Code	Begin Using cellId, determine the coordinates of the cell to be updated Draw the cell background image If currPlayer is on this cell Display currPlayer's token in the center of the cell. For each player who's token is on this cell Display a square with the player's color in the lower portion of the cell. End

4.2.4 Class <JFLCardWindow>

Class Name	JFLCardWindow (Form)			
Description	This view is shown to the user if he lands on a JFL cell. It displays to the user the JFL card and what actions he must perform.			
Attributes	Visibility	Data Type	Name	Description
	Private	Image	background	Background image of the window.
	Private	Label	cardDescription	Displays the description of the JFL card.
	Private	CommandButton	okButton	The game proceeds when the player clicks ok.
Methods	Visibility	Method Name	Description	
	Public	showJFLCard()	Displays this pop-up window with the given card description.	

4.2.4.1 Method Descriptions

Method name	showJFLCard()
Description	Displays this pop-up window with the given card description.
Input	String cardDescription
Output	None
Return Type	None
Pseudo Code	Begin Create a new pop-up window Draw the window background Display the card description Display the okButton End

4.2.5 Class <CellInfoWindow>

Class Name	CellInfoWindow (Form)			
Description	This view is a pop-up window that is displayed to the user when he clicks on a cell. From this window, the use can perform many actions on the cell such as: buy the cell, build/destroy hotel, mortgage/un-mortgage, trade, etc…			
Attributes	Visibility	Data Type	Name	Description
	Private	Image	background	Background image of the window.
	Private	Label	cellName	Displays the name of the cell.
	Private	Label	cellDescription	Displays the description of the cell.
	Private	Label	price	Displays the purchase price of the cell.
	Private	Label	ownerName	Displays the owner of the cell.
	Private	Label	rent0	Displays the rent with no hotels.
	Private	Label	rent1	Displays the rent with 1 hotel.
	Private	Label	rent2	Displays the rent with 2 hotels.
	Private	Label	rent3	Displays the rent with 3 hotels.
	Private	Label	rent4	Displays the rent with 4 hotels.
	Private	Label	mortgageValue	Displays the mortgage value of the cell.
	Private	Label	hotelPrice	Displays the price of building a hotel.
	Private	CommandButton	buyButton	Allows user to buy the property.
	Private	CommandButton	okButton	Returns control to the MainWindow.
	Private	CommandButton	mortgageButton	Allows user to mortgage a property.
	Private	CommandButton	unMortgageButton	Allows user to un-mortgage a property.
	Private	CommandButton	buildHotelButton	Allows user to build a hotel.
	Private	CommandButton	sellHotelButton	Allows user to destroy (sell) a hotel.
	Private	CommandButton	tradeButton	Allows the user to trade the property.
Methods	Visibility	Method Name	Description	
	Public	showCellInfo()	Displays this view with general information about a cell.	
	Public	showStreetInfo()	Displays this view with information specific to a Street cell.	
	Public	showMetroInfo()	Displays this view with information specific to a Metro cell.	
	Public	showUtilityInfo()	Displays this view with information specific to a Utility cell.	

4.2.5.1 Method Descriptions

Method name	showCellInfo()
Description	Displays a pop-up window when a user clicks on a cell. This window contains information about the cell.
Input	Integer cellName, cellType, cellDescription
Output	None
Return Type	None
Pseudo Code	Begin Create a new pop-up window Depending on the cellType, draw appropriate background Display the cellName Display the cellDescription Display the ok button End

Method name	showStreetInfo()
Description	Displays a pop-up window when a user clicks on a street cell. This window contains information about the street, owner, price, rent, hotel and mortgage.
Input	Player currPlayer, Street theStreet
Output	None
Return Type	None
Pseudo Code	See below

```
Begin
        Create a new pop-up window
        Draw the background specific to a street
        Display the street name using theStreet.getName()
        Display the street buying price using theStreet.getPrice()
        If the street is owned by someone
          Display the street owner using theStreet.getOwner()
        Else
          Display the word "Vacant"
        Display the base rent using theStreet.getRentWithHotels(0)
        Display the rent with 1 hotel using theStreet.getRentWithHotels(1)
        Display the rent with 2 hotel using theStreet.getRentWithHotels(2)
        Display the rent with 3 hotel using theStreet.getRentWithHotels(3)
        Display the rent with 4 hotel using theStreet.getRentWithHotels(4)
        Display the mortgage value using theStreet.getMortgage()
        Display the cost of building a hotel using theStreet.getHotelPrice()
        If the street is unowned
          Display buy and ok buttons
        Else
          If theStreet.getOwner() = currPlayer   (ie the player who clicked is the owner)
            If theStreet.isMortgaged()
              Display unMortgage and ok buttons
            Else
              If theStreet.hotelCount() = 0
                Display mortgage button
              Display buildHotel, sellHotel and ok buttons
          Else
            Display trade and ok buttons
End
```

Method name	showMetroInfo()
Description	Displays a pop-up window when a user clicks on a Metro cell. This window contains information about the Metro cell.
Input	Player currPlayer, Metro theMetro
Output	None
Return Type	None
Pseudo Code	Begin Create a new pop-up window Display the metro background image Display the metro name using theMetro.getName() Display the street buying price using theMetro.getPrice() If the street is owned by someone Display the street owner using theMetro.getOwner() Else Display the word “Vacant” Display the rent value using theMetro.getRent() Display the mortgage value using theMetro.getMortgage() If the street is unowned Display buy and ok buttons Else If theMetro.getOwner() = currPlayer (ie the player who clicked is the owner) If theMetro.isMortgaged() Display unMortgage and ok buttons Else Display Mortgage and ok buttons Else Display trade and ok buttons End

Method name	showUtilityInfo()
Description	Displays a pop-up window when a user clicks on a cell. This window contains information about the cell.
Input	Player currPlayer, Utility theUtility
Output	None
Return Type	None
Pseudo Code	See below

```
Begin
        Create a new pop-up window
        Display the utility background image
        Display the metro name using theUtility.getName()
        Display the street buying price using theUtility.getPrice()
        If the street is owned by someone
          Display the street owner using theUtility.getOwner()
        Else
          Display the word "Vacant"
        Display the rent value using theUtility.getRent()
        Display the mortgage value using theUtility.getMortgage()
        If the street is unowned
          Display buy and ok buttons
        Else
          If theUtility.getOwner() = currPlayer  (ie the player who clicked is the owner)
            If theUtility.isMortgaged()
              Display unMortgage and ok buttons
            Else
              Display Mortgage and ok buttons
          Else
            Display trade and ok buttons
End
```

4.2.6 Class <TradeWindow >

Class Name	TradeWindow (Form)			
Description	This view is used to perform property trades between users. It is designed to interact with the user in: making a trade offer, making a counter offer, accepting or rejecting a trade.			
Attributes	Visibility	Data Type	Name	Description
	Private	Image	background	Background image of the window.
	Private	Label	propertyInfo	Displays information about the property being traded.
	Private	TextBox	tradeOfferAmount	Allows the user to enter the amount he is willing to pay.
	Private	CommandButton	offerTradeButton	Submits trade proposal.
	Private	CommandButton	cancelTradeButton	Cancels the trade proposal.
	Private	CommandButton	acceptTradeButton	Allows user to accept the trade.
	Private	CommandButton	counterOfferButton	Allows owner to make a counter offer.
	Private	CommandButton	rejectTradeButton	Allows owner to reject the proposal.
Methods	Visibility	Method Name		Description
	Public	showTradeOfferWindow()		Displays this view, allowing a user to make a trade offer to purchase another player's property.
	Public	showTradeReplyWindow()		Displays this view, allowing the owner of the property to reply to a trade offer made by another player.
	Public	showTradeCounterOfferWindow()		Displays this view, allowing the trade initiator to reply to a counter offer.

4.2.6.1 Method Descriptions

Method name	showTradeOfferWindow()
Description	This window is displayed when a user wants to make a trade offer to another player. Here, the initiator of the trade can enter the amount he is willing to pay to buy the property.
Input	String propertyType, propertyName, String ownerName, String traderName
Output	None
Return Type	None

Pseudo Code	Begin
	    Create a new pop-up window
	    Draw the window background
	    Display the propertyType, propertyName
	    Display the ownerName
	    Display the traderName
	    Display the tradeOfferAmount TextBox
	    Display the offerTrade, cancelTrade buttons
	End

Method name	showTradeReplyWindow()
Description	This window is displayed to the owner of a property, when another player has made an offer to buy this property. It allows the user to accept / reject / counter-offer the trade.
Input	String propertyType, propertyName, String ownerName, String traderName, String priceProposed
Output	None
Return Type	None

Pseudo Code	Begin
	    Create a new pop-up window
	    Draw the window background
	    Display the propertyType, propertyName
	    Display the ownerName
	    Display the traderName
	    Display the priceProposed
	    Display the tradeOfferAmount TextBox
	    Display the acceptTrade, counterOffer and rejectTrade buttons
	End

Method name	showTradeCounterOfferWindow()
Description	This window is displayed to the initiator of a trade, when the owner of the property being traded has made a counter offer. It allows him to accept / reject the counter-offer.
Input	String propertyType, propertyName, String ownerName, String traderName, String priceProposed
Output	None
Return Type	None

Pseudo Code	Begin
	    Create a new pop-up window
	    Draw the window background
	    Display the propertyType, propertyName
	    Display the ownerName
	    Display the traderName
	    Display the priceProposed
	    Display the tradeOfferAmount TextBox
	    Display the acceptTrade, rejectTrade buttons
	End

4.2.7 Class <GameEndWindow >

Class Name	GameEndWindow (Form)			
Description				
Attributes	Visibility	Data Type	Name	Description
	Private	Image	background	Background image of the window.
	Private	CommandButton	startNewGameButton	Allows user to start a new game.
	Private	CommandButton	exitGameButton	Allows user to exit the game.
	Private	TextBox	winnerName	Displays the name of the winner.
Methods	Visibility	Method Name	Description	
	Public	showWinnerWindow()	Displays this view, showing the winner of the game.	

Method name	showWinnerWindow()
Description	This window is displayed when a winner has been declared in the game. It allows the user to either exit the game or to start a new game.
Input	String winnerName
Output	None
Return Type	None
Pseudo Code	Begin Create a new pop-up window Draw the window background Display the winnerName Display the startNewGame and exitGame buttons End

4.3 Module <Controller >

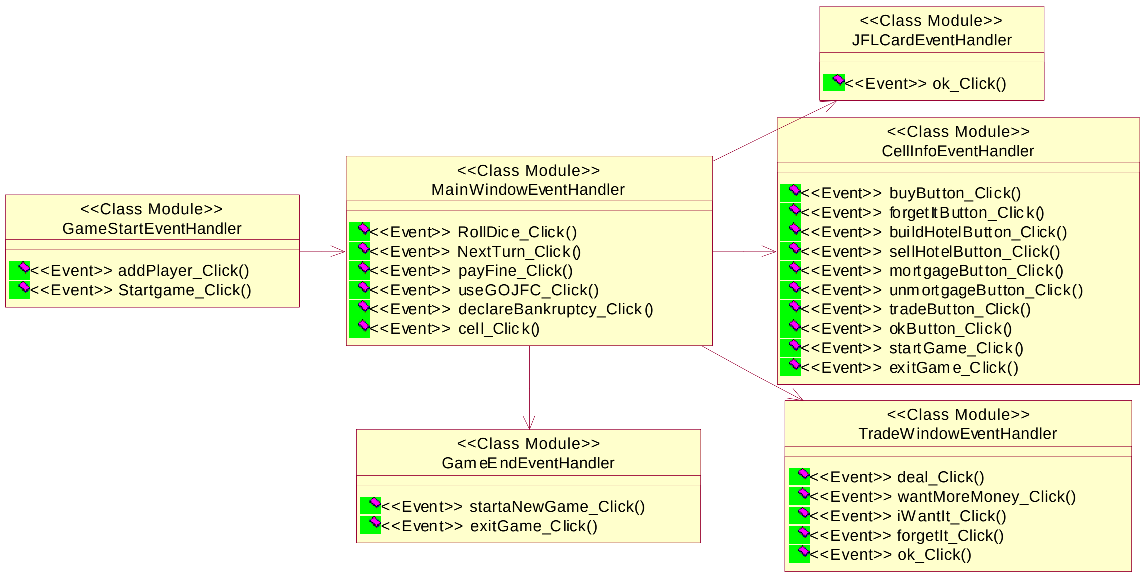
For the Controller module, we do not have any class since we implement the game in VB. In VB, the views are forms or windows (as described in section 4.2). There are many functions and event handlers associated with each form. These event handlers and functions make up the controller module.

4.3.1 Module Class Diagram

Our project has six forms (windows). They are:

- The GameStart window
- The Main window
- The JFLCard window
- The Trade window
- The CellInfo window
- The GameEnd window

Each window has some event handlers that are associated with the window. Mostly, these are button click events. The one exception is the event handler for user-clicks on a cell. This is the event cell_Click() of the Main window. Note that there are 40 cells of different types, having different attributes. The display functionalities of each cell_Click() are different. The status of the cells will also change the display content in the window. The details are described in the GUI section. The event handlers are shown in the diagram below.



<<Class Module>>
JFLCardEventHandler
<<Event>> ok_Click()
<<Class Module>>
CellInfoEventHandler
<<Event>> buyButton_Click()
<<Event>> forgetItButton_Click()
<<Event>> buildHotelButton_Click()
<<Event>> sellHotelButton_Click()
<<Event>> mortgageButton_Click()
<<Event>> unmortgageButton_Click()
<<Event>> tradeButton_Click()
<<Event>> okButton_Click()
<<Event>> startGame_Click()
<<Event>> exitGame_Click()
<<Class Module>>
MainWindowEventHandler
<<Event>> RollDice_Click()
<<Event>> NextTurn_Click()
<<Event>> payFine_Click()
<<Event>> useGOJFC_Click()
<<Event>> declareBankruptcy_Click()
<<Event>> cell_Click()
<<Class Module>>
GameStartEventHandler
<<Event>> addPlayer_Click()
<<Event>> Startgame_Click()
<<Class Module>>
GameEndEventHandler
<<Event>> startaNewGame_Click()
<<Event>> exitGame_Click()
<<Class Module>>
TradeWindowEventHandler
<<Event>> deal_Click()
<<Event>> wantMoreMoney_Click()
<<Event>> iWantIt_Click()
<<Event>> forgetIt_Click()
<<Event>> ok_Click()

Controller Class Diagram

4.3.2 Event Handlers

The main function of the controller module is to handle events and control the Model. Therefore, the controller will handle the events (user-clicks on elements of the form) and then manipulate the Model module through its interface.

Another function of the controller module is to validate input. If any error is found, the event handler will simply pass an error message to the MainWindow view. This will be done via the showErrorMessage() function which will display a message box with the error message.

4.3.2.1 GameStart Window Event Handlers

<u>Event addPlayer_Click():</u>
<u>Check the input information that includes user name and user type (human or computer and token file name that selected.).</u>
<u>If any of them is missing, pass error message to the window and the window will pop up message box to inform what the user did not input.</u>

- If the information is completed, then list the player's name in the player list box.

<u>Event startGame_Click():</u>
<u>Initialize the board, create a board object and create all the cell objects: 22 street cells, six JFL cards cells, four Metro cells, two utilities cells, one Go cell, one GotoJail cell, one pay income tax cell, one pay luxury tax cell one jail cell and one Olympic park cell;</u>

- Create two dices objects, a deck of twenty JFJ cards and numbers of player objects (from two to eight).
- Randomly generate the ID of each player.
- Assign $1500 to each player (ie: Player.credit(1500))
- Randomly generate the ID of each JFL card in the deck.
- Display all the cells. For each one, call the display() method.
- Assign the control to the first player, make currPlayer = 1.

4.3.2.2 Main Window Event Handlers

The main window has several buttons and 40 cells. Clicking on a button will active the event handlers. Clicking on the cells will display the CellInfo window. In addition, the main window has a menu bar that has several functionalities. Those can also be the event handlers.

The functionalities in the menu bar are:

- Open file(optional)
- Save file(optional)
- Change option(optional)
- Help(optional)
- Start game: start the startGame window to start a new game.
- Exit game: exit the game.

The most important event handlers are the button-click event handlers and cell-click event handlers.

<u>Event RollDice_Click()</u>

- Disable the rolldice button
- Enable the nextTurn button
- Call Board.rollDice() to roll the two dices
- If roll double, call currPlayer. incDoubleCount() to increase the double count of the payer.
- Move the token of the player by calling the currPlayer.move(int moveStep) function.

Note:
- Before RollDice_Click(), a player can do any of the pre-roll dice actions.
- Similarly, after RollDice_Click(), a player can do any of the post-roll dice actions.

Event NextTurn_Click()
- Enable the rolldice button;
- Disable the nextTurn button;
- Pass the control to the next player by calling the method Board.endTurn()

Event PayFine_Click()
- Call Board.getCurrPlayer() to retrieve the current player object
- Call Player.pay50GOJ()

Event UseGOJFC_Click()
- Call currPlayer.useGOJFC() function to change the inJail attribute to false
- The currpPlayer can play again.

Event Cell_Click()
- Determine the cell type and id
- Call Board.getStreet(id) (or getUtility(id)) to get the Cell object
- Call Cell.showInfo(), which will pass the information about the cell to the View module's showStreetInfo() function.

Event DeclareBankruptcy_Click()
- If the balance is not negative, send a error message to the main window.
- Otherwise, call currPlayer.declareBankruptcy() function to set the player bankrupted.
- Pass the control to the next player.

4.3.2.3 JFLCard Window Event Handlers

Event Ok_Click()
- Close the JFL card window.
- Call JFLCard.doAction() function to manipulate the current player.

Note, the details of how each JFL card does its action is described in section 4.1.

4.3.2.4 CellInfo Window Event Handlers

Event BuyIt_Click()
- Call Board.getCurrPlayer() to get the current player object.
- Call Player.buyProperty(property id)

Event ForgetIt_Click()
- Close the window.
- Set the main window as the active window.

Event BuildHotel_Click()
- Call Board.getStreet() to get the Street object that needs to be modified.
- If the player cannot buy the property since he does not have enough money, a error message will be displayed.
- Otherwise, Call street.buildHotel() function to add a hotel on the street.

Event SellHotel_Click()
- Call Board.getStreet() to get the Street object that needs to be modified.
- Call street.sellHotel() function to destroy a hotel on the street.

Event Mortgage_Click()
- Call property.mortgage() function to mortgage it.

Note, in this case, the user will not make mistake like mortgage a mortgaged property since the mortgage button is disabled if the property has been mortgaged.

Event Unmortgage_Click()
- Call property.unMortgage() function to mortgage it.

Note, in this case, the user will not make mistake like un-mortgage an un-mortgaged property since the mortgage button is disabled if the property has been mortgaged.

Event Trade_Click()
- Display trade window.
- Find the player to trade by calling cell.getOwner() function

Event Ok_Click()
- Close the window
- Set the main window as the active window.

4.3.2.5 Trade Window Event Handlers

Event Deal_Click()
- The trade is accepted.
- Call Player.commitTrade() to update the Model with the trade transactions
- The window is closed.

Event WantMoreMoney_Click()
- Another player can input the cost he wants to offer.

Event IWantIt_Click()
- Initiate the negotiations for the property
- Check if the input value is valid or not.

Event ForgetIt_Click()
- Close the window. The player does not want to trade on this property.
- Set the main window as the active window.

Event Ok_Click()
- Close the window.
- Set the main window as the active window.

4.3.2.6 GameEnd Window Event Handlers

Event StartaNewGame_Click()
- Close the gameEnd window.
- Open the startGame window. Start a new game.

Event ExitGame_Click()
- Close the window and end the game.

5. Team Members Log Sheets

5.1 Stefan Thibeault

Date	Task	Duration
Sept. 30	Initial design meeting	1 Hour
Oct. 5	Group meeting – came up with class diagram	6 Hours
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 9	Quick meeting with the professor	0.5 Hours
Oct. 11	Researched MVC model	4 Hours
Oct. 12	Group Meeting – Worked with MVC Model	8 Hours
Oct. 13	Worked on system architecture	4 Hours
Oct. 15	Worked on section 2	4 Hours
Oct. 18	Worked on section 3.2	4 Hours
Oct. 19	Group meeting – reviewed decision trees, activity diagrams, made corrections	10 Hours
Oct .20	Completed corrections	1 Hour
Oct. 22	Sections 2 and 3.2 revisions	6 Hours
Oct. 23	Final revisions	2 Hours
	Total:	51.5

5.2 Robert Hanna

Date	Task	Duration
Sept. 30	Initial design meeting	1 Hour
Oct. 5	Group meeting – came up with class diagram	6 Hours
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 9	Quick meeting with professor	0.5 Hours
Oct. 12	Group Meeting – Worked with MVC Model	8 Hours
Oct. 17	Individual – Section 4.1 part 2 corrections, comments	4 Hours
Oct. 15	Individual – Rational Rose VB integration	4 Hours
Oct. 18	Section 4.1 part 1 corrections	2 Hours
Oct. 19	Group meeting – reviewed decision trees, activity diagrams	6 Hours
Oct. 19	Individual – View module	4 Hours
Oct. 20	Individual – Finalized View module	2.5 Hours
Oct. 21	Individual – Document Integration and corrections	8 Hours
Oct. 22	Individual – Section 3.3 – Dynamic Models Document Integration – corrections	12 Hours
Oct. 23	Individual – Section 4.3 integration, correction	5 Hours
Oct. 23	Final Revision	2 Hours
	Total:	66

5.3 Simon Lacasse

Date	Task	Duration
Oct. 5	Group meeting – came up with class diagram	6 Hours
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 19	Implementation Group meeting	3 Hours
Oct. 20	Worked on Implementation	6 Hours
	Total:	16

5.4 Alexandre Bosserelle

Date	Task	Duration
Sept. 30	Initial design meeting	1 Hour
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 17	User interface for design document	4 Hours
Oct. 18	User interface for design document and design of new graphical components	12 Hours
	Total:	18 Hours

5.5 Eugena Zolorova

Date	Task	Duration
Sept. 30	Initial design meeting	1 Hour
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 21	Individual – Introduction	3 Hours
	Total:	8 Hours

5.6 Zhi Zhang

Date	Task	Duration
Sept 28	Meeting/scenario, Class for prototype 1	6
Sept 29	Ai rules	2
Sept 30	Scenario design/ Class prelimary design/ Initial design meeting	5
Oct 1	Class preliminary design	7
Oct 2	Scenario design	6
Oct 3	Scenario design	8
Oct 4	Scenario design/Testing scenario	8
Oct 5	Meeting/Class design/Ai rules/JFLCard	6
Oct 6	Class detail design: Board	4
Oct 7	Group meeting- Class detail design/discussion: Dice JFLDeck	7
Oct 8	Class detail design: player	6
Oct 9	Class detail design: AI Detail player	9
Oct 10	Revise: Check result with group mate	5
Oct 11	Study MVC design	6
Oct 12	Meeting & MVC	6
Oct 15	Decision tree of AI:AutoPlay()	5
Oct 16	Decision tree of AI: AutoMakeMoney()	4
Oct 18	Activity diagrams/report 4.3	4
Oct 19	Meeting/revise diagrams	9
Oct 20	Report 4.3	5
Oct 21	Report 4.3/revise diagrams	4
Oct 22	Report 4.3, appendix I, II	9
	Total:	130 Hours

5.7 Xin Xi

Date	Task	Duration
Sept. 30	Initial design meeting	1 Hour
Oct. 5	Group meeting – came up with class diagram	6 Hours
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 9	Individual – Class Descriptions	6 Hours
Oct. 10	Individual – Class Descriptions	6 Hours
Oct. 12	Group Meeting – Worked with MVC Model	8 Hours
Oct. 19	Group meeting – reviewed decision trees, activity diagrams	6 Hours
Oct. 20	Individual – Class Description Corrections	3 Hours
	Total:	37 Hours

5.8 Patrice Michaud

Date	Task	Duration
Sept. 30	Initial design meeting	1 Hour
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 12	Group Meeting – Worked with MVC Model	4 hours
Oct. 19	Implementation Group meeting	3 Hours
	Total:	9 Hours

5.9 Hu Shan Liu

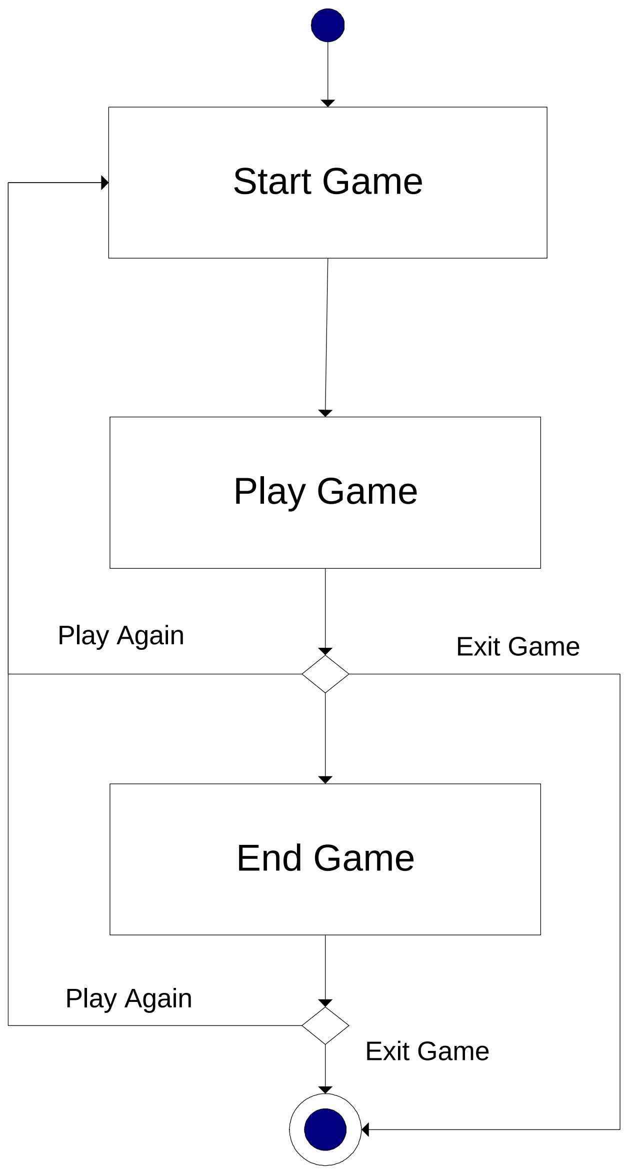
Date	Task	Duration
Sept. 30	Demonstrated mortgage program	0.5 Hour
Oct. 7	Group meeting – discussed class diagrams	1 Hour
Oct. 12	Group Meeting – Worked with MVC Model	4 Hours
Oct. 19	Implementation Group meeting	3 Hours
	Total:	8.5 Hours

5.10 Jens Witkowski

Date	Task	Duration
Oct. 19	Implementation Group meeting	3 Hours
	Total:	3 Hours

6. Appendix A – Game Flow

General activity diagram of the Montrealopoly game



There are three main scenarios that make the game, the start game, the play game and the end game scenario. The first and last scenario is composite of one scenario for each. The play game scenario has several sub scenarios to make the game attractive.

This diagram gives the general structure of the game.

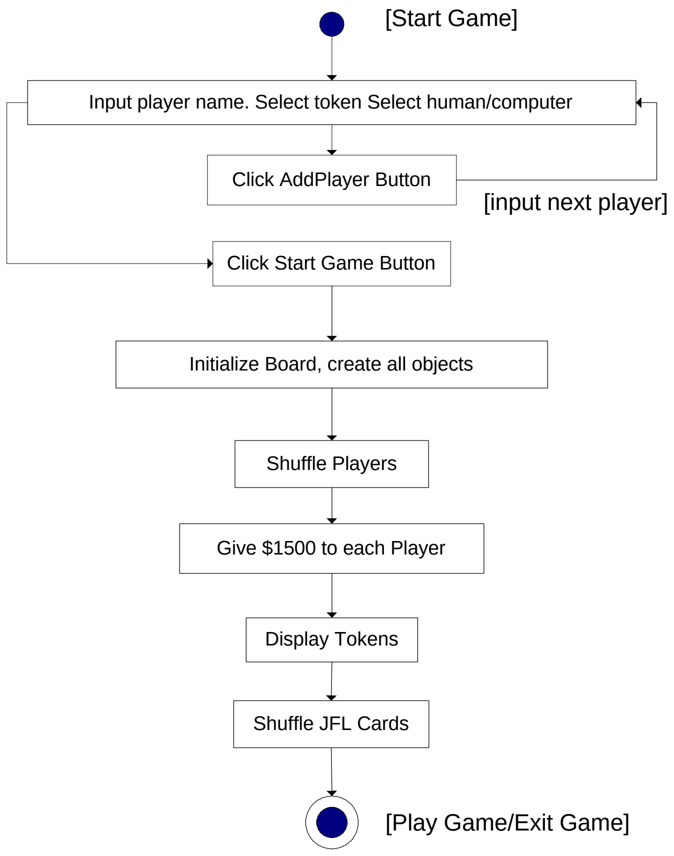
6.1 Start Game Scenario

Game start scenario contains several steps.

- The main functionality is to input the player information, initializing the game.
- All the actions in this scenario are in time order.
- The details of the action are: (see the following figure)
  0) Display the “Start a game interface”
  1) Input player name(s);
  2) Select tokens of each players;
  3) Input cyber/human player;
  4) Initial the board, create all the objects, set all the prices and initialize variables like the time delay, the tax rate, rent rate, the mortgage/unmortgage rate.
  4) Generate the random playing order,i.ie, shuffle the players.
  5) Assign $1500 each player.
  6) Display Tokens.
  7) Shuffle JFLCards.

The details of this scenario are shown in the section 3.3.1 as an example of the dynamic module interface.

Activity diagram Of the Montrealopol Game
(I)
Start Game



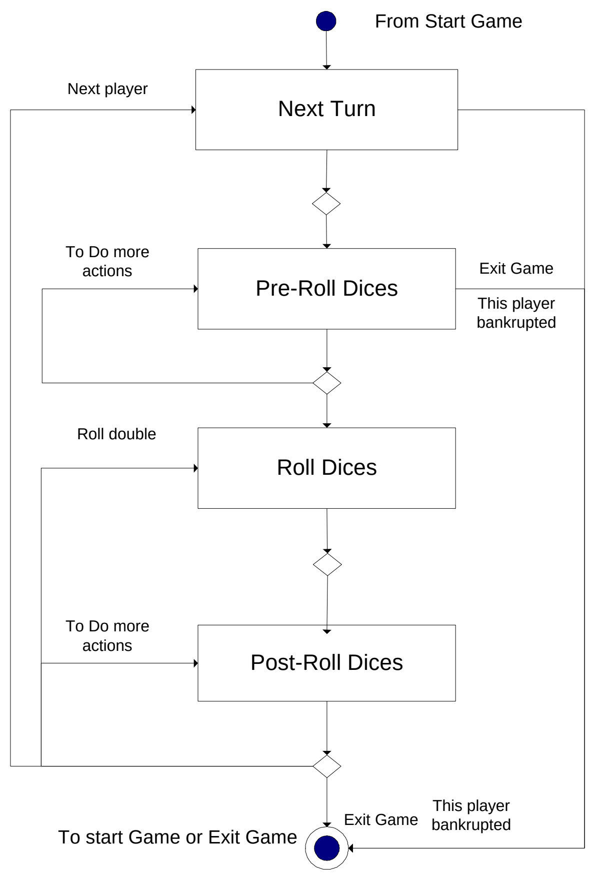
6.2 Play Game Scenario

Play game scenario consists of several scenarios since it is the core of the game and implements most of the requirements.

- The main functionality is playing the game and the detailed description of them will be in the following sections.
- All the actions in this scenario are in time order and logical order. Some of the action may not be done.
- The details of the action are: (see the following figure)
  1) The next turn (I.e., to pass the control to the next player.)
  2) The pre-roll dices
  3) The roll dices, and
  4) The post-roll dices.

- The transitions among the action are:
  1) Next turn to end game: A player is bankrupt, only one player left so the game is over.
  2) Pre roll dices to end game: the player click exit game button.
  3) Post roll dices to end game: either there is a winner or the player click exit game button.
  4) Post roll to roll dices: if a player rolls double, he will continues to roll the dices again.
  5) Post roll dices to post roll dices: a player can do many actions in this step.
  6) Post roll dices to next turn: control of the game is passed onto the next player.

General activity diagram of the Montrealopoly game (II) Play Game



6.3 Pre-roll Dices

This scenario contains all the activities that can be done before rolling the dices. The player can:

- Try to get out of the jail by using GOJFC card. Get out of jail means to change attribute. Finally, return the card back to the deck
- Pay $50 if the player is in the jail. If a player is in the jail he can pay $50 to go out of the jail if he does not have GOJFC card.
- Change options: the player may turn on or off the music, enable/disable the time delay.
- Display the help information. Similar to change option, this function is in the menu. Player can click the File of the menu and find the Exit item. Click it to exit the game.
- Exit game. This function is in the menu. Player can click the File of the menu and find the option item. Click any cell.

If a user clicks on a cell, he can do several actions:

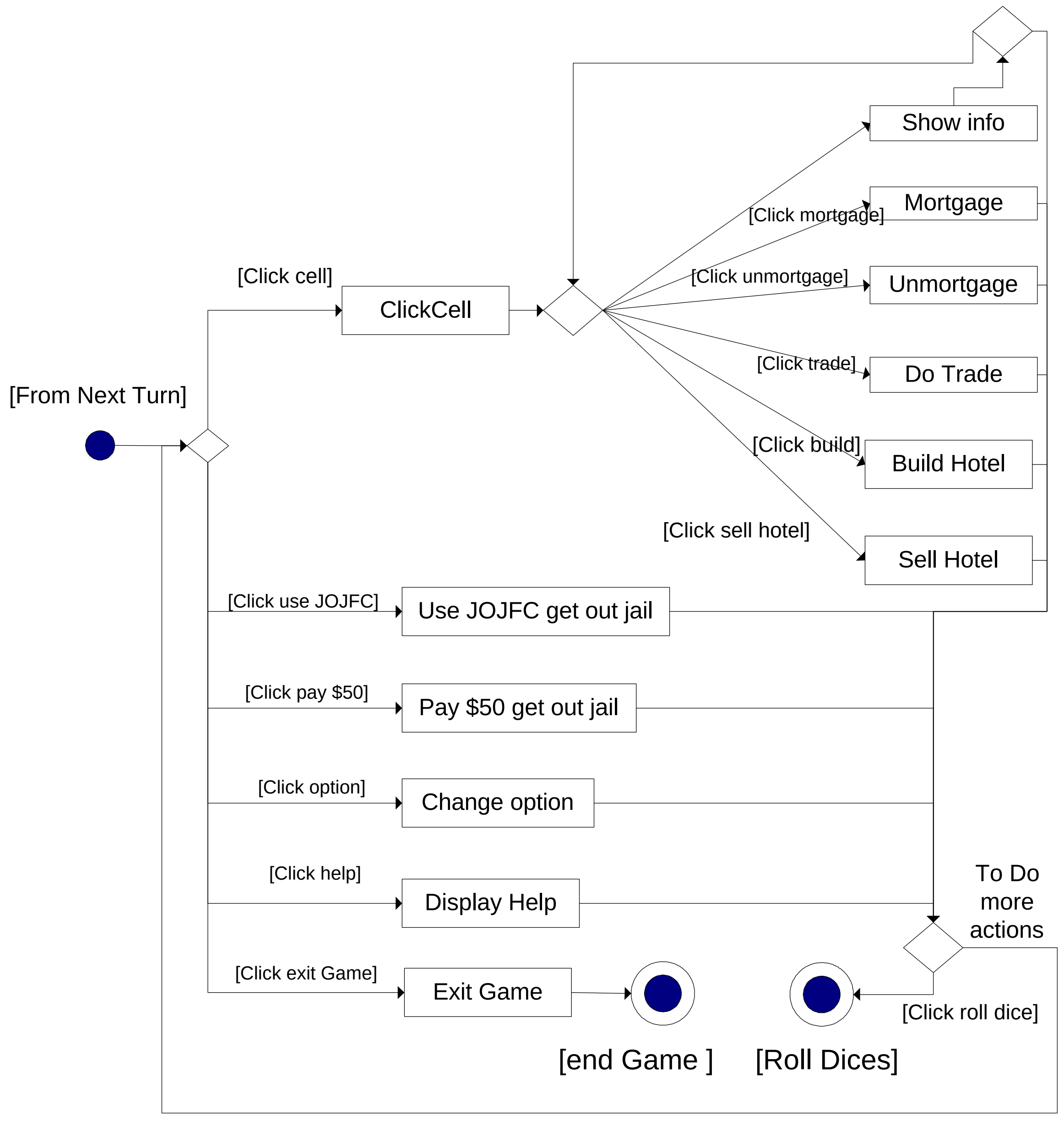
- The information of the cell will be displayed

Then a player can:

- Mortgage an unmortgaged property. If the property owner is the player and he did not mortgage the property, a mortgage button is displayed that enables the player get certain amount of cash to get it mortgaged. This changes the attribute of the cell.
- Unmortgage a mortgaged property. If the property owner is the player and if he mortgaged the property, an unmortgaged button is displayed that enables the player to pay certain amount of cash to get it unmortgaged. This changes the attribute of the cell.
- Build a hotel. If the street owner is the player and he did not mortgaged the street, a build hotel button and a sell hotel button is displayed that enables the player to pay a certain amount of cash to build a hotel.
  The interface of build hotel: See section 3.3.2.
- Sell a hotel. If the street owner is the player and he did not mortgaged the street, a build hotel button and a sell hotel button displayed that enables the player sell a hotel to get certain amount of cash.
  The interface of Sell hotel: See section 3.3.2.
- Click roll dice button will end this scenario to the next one.
- Do trade: click a cell that is owned by other player, an information window is displayed. There is a trade on the window that enable a player does trade. If the player clicks the trade button, a trade window is displayed. Two players can offer price accept price reject the trade.
  The interface of do trading, see section 3.3.5.

- Note all the windows have an ok button. A player can do nothing by click it. After do action, a player can click it to close the window.

Activity diagram Of the Montrealopoly Game (III) Pre Roll Dices



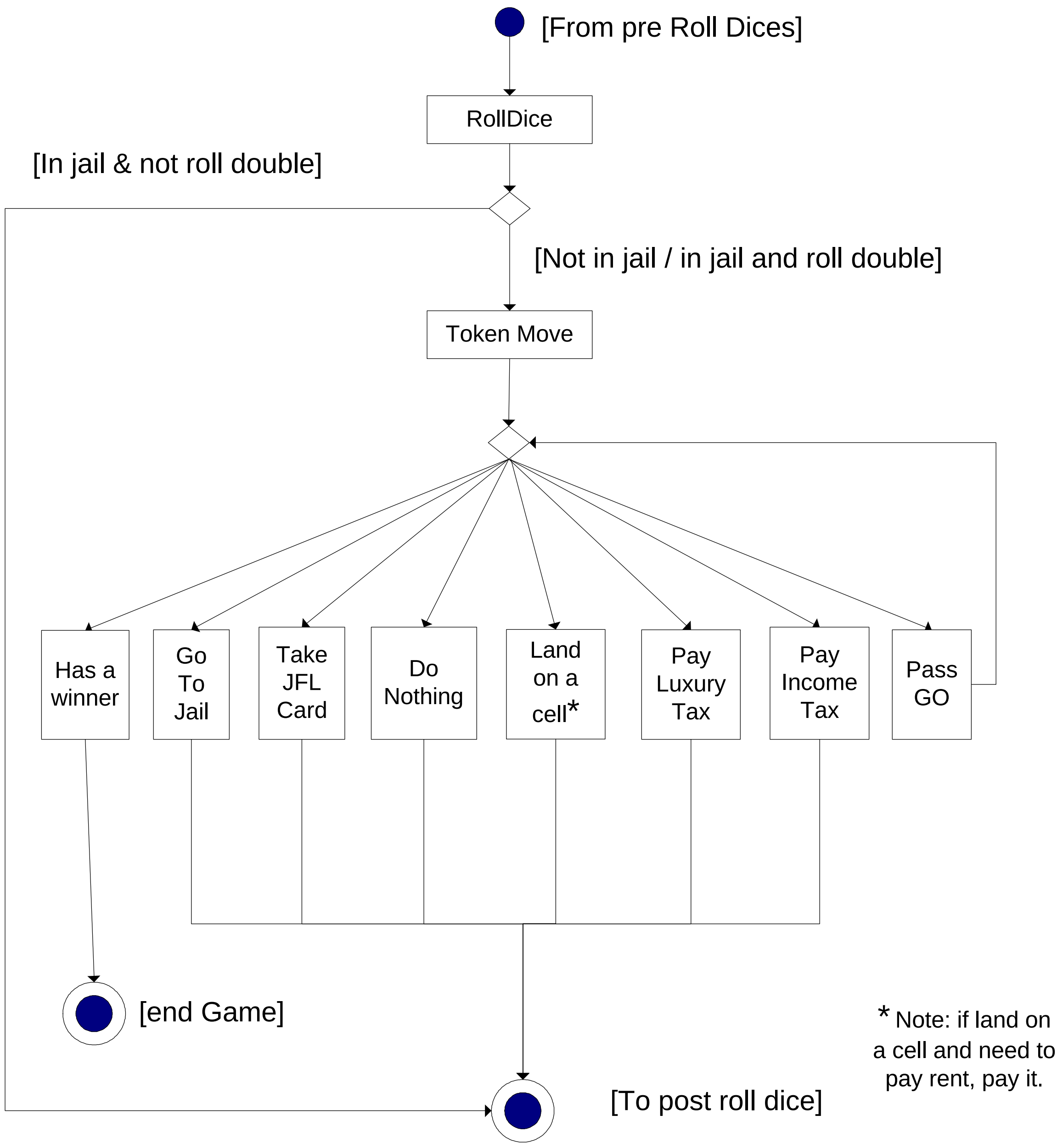
.

6.4 Roll Dices

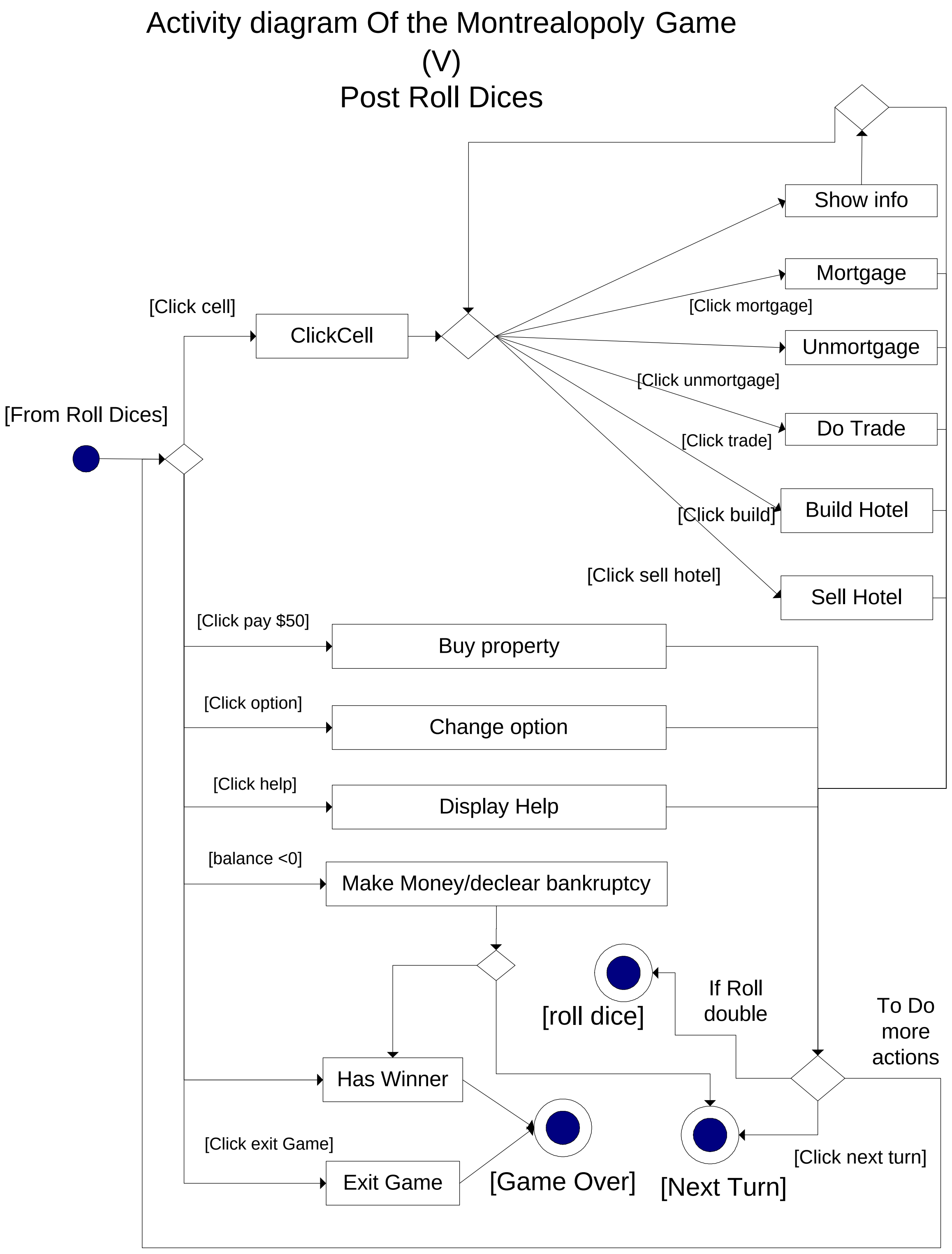
Starting with clicking the Roll dice button, this scenario has several actions that are done automatically by the program rather than by the player. The program, without the interruption of human player automatically perform these scenarios. (Seeing the next diagram.)

- Go to Jail: if a player lands on the go to jail cell, then the payer’s token will be move to the jail.
- Land on the Olympic Park, then he do nothing.
- Land on income tax cell or luxury cell, pay the tax by calling the doAction().
- Whenever pass the Go cell (or land on the Go cell,), the player collect $200.
- Land on the JFL card cell, the player will perform what is said on the card vi JFLCrad.DoAction(). However, if it is get out of jail free card, the player will keep the card.
- Land on a property owned by other player, he pays the rent.
- When player is bankrupt, the last player will be the winner.
- When balance is negative, the player will not allowed to make money now.

Activity diagram Of the the Montrealopoly Game (IV) Roll Dices



6.5 Post-roll Dices



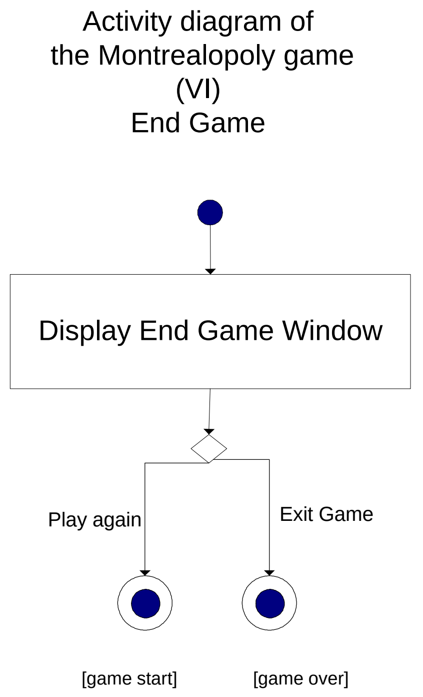
This scenario contains all the activities that can be done after rolling the dices. Most of the activities are the same as the pre roll dice scenario. The differences between the pre roll dice and post roll dice are:

The player can buy the property that he lands on if the property is not owned by any player.
There may be a winner since a player may fall into debt which he cannot pay.
A player can roll the dice again rather than pass the control to the next player.
After rolling the dice, a player can declare bankruptcy.

- Buy property/metro/utility/street
  If the player is computer player, call autoPlay() to use AI rule. Display buy property/metro/street frame, the player can press the confirm button to buy it (changes ownerID of the property, display players token on the cell) or do nothing.

For the interface of buy property function, see section 3.3.3.

6.6 End Game Scenario



Display the end game window if there is a winner and display two buttons that enable the player to play again or exit the game.