# CSP 301 Project - Designing *Monopoly* Video Game

Sahil Loomba (2012CS10114)

August 20, 2014

### Software Design Document

## Contents

# 1 Introduction

## 1.1 *Monopoly* - The Fast-Dealing Property Trading Game

**Monopoly** is a property-dealing family board game, for 2 to 8 players. The objective of the game is to become the wealthiest player in the game through strategised selling and buying of properties. This video game, henceforth referred to as the "game", would emulate the original boardgame[1], along with few additional features (see sec 1.2), allowing for bot players, multiplayer gaming, cheating moves and user-defined configurations for the gameplay.

In a traditional game, the players participate in simple game interactions in a round-robin fashion:

- Roll the dice and move their token the number of places shown on the dice.
- The player may land on a property slot, tax slot, chance slot, community chest slot or a special slot.
- For **Property**:
  - If the property they land on is unowned, then decide on buying it. If they decide to not buy it, property is held out for auction by the bank, for which all players are eligible to bid. If nobody bids, property remains unowned.
  - If the property they land on is owned by someone other than themself, then they must pay rent to the player to which the property belongs (depending on the rent calculations as per the colour set, number of houses/hotels, whether property is mortgaged or not, etc.).
- For tax slots (**Income Tax** and **Super Tax**), the player must pay the shown tax amount to the bank.
- For **Chance** and **Community Chest** slots, the player must pick the topmost chance/community chest card and follow the one-step instruction on the card.
- For special slots:
  - **Go**: The player receives their salary on passing Go.
  - **In Jail/Just Visiting**: The player lands up in this slot when
    - thrown into Jail by a chance/community card or the "Go To Jail" slot.
    - just visiting, that is, do nothing.
  - **Free Parking**: Do nothing.
  - **Go To Jail**: The player must move to the In Jail slot.

Beyond their own chance, the players' interactions with the game could be restricted to:

- Receiving rent when others land on their property.
- Bidding for unowned property.
- Mortgaging/un-mortgaging their property.
- Paying to bank/player (owing to a certain chance/community chest card which some other player picks up).
- In multiplayer mode,
  - reporting cheating and denote punishment for the cheater/reward for the cheated.
  - chatting with other human players within the game's interface.
  - barter trade properties with other human players.
- Viewing
  - the current game statistics like player positions, number of sold/unsold properties, etc.
  - other players' profiles showing their currently owned (mortgaged or otherwise) properties, number of houses and hotels, etc.
  - the title deeds of all properties, along with their current owners.

[1] The complete official rulebook can be found at `http://www.hasbro.com/common/instruct/monins.pdf`

## 1.2 Game's Technical Overview

The game will be written in C++, using the OpenGL library for 2D & 3D rendered graphics. The game will be played on a traditional PC, but it may be extended to cross-platform mobile devices. The game would include a multiplayer feature, allowing players to connect through the internet, using the SDL library.

### 1.2.1 Control Scheme

To keep the user-game interaction simple, the player would mostly control the game through mouse clicks. This includes functions of:

- Selecting and moving objects (tokens, houses/hotels, dice, chance and community chest cards) around the board.
- Selecting commands to execute: "Pay", "Bid", "Undo Last Payment", "End Turn", "View", "Trade", "Mortgage", "Report Cheating", "Main Menu", etc.
- Selecting currency and players/bank to make payments to.
- Viewing player profiles, title deeds, game progress, etc.

Keyboard-input is required from user only when:

- Entering the player name.
- Chatting with other (human) players in the chat window.

### 1.2.2 Setting up the Game - *config* files and *save* files

The game starts after loading its default configuration file. But to make things more modular and customisable, the user may change the game setup by creating their own config file (allowed only in single-player mode). Also, the user has an option to save a game's progress in a save file (allowed only in single-player mode), and load the game at another time. Therefore, when the game is first begun, the initial configurations are read out from the selected config and/or save files.

**Config File Structure** The configuration file consists all the information necessary to setup the game. It is loaded each time the game is started and contains fields to allow setup of:

1. **The Board**
   (a) Define the structure of the board by providing the number of slots (traditionally, 40 slots).
   (b) Define each slot of the board, starting with the **Go slot (mandatory)**. In order, for each slot, clearly prescribe the slot name and slot type.
      i. "In Jail/Just Visiting", "Free Parking", "Go To Jail" - Slot Type 'x'
      ii. "Chance", "Community Chest" - Slot Type 'c'
      iii. "Income Tax", "Super Tax" - Slot Type 't'
      iv. User-defined properties - Slot Type 'p' **(slot name must match with a unique property name)**
      v. User-defined utilities (also a kind of property) - Slot Type 'u' **(slot name must match with a unique utility name)**
      vi. User-defined stations (also a kind of property) - Slot Type 's' **(slot name must match with a unique station name)**
2. **The Properties, Utilities and Stations**
   (a) For properties, define the number of colour groups and number of properties in each colour group. Then, prescribe the
      i. property name, purchasing value, mortage value, group colour
      ii. rent on site, with 1/2/3/4 houses, with hotel
      iii. cost of each house and hotel

(b) For utilities, define the number of utility pairs and numeric codes for each pair. Then, prescribe the

i. utility name, purchasing value, pair code

ii. rent if singly owned, rent if pair is owned (for example, rent is 4 times the number shown on dice if only Water Works is owned; if Electric Works is also owned, then rent is 10 times)

(c) For stations, define the number of stations. Then, prescribe the

i. station name, purchasing value

ii. rent if singly owned, doubly owned, triple owned, ... (depending on number of stations)

(d) Optionally, image files can also be associated to be displayed on the title deed of the property.

3. **The Players**

(a) Define the number of players, and their player type (Player Type 'h' for human, 'b' for bot).

(b) Define each player, starting with self. In order, for each player, prescribe the player name, initial amount of currency, token colour.

(c) Optionally, an image file can be associated with each player to be displayed on the player's profile.

4. **The Currency**

(a) Define the number of currency notes, and the value of each currency note.

(b) Optionally, an image file can be associated to be displayed on the currency note.

5. **Chance and Community Chest cards**

(a) Define the number of chance and community chest cards.

(b) Prescribe the card name and card type for each one (Card Type 'c' for chance, 'cc' for community chest).

(c) Provide an Action Type code for each card:

i. 'j' - Go To Jail Card

ii. 'nj' - Get Out of Jail Card

iii. 'pb' - Pay $x$ amount to bank

iv. 'rb' - Receive $x$ amount from bank

v. 'pp' - Pay $x$ amount to each player

vi. 'rp' - Receive $x$ amount from each player

vii. 'hh' - Pay $x$ amount for repair of each house, hotel

viii. 'ad' - Advance to $y$

6. **Cheating Penalties**

(a) Define number of penalties.[2]

(b) Prescribe the penalty name and penalty type for each one:

i. 'p' - Cheater pays $x$ amount to the reporter

ii. 'pp' - Cheater pays $x$ amount to each player

iii. 'j' - Cheater goes to Jail

iv. 'bp' - Reporter receives $x$ amount from bank

v. 'q' - Cheater goes bankrupt

vi. 'pr' - Reporter receives (unmortgaged) property $y$ from cheater (along with any houses/hotels)

(c) Define penalty on reporter for a false accusation (reporter pays $x$ amount to bank).

[2]Note that cheating, if caught by another human player, has inevitable penalties for which the cheater would be automatically charged by the game server.

**Save File Structure** The save file stores the last saved state of a game. They are loaded only if the user asks for continuing a previously stored game. Note that they do not store the whole progress of the game, but **only the last state**. A save file must contain just enough information, including:

- Name of the config file and of the single human player.
- Positions of each token, house, hotel.
- Player name, whose turn it is.
- Amount of money with each player.
- Status of each title deed (owned/unowned, mortgaged/unmortgaged), and mappings to their owners.
- Players holding Get Out Of Jail cards.

# 2 System Architecture

The whole program is modular in structure, with specific tasks allocated to different modules, so as to acheive the complete functionality of the system. Before looking deep into the subsystems and how they are related, we must look at the flow of the gameplay to remark on how the game proceeds.

## 2.1 Game Flow Description

### 2.1.1 Outside the *Game Arena*

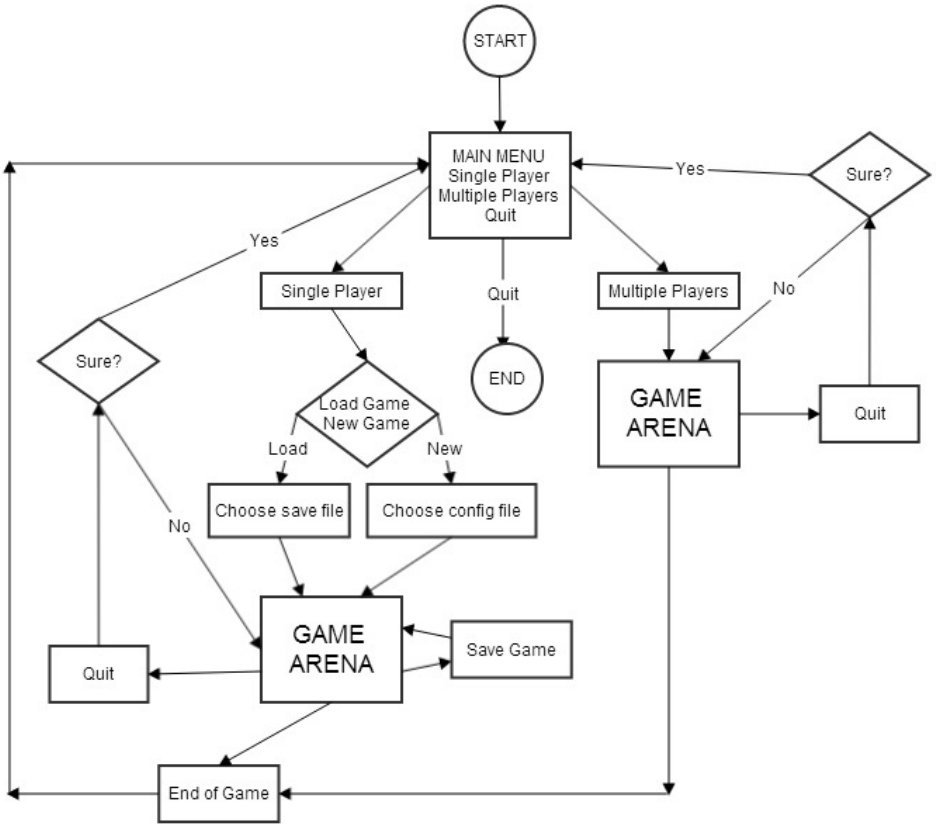


Figure 1: Game Flow Diagram for outside of the Game Arena

### 2.1.2 Inside the *Game Arena*[3]

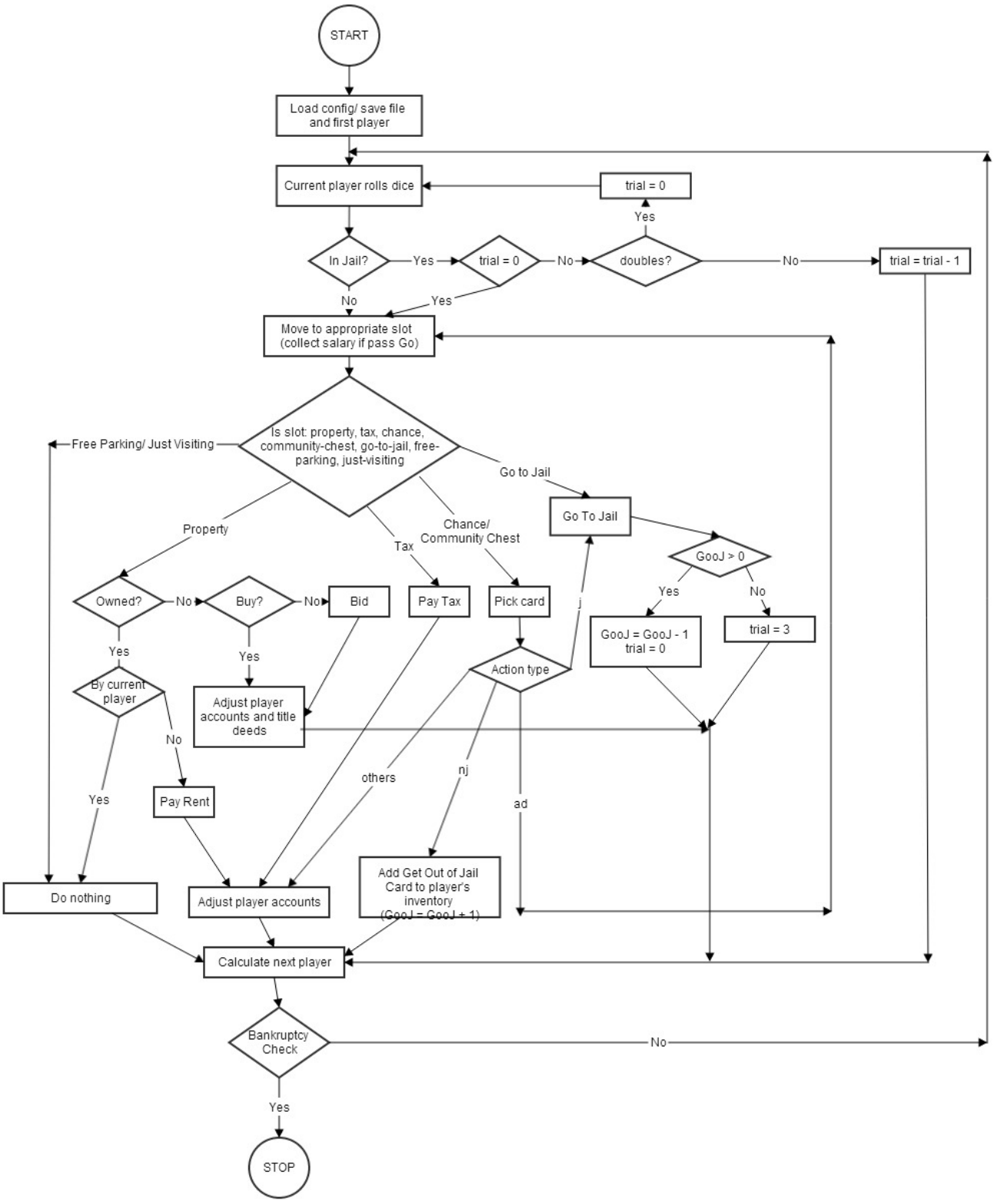


Figure 2: Game Flow Diagram for inside of the Game Arena (main gameplay)

[3]Without incorporating for "cheating", "chatting" and other interruptions.

## 2.2 Game Modules

As clearly depicted in the above flow diagrams, the game design would be divided into the following modules:

**MainMenu** module will describe the outermost game layout, and incorporate methods to load config/save files.

**FileParse** module will allow parsing of config/save files.

**SaveGame** module will allow the user, in the single-player mode, to save the current game session.

**Multiplayer** module will allow users to connect over the internet for a default-configured game.

**GameArena** module will run the actual game, incorporating 3D graphics.

**Chatting** module will display the progress of the game after each move, and also allow users to type and communicate messages to each other in the multiplayer mode.

**Cheating** module will permit users to report cases of cheating, and for the server to verify such cases and to award relevant punishment.

**Special** modules will account for each of the auxilliary commands available to the user: viewing rules, viewing status window, viewing properties, viewing players, bidding, trading, mortgaging, undoing and quitting.

# 3 Data and Component Design

The game's implementation is completely driven by object-oriented programming. Every game module works on cohesive functioning of relevant components. The components, in turn, are instances of classes (data structures). The classes are further characterised by various attributes and procedures.

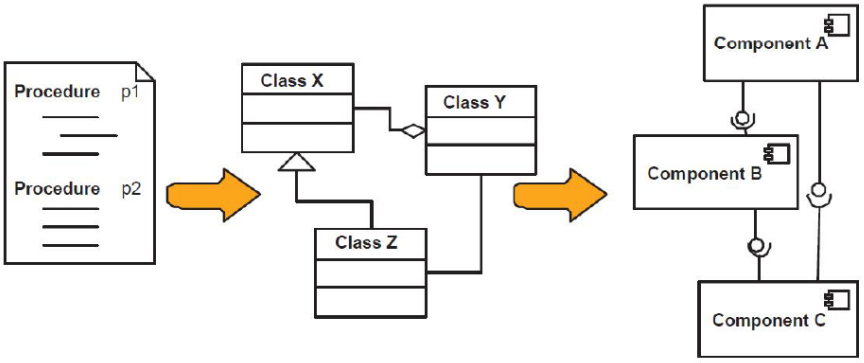


Figure 3: Stages of Component Design

The data structure and basic procedures related to some key game components are given below:

**The Board** will be visualised as a circular list of various Slot instances. Its attributes include the **head**, the **number of slots**. Its procedures include instantiate, pan, etc.

**The Slots** will be structures with attributes such as the **Slot Type** and **Slot Name**.

**The Properties, Utilities and Stations** will have attributes such as the **Property Type** and **Property Name**, purchasing value, mortgaged/not mortgaged, along with other characteristics depending on the kind of property (see sec 1.2.2 for more information).

**The Chance/Community Chest Cards** will have attributes like **Card Type** and **Action Type**, card instructions and other attributes depending on the kind of action (see sec 1.2.2 for more information).

**The Players** will have attributes like **Player Type** and **Player Name**, token name, amount of currency, number of properties, list of property names, number of get out of jail cards, number of trials, bankruptcy status, etc.

**The Tokens** will have attributes like **Token Name**, player name, current position on the board and procedures like **move**.

**The Houses** will have attributes like **House Type,** player name, property name, current position and procedures like **move**.

**The Dice** will have attributes like **last value**, and procedures like roll, appear/disappear.

**The Currency** will have attributes like **value.**

# 4 User Interface Design

## 4.1 UI Design Samples


Figure 4: Game's imagined main user interface screen

## 4.2 Screen Objects and Actions

### 4.2.1 The Menu Bar

It is the topmost bar on the main game screen, consisting of various less-often used commands/actions like:

RULES to open the Rules Window, where the player can view the rules of the game.

STATUS to open the Status Window, where the player can view the overall status of the game (the properties of each player, number of sold/unsold properties, etc.)

PROPERTIES to open the Properties Window, view the title deed cards of all properties, stations and utilities (owned or not-owned), including their mortgaging status.

BID to open the Bidding Window, where interested players can place bids on unowned properties when the server announces so in the Notifications.

TRADE to open the Trading Window, wherein players can barter exchange properties with each other.

QUIT to quit from the current game, which pops a Quit Notification confirming the quit.

### 4.2.2 Sliding Menus

The right side of the screen is reserved for sliding between three frequently seen and used entities:

**MyProperties** consists of all the properties (and any get-out-of-jail cards) owned by the player. The player can click on any title deed and use the **View** command to view its attributes.

**AllPlayers** consists of all the players in the game and the Bank. It highlights the player who's currently in play. The user can click on a player to make payments using the **Pay** command. The user can also view player profiles by using **View.**

**MyAccount** shows all the currency notes that the user is in possession of, and displays the total amount of money in the account. The user can make payments by using the **Pay** command.

### 4.2.3 Push Buttons

The centre console displays 6 push buttons:

PAY to dispatch the payment of the selected notes to the selected player/bank.

DONE to indicate end of turn so that the game can proceed to the next player.

UNDO to undo the last payment made (active only till the player doesn't commit by pressing Done).

MORTGAGE to mortgage the selected property.

VIEW to view information about the selected player/property.

REPORT to report the last player's cheating (only for reporting the latest player).

### 4.2.4 The Board and the Chat Console

The Board is located in the top-left area of the game screen. It also includes the dice in the centre of the board, and the chance and community chest cards on the opposite corners of the board. When a player clicks the dice to roll them, the board zooms in to show the rolled numbers, and zooms out to allow the player to make his move (by clicking, dragging and releasing his token). The player can always click around the board to zoom in and pan across the board. (The design of the board would not be the conventional Monopoly board-game, but rather spread out like a road circuit across cities.) He can click on slots to view the title deed of the property positioned on that slot (which would be popped up as a "Notification"; see Fig 4). If the player moves to chance/community chest, the player can click on the appropriate card deck and the topmost card would flip and pop up to reveal the message (again, as a "Notification").[4]

[4]Notifications are small windows that overlay the board area and are popped up as required, say to announce bids, to pop up title deeds of property the player arrives at, to pop up the chance/community chest card message, to inform the amount of money the player must pay to another player for landing on their property, etc.

The Chat Console keeps displaying status messages after each move of the game. This is important for players to keep track of who's cheating in the game. Also, in multiplayer mode, it allows for users to communicate with the other players by submitting text to the Chat Console.

# 5 Testing Strategy

The game would be tested in a three-tier fashion:

1. **Code-level testing:** Ensure accurate working of each procedure and function, by testing with various parameters, before finalising the code block. Includes checking for branches, loop conditions, catching exceptions, etc.

2. **Module-level testing:** Test each module independently before assembling them together to form the complete game.

3. **Game-level testing:** Play the game with various configurations and over possible conditions, to exhaust most gameplay possibilities. Test the overall quality of play, speed of execution, etc.