**COMP3211 Software Engineering**

**Group 20**

**The Monopoly Game**

**Design document**

**Wong Keng Wa Kovey 24018054D**

**Tsang Cheuk Yu 24025237D**

**Chan Ho Lok 24032915D**

Content

[1. The Monopoly Game Architectural Diagram 3](#_Toc183125562)

[2.1 Structure of and relationship among main code components: 5](#_Toc183125563)

[2.2 Class diagram 7](#_Toc183125564)

[3. Collaboration among the code components 8](#_Toc183125565)

# 1. The Monopoly Game Architectural Diagram

**一張含有 文字, 螢幕擷取畫面, 字型, 圖表 的圖片

自動產生的描述**

**Figure 1: MVC Architectural Patterns**

**MVC Architectural is picked for the Monopoly game because there are multiple ways to view and interact with data.Also, This gives a higher Scalability in the game, allowing developers to Add new Features easily.**

1. Model (game logic and data)

This model represents core game mechanics and statistics. It saves the state of the game, including the game board, player positions, attributes, money, and other game rules.

Responsibilities:

• Manage game data (e.g., current state of the board, player status).

•Store and update attributes, player positions, and other game-related objects (e.g. board, players, blocks, attributes).

• Implement the rules of the game, such as purchasing properties, collecting rent, and determining winner conditions.

• Provides methods for manipulating and retrieving game data (eg, buyProperty(), rollDice(), movePlayer()).

• Handle persistence (for example, saving game state to a file or loading game state from a file).

2. View (user interface)

Views are responsible for displaying game data to the user and presenting any information needed to interact with the game.

Responsibilities:

• Present the current game state to the user (e.g., display the board, player's money, possessions).

• Display user input prompts (for example, "Press '1' to roll the dice").

• Process output from the console (or GUI in more advanced versions).

3. Controller (user input and interaction)

The controller manages user input, updates the model based on user actions, and coordinates the interaction between the model and the view.

Responsibilities:

• Accept user input (eg, move a player, buy a property).

• Update the model based on input (e.g., modify the board, update player profiles).

• Notification views update their display based on changes in the model.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Class** | **METHOD** | **ARGUMENT**  **NAME AND**  **TYPE** | **RETURN TYPE** | **EXCEPTION**  **DECLARATIONS** | **EXPLANATION** |
|  | **main** | **args (String array)** | **void** | **N/A** | **The entry point of the program. Initializes the Board object and starts the main menu for the game. Serves as the controller for initiating game flow.** |
| **App** | **showMainMenu** | **N/A** | **void** | **None** | **Displays the main menu and handles user input to either start a new game, load a game, customize the board, or exit. Contains input validation and loops until valid commands are given.** |
|  | **turnFileToString** | **fileName (String)** | **String** | **Handles IOException for file reading errors.** | **Converts the content of a file into a String. It reads the file line by line using a buffered reader and appends it to a StringBuilder.** |

# 2.1 Structure of and relationship among main code components:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **loadBoard** | **fileName (String)** | **void** | **N/A** | **Loads the board configuration from a specified JSON file. The file contains the state of the game board.** |
|  | **createBoard** | **N/A** | **void** | **N/A** | **Initializes a default game board with predefined properties and positions. This method is used when starting a new game with default settings.** |
| **Board** | **editPosition** | **position (int)** | **void** | **N/A** | **Allows modification of a specific position on the board. It is part of the board customization feature.** |
|  | **printBoard** | **None** | **void** | **N/A** | **Displays the current state of the board, showing positions and properties. This is a utility for debugging and gameplay.** |
|  | **saveBoard** | **fileName (String)** | **void** | **N/A** | **Saves the current state of the board to a JSON file for later retrieval.** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **controller** | **startGame** | **numOfPlayer (int)**  **saveData (JSONObject)**  **savename (String)** | **void** | **N/A** | **Starts the game with the specified number of players and board configuration. It initializes players and manages game progression using the given save data or a new game setup.** |
|  | **getType** | **N/A** | **String** | **N/A** | **Returns the type of the square. The type represents what kind of square it is (e.g., property, tax, jail, etc.).** |
|  | **setType** | **type (String)** | **void** | **N/A** | **Sets the type of the square. The type determines the behavior or function of the square during the game.** |
|  | **getPosition** | **N/A** | **Int** | **N/A** | **Returns the position of the square on the board. Position is used to identify where the square is in the sequence of the game board.** |
| **Square** | **setPosition** | **Position** | **Void** | **N/A** | **Sets the position of the square on the board. This helps in initializing or modifying the square’s location.** |
|  | **setPlayerHere** | **p (int)** | **Void** | **N/A** | **Marks that a player (identified by p) is currently on this square. Updates the player array to indicate the presence of the player.** |
|  | **getPlayerHere** | **N/A** | **Int[]** | **N/A** | **Returns an array representing the players currently on this square. Each element corresponds to a player (0 if absent, 1 if present).** |

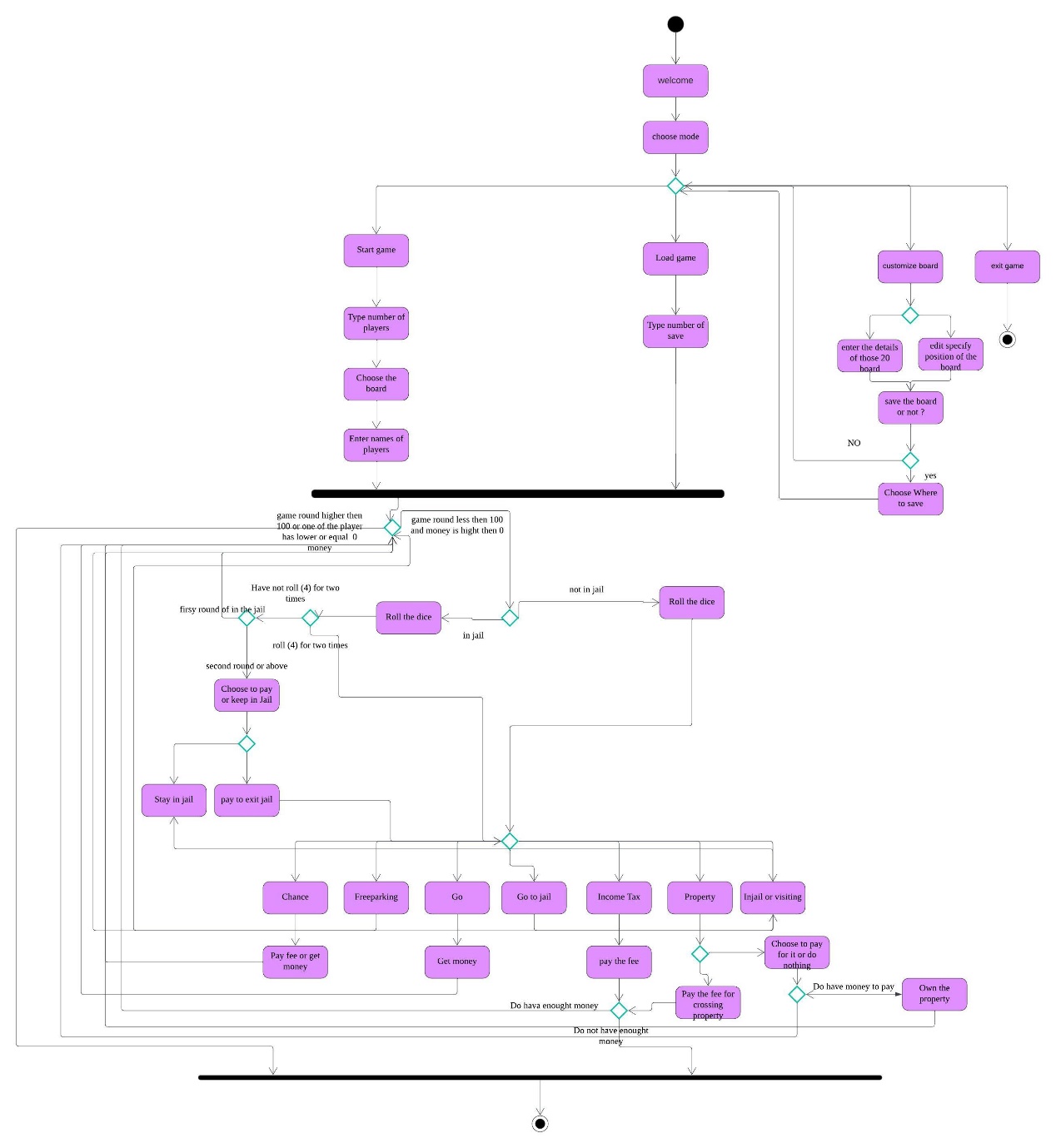
# 2.2 Class diagram

**一張含有 文字, 寫生, 圖表, 圖畫 的圖片

自動產生的描述The following Class diagrams show the classes of major code components in a system and the relationships between these classes. Including the classes App, Board, Chance, controller, Dice, FreeParking, Go, GoToJail, IncomeTax, InJailOrJustVisiting, Player, Property, and Square.**

**Figure 2: UML Class Diagram of the Monopoly game**

# 3. Collaboration among the code components

****

**Figure 3: UML Activity Diagram of the monopoly game**

**The above activity diagram outlines the sequence of activities that occur during gameplay, emphasizing the flow of decisions and interactions between players and the system. The diagram begins with the welcome pages, where players can choose to start a game, open a save or customize a board. When the game is started, the game enters a loop where each player's turn is processed sequentially. A player starts their turn by rolling the dice, determining their movement on the board. Based on the square landed upon, decisions are made whether to buy a property, pay rent or follow specific rules (e.g., going to jail). These decisions branch into subsequent activities, such as updating player balances, property ownership, or moving to specific board locations.**