COMP3211

The Monopoly Game

Created By:

Group 8

Overview

01

GAME USER INTERFACE

User commands, game status displays, and error handling mechanisms in the game.

02

GAME ARCHITECTURE

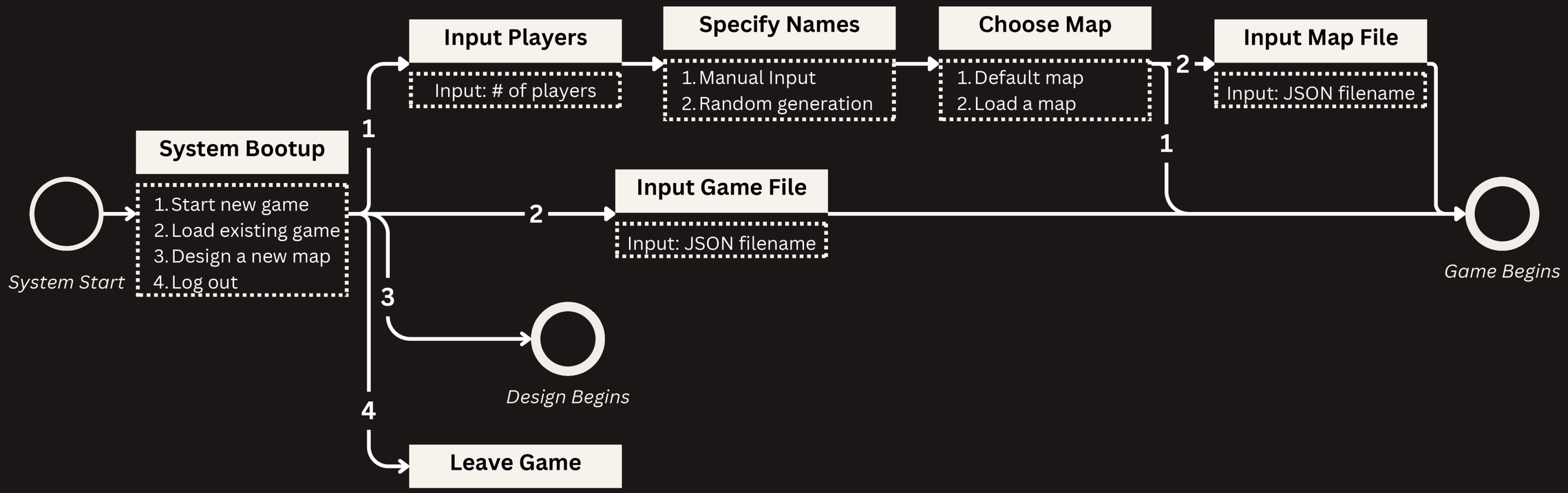
Explanation of the architectural pattern chosen, key components, and their interactions.

03

IMPORTANT LESSON

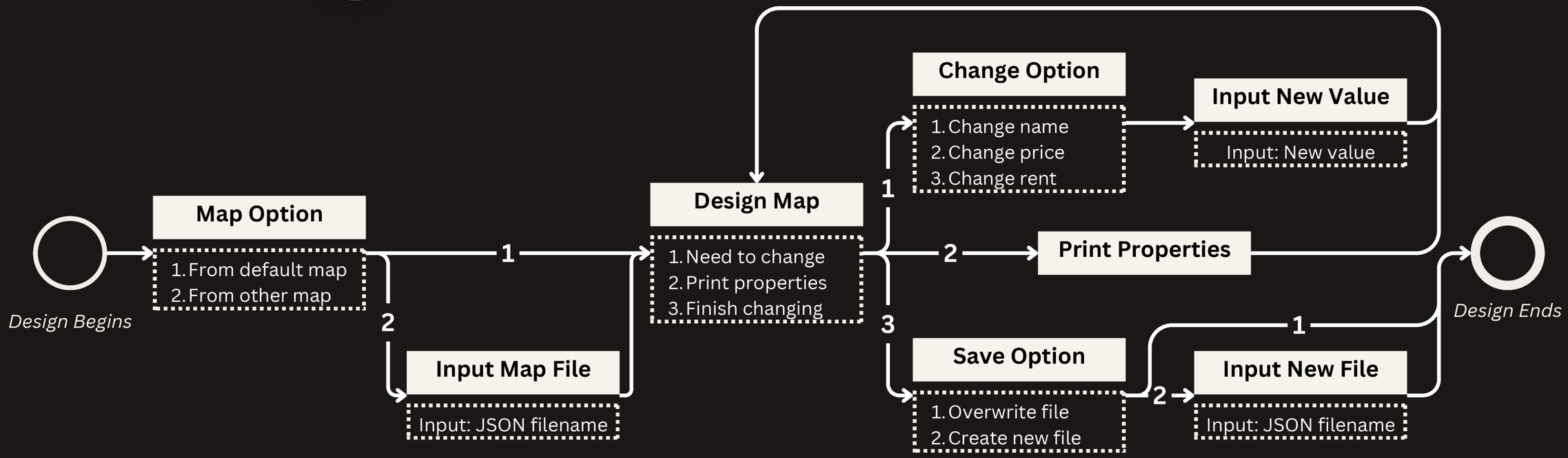
Insights related to unit testing during the project.

System Bootup



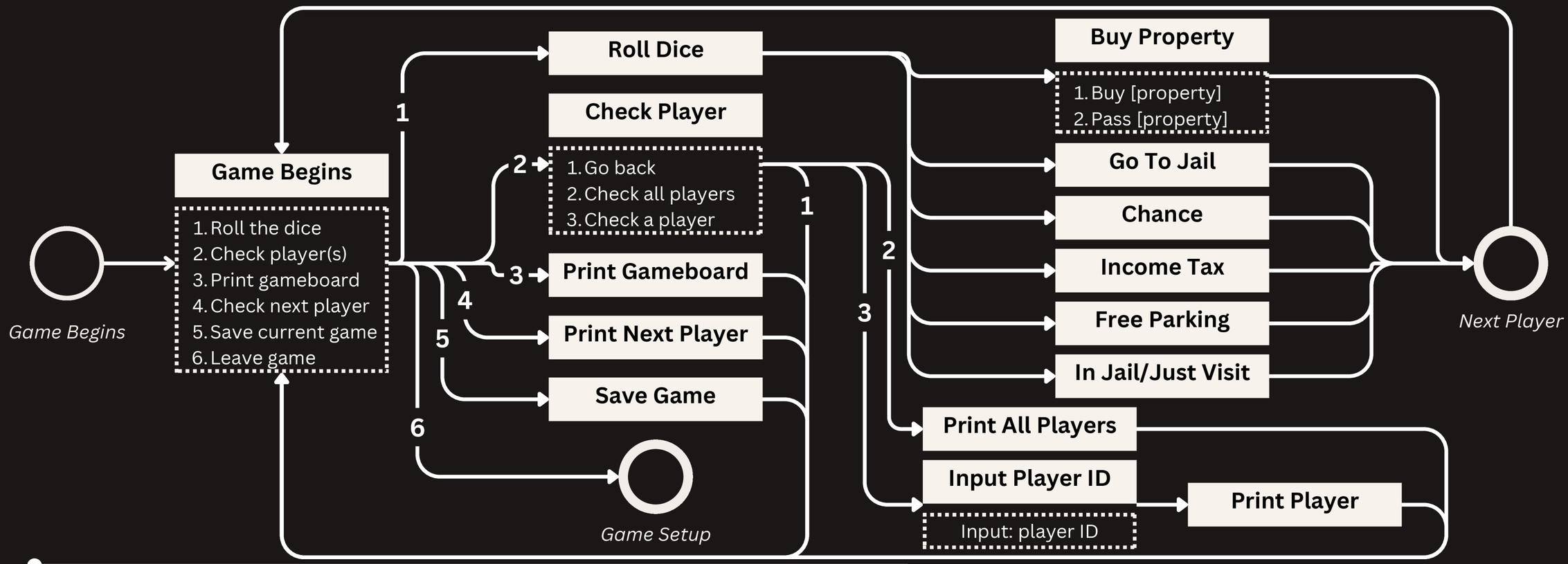
*Type out the numbers to proceed

Designer Commands



**Type out the numbers to proceed*

Game Commands



*Type out the numbers to proceed

Output

Gameboard

This represents the current state of the Monopoly board.

Player(s) Status

This provides a summary of each player’s current state in the game.

Next Player

This prints the details of the player playing in the next turn.

Error Handling

01

Input Handling for Player Deletion

- **Empty Name Check**: Ensures a player name is specified
- **Non-Existent Player**: Checks if the name exists

02

Input Handling for Player Addition

- **Name Length Check**: Ensures the name is under a valid length
- **Player Slot Availability**: Checks if limit has been reached

03

Input Handling for New Game Setup

- **Valid Number of Players**: Ensures it is between 2-6 players
- **Empty Player Slots**: Checks if all player slots are filled
- **Map Loading Validation**: Ensures map is loaded properly

04

Game End Handling

- **Multiple Winners Handling**: If more than one winner exists, constructs and displays a message listing all winners.

05

General Input Validation

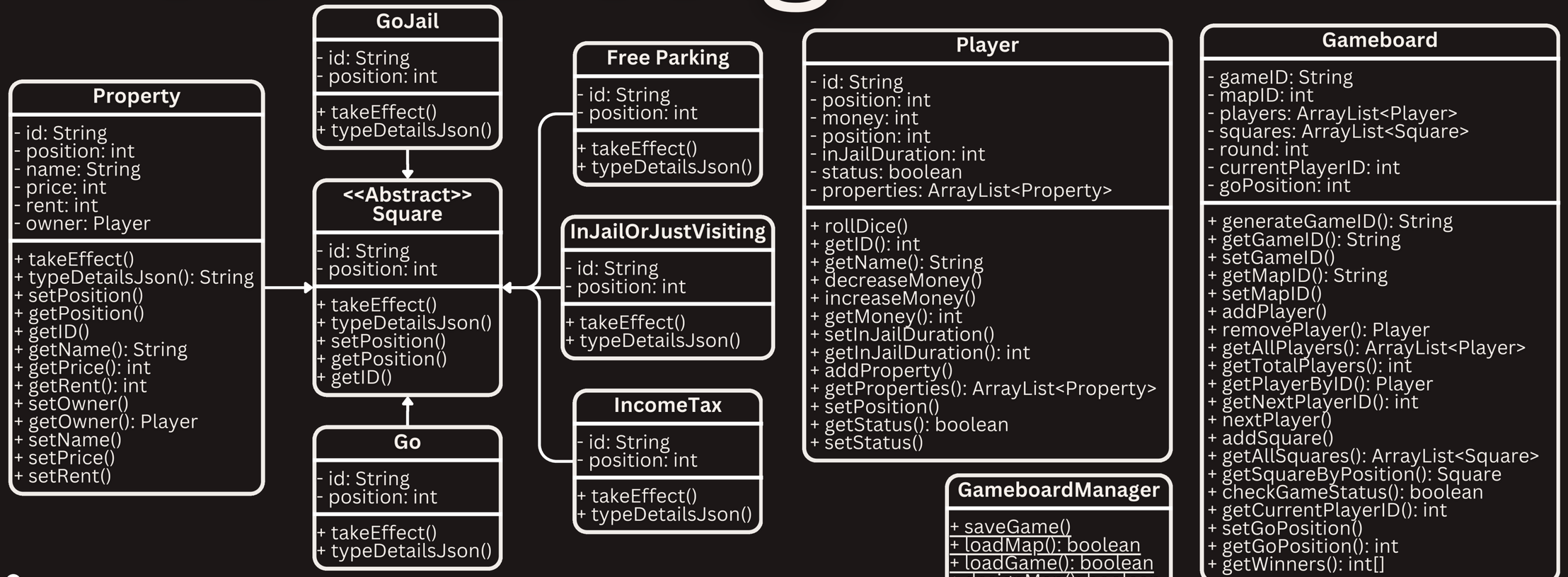
- **Prompts with Limited Options**: The InputView.inputPrompt method enforces that user input matches predefined options

06

Main Game Loop

- **Turn Options Handling**: Ensures it is between 2-6 players
- **Player in Jail**: Checks if all player slots are filled
- **Negative Money Check**: Ensures map is loaded properly

Model Design



View & Controller

GameboardView

- boardString: String

+ displayAllPlayers()
+ displayPlayer()
+ displayGameboard()
+ updateGameboard()
+ replaceBlockBySquare()
+ replaceLineByItem(): String

InputOutputView

- scanner: Scanner
- SIMULATED_INPUT: String

+ setInput()
+ promptInput(): String
+ promptFilename(): String
+ displayMessage()
+ displayAllProperties()
+ displayProperty()
+ promptString(): String
+ promptInteger(): int

GameboardController

- gameboard: Gameboard
- gameboardView: GameboardView

- getNumberOfPlayers(): int
- getNamingOption(): int
- displayCurrentPlayers()
- collectPlayerNames(): ArrayList<String>
- generateRandomNames(): ArrayList<String>
- chooseAndLoadMap(): boolean
- initializePlayers()
- checkPlayerStatus()
- displayGameboardStatus()
- displayNextPlayer()
- saveGame()
- handlePlayerOptions(): int
- handlePlayerMovement()
- checkPlayerBankruptcy()
+ newGame(): boolean
+ startGame()
+ endGame()
+ designMap()

Architecture

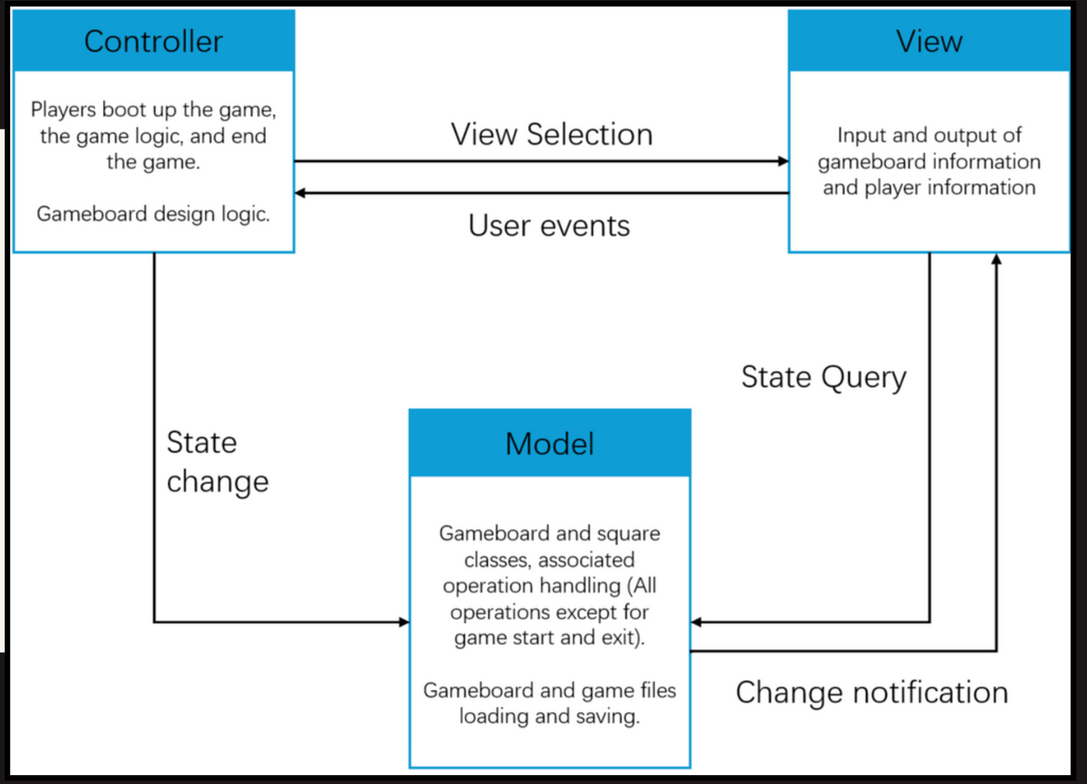
The MVC architecture for a Monopoly supports

Separation of Concerns
The game logic, user interface, and input handling are distinct.

Maintainability
Debugging and updating the code becomes more efficient.

Reusability
Components can be reused across different implementations.

Flexibility
Changing how the game is displayed without altering the core.



Lesson Learned

01

Code-Then-Test

At the early stage of our agile development, we developed one component followed by the unit tests of it. However, this introduced problems:

1. Delay of test development;
2. Changing interface conflicts with tests.

02

Test-Driven-Development (TDD)

When we realized the problems and applied TDD, we benefited from this method:

1. Test first, thus no test-lag problem;
2. The tests implicitly specified the interface and functionality of components.

Contact Us

+852 6475 3741

monopolyu@gmail.com

The Hong Kong Polytechnic University