MonoPolyU

Design Document

The Monopoly Game

Authored by:

- Dominicus Dylan HARYOTO
- Haoran BAI
- Tao FU
- Weichen NING

© 2024 MonoPolyU. All rights reserved.

Introduction

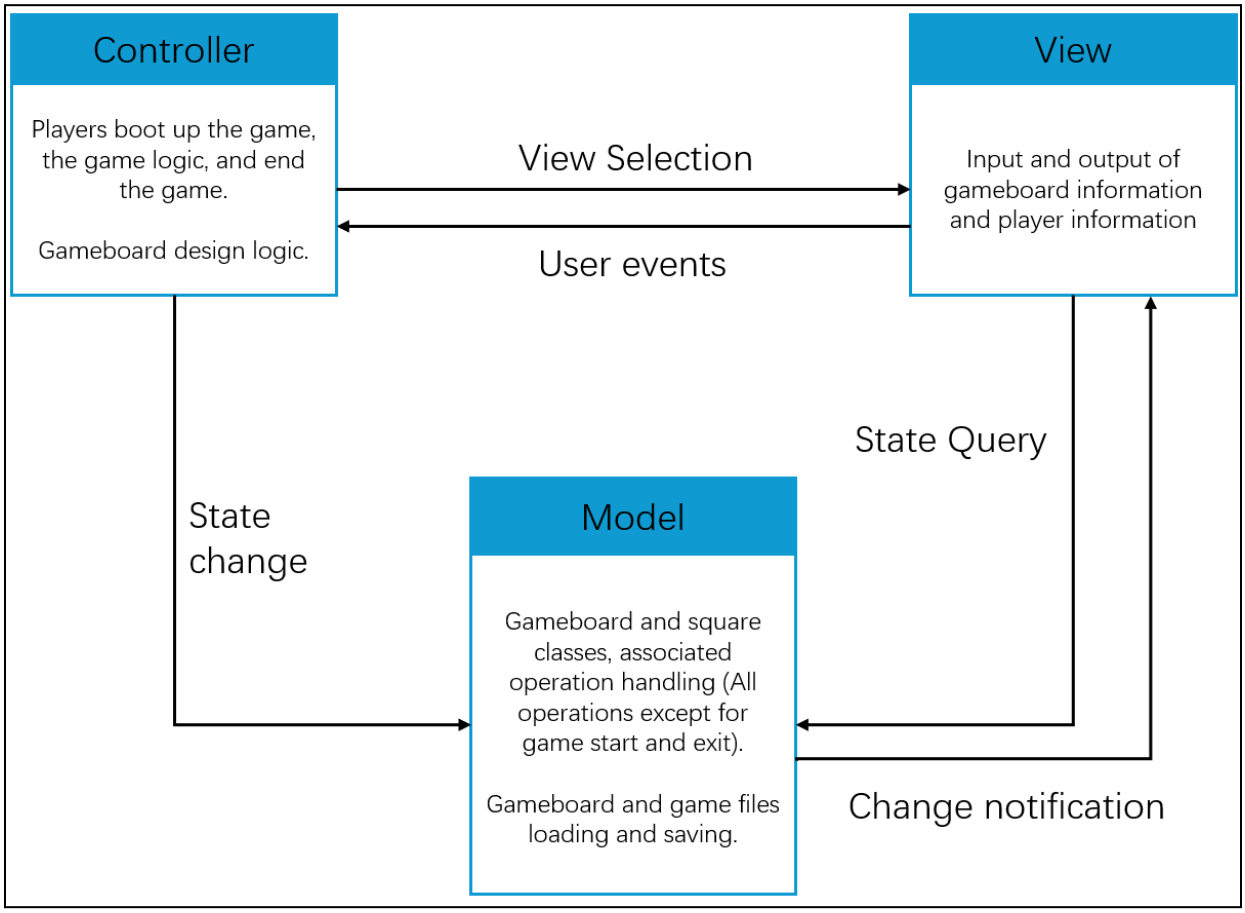
This design document is for demonstrating the main architecture of the project, the structure and relationship between mean codes, and the example of use of the software.

Our project is designed in terms of modularity, efficiency, extendibility, and justifiability.

1. Modularity: Project code is mainly divided into three packages of model, view, and controller emphasizing the modularity of the software. Deepen into the code, for example the load map function and the load game function for implementing load previous game. Load map function is specified for loading the gameboard and load game loads the gameboard along with other game data. This distinction enhances the reusability and flexibility in other parts of the software that may require map loading independent of the full game data.
2. Efficiency: The code has been frequently refactored and improve the efficiency of the software performance. Implementing the agile software development, the group refactored the code continuously as soon as possible code improvements are found, which keeps the code simple and maintainable.
3. Extendibility: The code design formed great extendibility for the software. For example, the group designed the abstract class of square and the abstract function of take effect under square. All other classes for specific squares (Chance, FreeParking, Go, GoJail, IncomeTax, InJailOrJustVisiting, and Property) inherit the function of take effect from abstract class square.
4. Justifiability: The project design follows great justifiability, since the group specified the requirements clearly before the code development, which guided the distribution of functionality and ensured alignment with project goals.

© 2024 MonoPolyU. All rights reserved.

Architecture



Graph 1: Architectural for the Monopoly Game

As shown in Graph 1 of the architecture designed for the project, it has been decided to adopt MVC architecture pattern for the task to allow multiple ways of viewing to view and interact with data. It enables the data to be modified independently of how it is represented, and vice versa. Additionally, it allows the same data to be displayed in various formats, ensuring that any changes made to one representation are reflected across all of them.

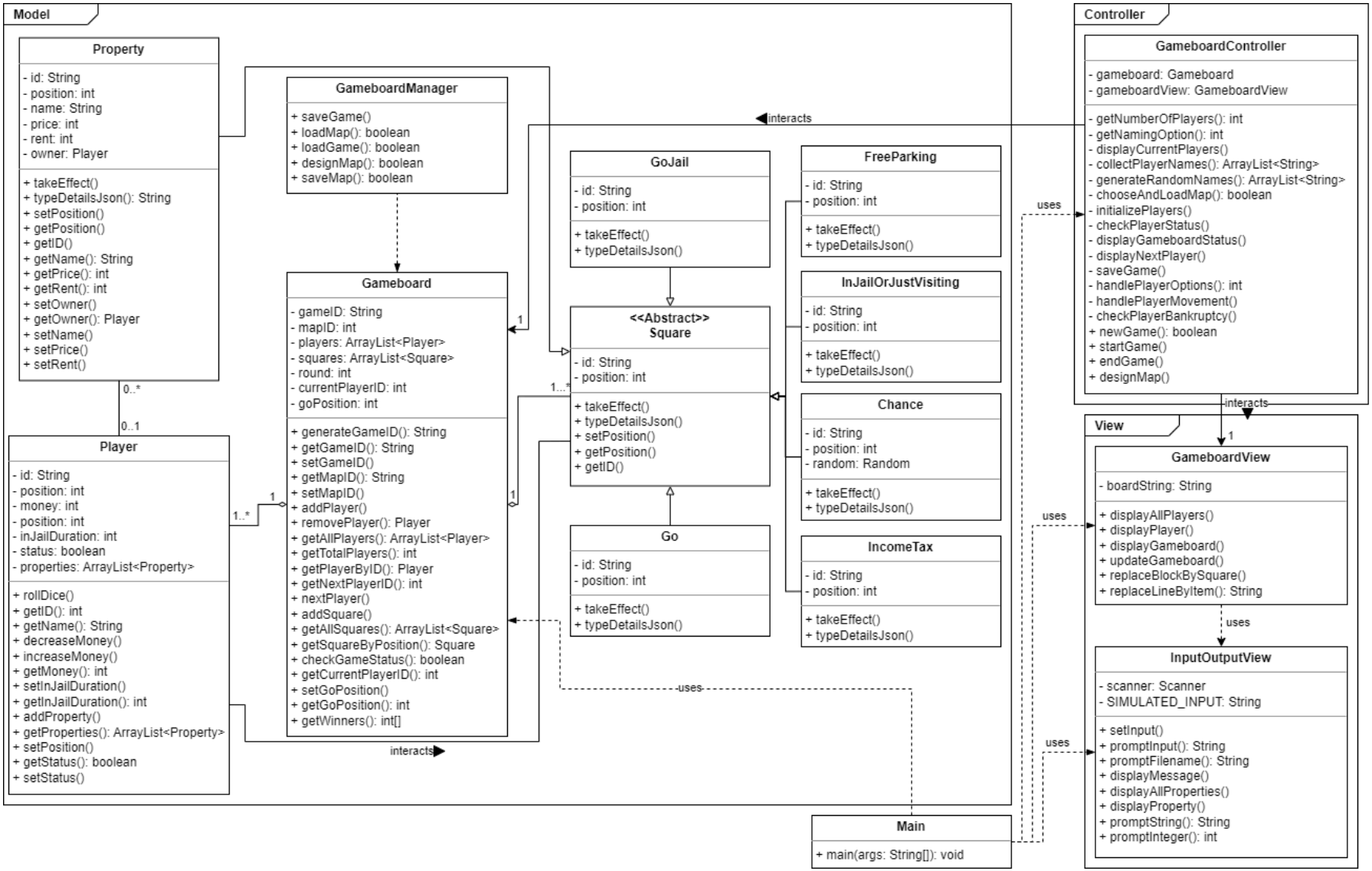
Controller: The interactions between the players, designers and the system are managed by this part. It facilitates the sharing of user data with the View and the Model after booting up the system and processing updates.

© 2024 MonoPolyU. All rights reserved.

View: The player and gameboard data will be presented through this component, either upon request or during critical situations.

Model: The player data and gameboard data will primarily be managed within this component. It will handle the operations requested by users and update the data or files accordingly.

© 2024 MonoPolyU. All rights reserved.



© 2024 MonoPolyU. All rights reserved.

Structure and Relationship of Main Code Components

As shown in graph two of the UML, it can be seen that the whole project followed the MVC structure and the code has been divided into the three packages of model, control, and view. This shows the modularity of our code, and the detailed information is explained below.

Model Package

1. Square (Abstract Class): Serves as the base class for specific types of game squares.

Constructor		
1	Square(int position, String id)	
	Arguments	position (Type: int) id (Type: String)
	Description	Calls the superclass constructor to initialize the position and ID of the FreeParking square.
Abstract Methods		
1	takeEffect(Player player)	
	Return Type	void
	Arguments	player (Type: Player)
	Description	Takes an effect on a Player based on the Square.
2	detailsInJSON()	
	Return Type	String
	Arguments	None
	Description	Returns a JSON representation of the details of the FreeParking square. Currently, it returns an empty details object.
Methods		
1	setPosition(int position)	

© 2024 MonoPolyU. All rights reserved.

	Return Type	void
	Arguments	position (Type: int)
	Description	Set the position of the Square in the gameboard.
2	getPosition()	
	Return Type	int
	Arguments	None
	Description	Get the position of the Square in the gameboard.
3	getId()	
	Return Type	int
	Arguments	None
	Description	Get the id of the Square in the gameboard.

Below are the subclasses:

a. Chance : Represents the square where a player gains random amount of money as awards or lose random amount of money as penalty.
b. FreeParking: Represents the square where no action occurs.
c. Go: Represents the square where players starts from and receives salary every time landing on or passing the square.
d. GoJail: Represents the square sending players landing on itself into jail.
e. IncomeTax: Represents the square where players needs to pay tax when visiting.
f. InJailOrJustVisiting: Represents the square where players sent to jail are kept and doesn't affect those just visiting.
g. Property: Represents squares that players can purchase to occupy and gain rents from other players visiting.

2. Player: Represents game participants with attributes such as name, money, and properties owned.

Constructor	
1	Player(int id, String name, int money, int position)

© 2024 MonoPolyU. All rights reserved.

	Description	Initializes the player’s ID, name, money, and position.
Methods		
1	rollDice()	
	Return Type	void
	Arguments	None
	Description	Roll the dice on the gameboard.
2	getId()	
	Return Type	int
	Arguments	None
	Description	Get the player ID.
3	getName()	
	Return Type	String
	Arguments	None
	Description	Get the player's name.
4	decreaseMoney(int amount)	
	Return Type	void
	Arguments	amount (Type: int)
	Description	Decrease the player's money.
5	increaseMoney(int amount)	
	Return Type	void
	Arguments	amount (Type: int)
	Description	Increase the player's money.
6	getMoney()	
	Return Type	int
	Arguments	None

© 2024 MonoPolyU. All rights reserved.

	Description	Get the player’s money.
7	setInJailDuration(int duration)	
	Return Type	void
	Arguments	duration (Type: int)
	Description	Set the player’s in-jail duration.
8	getInJailDuration()	
	Return Type	int
	Arguments	void
	Description	Get the player’s in-jail duration money.
9	addProperty(Property property)	
	Return Type	void
	Arguments	property (Type: Property)
	Description	Add a property to the player.
10	getProperties()	
	Return Type	ArrayList<Property>
	Arguments	None
	Description	Get all the properties of the player.
11	getPosition()	
	Return Type	int
	Arguments	None
	Description	Get the position of the player.
12	setPosition(int position)	
	Return Type	void
	Arguments	position (Type: int)
	Description	Set the position of the player.

© 2024 MonoPolyU. All rights reserved.

13	getStatus()	
	Return Type	boolean
	Arguments	None
	Description	Get the status of the player.
14	setStatus(boolean status)	
	Return Type	void
	Arguments	status (Type: boolean)
	Description	Set the status of the player.

3. Gameboard: Manages the overall game environment including players, squares, and game state such as rounds and current player.

Constructor		
1	Gameboard()	
	Description	Initializes the players and squares lists, sets the round to 1, and the current player ID to 1.
Methods		
1	generateGameID()	
	Return Type	String
	Arguments	player (Type: Player)
	Description	Generates a unique game ID based on the current date and time.
2	getGameID()	
	Return Type	String
	Arguments	None
	Description	Get the game ID.
3	setGameID(String gameID)	
	Return Type	None

© 2024 MonoPolyU. All rights reserved.

	Arguments	gameID (Type: String)
	Description	Set the game ID.
4	getMapID()	
	Return Type	String
	Arguments	None
	Description	Get the map ID.
5	setMapID(String mapID)	
	Return Type	None
	Arguments	mapID (Type: String)
	Description	Set the map ID.
6	getRoundNumber()	
	Return Type	int
	Arguments	None
	Description	Get the current round number.
7	setRoundNumber(int round)	
	Return Type	None
	Arguments	roundNumber (Type: int)
	Description	Set the current round number.
8	addPlayer(Player player)	
	Return Type	void
	Arguments	player (Type: Player)
	Description	Adds a player to the game.
9	removePlayer(Player player)	
	Return Type	Player
	Arguments	player (Type: Player)

© 2024 MonoPolyU. All rights reserved.

	Description	Removes a player from the game by setting their status to false.
10	getAllPlayers()	
	Return Type	ArrayList<Player>
	Arguments	None
	Description	Returns a list of all players.
11	getPlayerByID(int playerId)	
	Return Type	Player
	Arguments	playerId (Type: int)
	Description	Retrieves a player by their ID.
12	getNextPlayer()	
	Return Type	Player
	Arguments	None
	Description	Determines the next player with a positive balance.
13	nextPlayer()	
	Return Type	void
	Arguments	None
	Description	Advances to the next player and updates the round if needed.
14	addSquare(Square square)	
	Return Type	void
	Arguments	square (Type: Square)
	Description	Adds a square to the gameboard.
15	getAllSquares()	
	Return Type	ArrayList<Square>
	Arguments	None

© 2024 MonoPolyU. All rights reserved.

	Description	Returns a list of all squares.
16	getSquareByPosition(int position)	
	Return Type	Square
	Arguments	position (Type: int)
	Description	Retrieves a square by its position
17	checkGameStatus()	
	Return Type	boolean
	Arguments	None
	Description	Checks if the game can continue based on the number of active players and rounds
18	getCurrentPlayerID()	
	Return Type	int
	Arguments	None
	Description	Returns the ID of the current player.
19	setGoPosition(int position)	
	Return Type	None
	Arguments	position (Type: int)
	Description	Set the position of the "Go" square.
20	getGoPosition()	
	Return Type	int
	Arguments	None
	Description	Get the position of the "Go" square.
21	getWinners()	
	Return Type	int[]
	Arguments	None

© 2024 MonoPolyU. All rights reserved.

	Description	Determines the winner(s) based on the highest money among active players.

4. GameboardManager: Handles saving and loading game states, managing game data persistence through file operations.

Methods		
1	saveGame(Gameboard gameboard, String filepath)	
	Return Type	void
	Arguments	gameboard (Type: Gameboard) filepath (Type: String)
	Description	Saves the current state of the gameboard to a JSON file at the specified filepath.
2	loadMap(String filepath, Gameboard gameboard)	
	Return Type	void
	Arguments	filepath (Type: String) gameboard (Type: Gameboard)
	Description	Loads a map from a JSON file and populates the gameboard with squares.
3	loadGame(Gameboard gameboard)	
	Return Type	void
	Arguments	gameboard (Type: Gameboard)
	Description	Loads a complete game state from a JSON file into the gameboard.
4	saveMap(ArrayList<Square> squares, String mapid, String filepath)	
	Return Type	void
	Arguments	squares (Type: ArrayList<Square>) mapid (Type: String) filepath (Type: String)
	Description	Saves the map's squares to a JSON file.

© 2024 MonoPolyU. All rights reserved.

View Package

1. InputOutputView: Manages user interactions, handling inputs and outputs across the game system.

Static Methods		
1	setInput(String input)	
	Return Type	void
	Arguments	input (Type: String)
	Description	Sets a simulated input string for testing purposes.
2	promptInput(String prompt, String[] options)	
	Return Type	String
	Arguments	prompt (Type: String) options (Type: String[])
	Description	Prompts the user for input with specific options. Continues until a valid option is entered.
3	promptFilename(String prompt)	
	Return Type	String
	Arguments	prompt (Type: String)
	Description	Prompts the user for a filename, rejecting input with slashes.
4	displayMessage(String message)	
	Return Type	void
	Arguments	message (Type: String)
	Description	Displays a message to the console.
5	displayAllProperties(ArrayList<Property> properties)	
	Return Type	void
	Arguments	properties (Type: ArrayList<Property>)
	Description	Displays information for each property in the list.

© 2024 MonoPolyU. All rights reserved.

6	displayProperty(Property property)	
	Return Type	void
	Arguments	property (Type: Property)
	Description	Displays details of a single property.
7	promptString(String prompt)	
	Return Type	String
	Arguments	prompt (Type: String)
	Description	Prompts the user for a valid alphanumeric string input.
8	promptInteger(String prompt)	
	Return Type	int
	Arguments	prompt (Type: String)
	Description	Prompts the user for a positive integer. Handles invalid input with error messages.

2. GameboardView: Handles the visual representation of the game board, updating and displaying the board based on game state changes.

Methods		
1	displayAllPlayers(ArrayList<Player> players)	
	Return Type	void
	Arguments	players (Type: ArrayList<Player>)
	Description	Displays information for all players. If no players exist, it shows a message indicating so.
2	displayPlayer(Player player)	
	Return Type	void
	Arguments	player (Type: Player)
	Description	Displays detailed information for a single player, including their ID, money, position, and properties. If the player is null, it displays a message indicating no player to display.

© 2024 MonoPolyU. All rights reserved.

3	displayGameboard()	
	Return Type	void
	Arguments	None
	Description	Displays the current state of the gameboard
4	updateGameboard(Gameboard gameboard)	
	Return Type	void
	Arguments	gameboard (Type: Gameboard)
	Description	Updates the gameboard string with the current positions of all players.
5	replaceBlockBySquare(Square square)	
	Return Type	void
	Arguments	square (Type: Square)
	Description	Updates the gameboard string with the details of a specific square, depending on its type (Property, Go, Chance, etc.).
6	replaceLineByItem(String item)	
	Return Type	String
	Arguments	item (Type: String)
	Description	Centers a given string within a predefined space and returns the formatted line.

Controller Package

1. GameboardController: Coordinates interactions between the model (Gameboard) and the view components, managing game logic and user interactions.

Constructor		
1	GameboardController(Gameboard gameboard, GameboardView gameboardView)	
	Return Type	N/A (Constructor)

© 2024 MonoPolyU. All rights reserved.

	Arguments	gameboard (Type: Gameboard) gameboardView (Type: GameboardView)
	Description	Initializes the controller with a Gameboard and a GameboardView.
Private Methods		
1	getNumberOfPlayers(int min, int max)	
	Return Type	int
	Arguments	min (Type: int) max (Type: int)
	Description	Prompts the user to enter the number of players, ensuring it is between min and max.
2	getNameingOption()	
	Return Type	int
	Arguments	None
	Description	Prompts the user to choose a player naming option (manual or random).
3	displayCurrentPlayers(ArrayList<String> names)	
	Return Type	void
	Arguments	names (Type: ArrayList<String>)
	Description	Displays the list of current player names.
4	collectPlayerNames(int numberOfPlayers)	
	Return Type	ArrayList<String>
	Arguments	numberOfPlayers (Type: int)
	Description	Collects names for the specified number of players, ensuring uniqueness and length constraints.
5	generateRandomNames(int numberOfPlayers)	
	Return Type	ArrayList<String>
	Arguments	numberOfPlayers (Type: int)

© 2024 MonoPolyU. All rights reserved.

	Description	Generates random names for the specified number of players.
6	chooseAndLoadMap()	
	Return Type	boolean
	Arguments	None
	Description	Allows the user to choose and load a map. Returns true if a map is successfully loaded.
7	initializePlayers(ArrayList<String> names, int startingMoney)	
	Return Type	void
	Arguments	names (Type: ArrayList<String>) startingMoney (Type: int)
	Description	Initializes players with the given names and starting money.
7	checkPlayerStatus()	
	Return Type	void
	Arguments	None
	Description	Prompts the user to check player statuses and displays information accordingly.
8	displayGameboardStatus()	
	Return Type	void
	Arguments	None
	Description	Updates and displays the current gameboard status.
9	displayNextPlayer()	
	Return Type	void
	Arguments	None
	Description	Displays the next player's information.
10	saveGame()	
	Return Type	void

© 2024 MonoPolyU. All rights reserved.

	Arguments	None
	Description	Saves the current game state to a file.
11	handlePlayerOptions(Player currentPlayer)	
	Return Type	int
	Arguments	currentPlayer (Type: Player)
	Description	Handles player options during their turn. Returns an integer representing the action taken.
12	handlePlayerMovement(Player currentPlayer, int initialPosition, int currentPosition)	
	Return Type	void
	Arguments	currentPlayer (Type: Player) initialPosition (Type: int) currentPosition (Type: int)
	Description	Manages player movement and applies effects based on their position.
13	checkPlayerBankruptcy(Player currentPlayer)	
	Return Type	void
	Arguments	currentPlayer (Type: Player)
	Description	Checks if the player is bankrupt and removes them from the game if necessary.
Public Methods		
1	newGame()	
	Return Type	void
	Arguments	None
	Description	Starts a new game by setting up players and loading a map.
2	startGame()	
	Return Type	void
	Arguments	None

© 2024 MonoPolyU. All rights reserved.

	Description	Begins the game loop, handling player turns and game progression.
3	endGame()	
	Return Type	void
	Arguments	None
	Description	Ends the game and announces the winner(s).
4	designMap()	
	Return Type	void
	Arguments	None
	Description	Allows designing a new map by modifying properties and saving it as a new file.
	Exceptions	Handles IOException during file reading.

Main Class

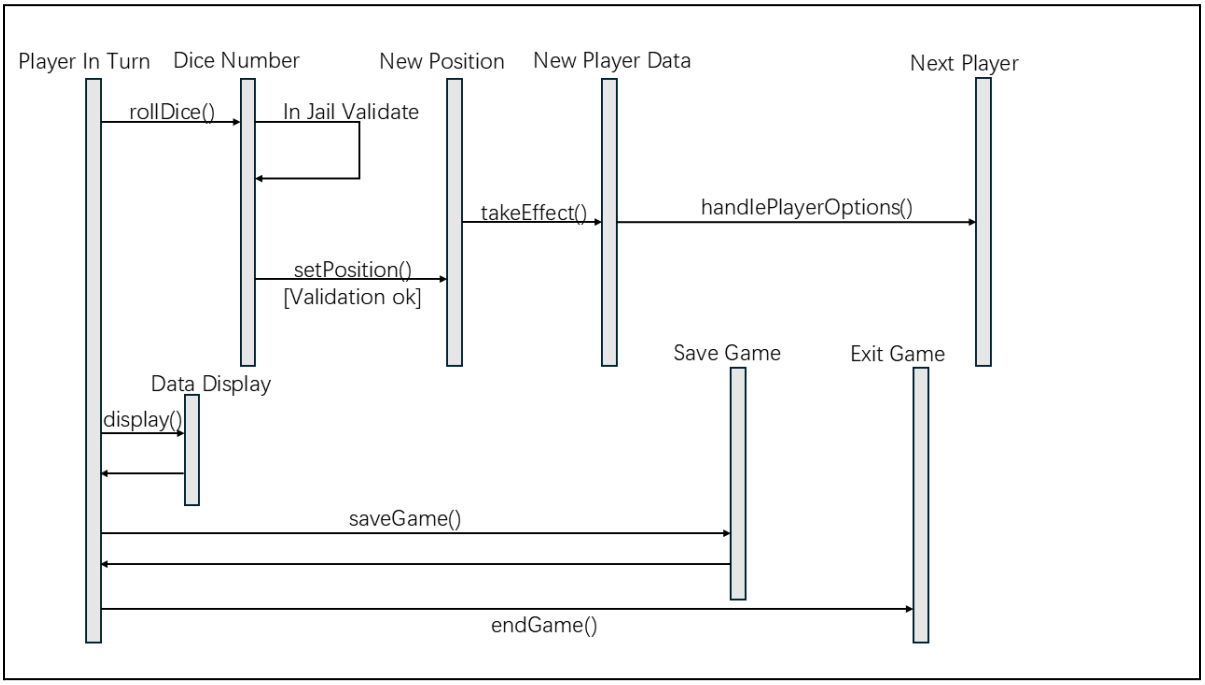
1. Main: Acts as the entry point of the application, initializing the game and managing initial user choices such as starting or loading a game.

Relationships

1. Subclasses of Square inherit common functionality from the abstract class.
2. Gameboard aggregates Player and Square, indicating that it manages multiple instances of these objects.
3. GameboardController interacts directly with Gameboard and GameboardView to synchronize game state with its presentation.
4. Dependency relationships show usage of InputView for handling user inputs and displaying information, essential for interactive components like GameboardView and GameboardManager.

© 2024 MonoPolyU. All rights reserved.

Example Use



Graph 3: sequence diagram of the Monopoly Game

As shown in Graph 3, it outlines the steps involved in a player's turn in the Monopoly Game:

1. Player In Turn initiates the turn by rolling the dice.
2. New Position processes the dice result:
    a. Validates if the player is in jail.
    b. Updates the player's position if validation passes.
    c. Applies the effects of the landed square (like buying property or paying rent).
3. New Player Data updates player information after landed square taken effect.
4. Save Game offers an option to save the current game state.
5. Exit Game includes an option to leave the game.
6. Next Player transitions control, handling options for the next player's turn.

© 2024 MonoPolyU. All rights reserved.