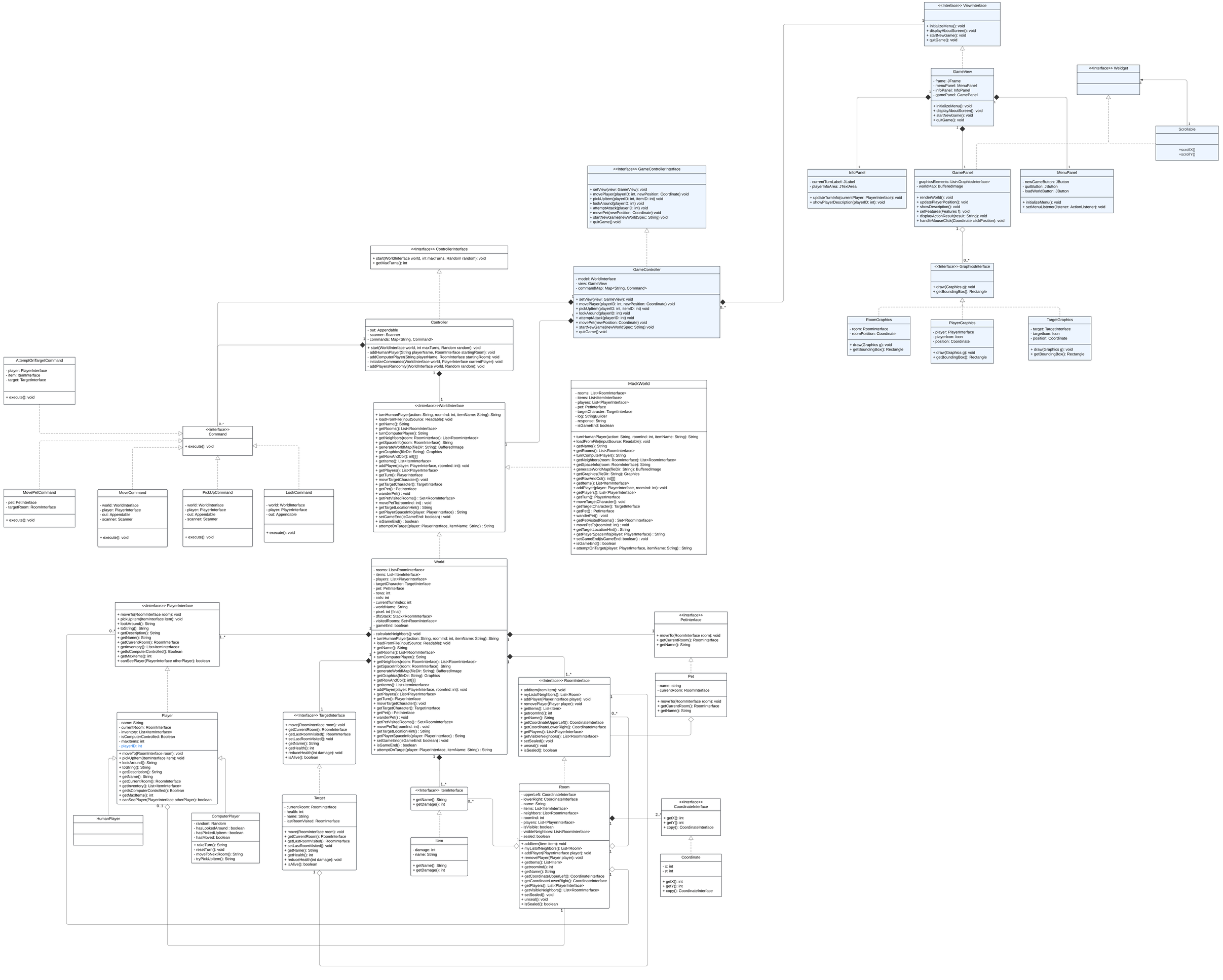

<<Interface>> ViewInterface
+ initializeMenu(): void
+ displayAboutScreen(): void
+ startNewGame(): void
+ quitGame(): void
GameView
- frame: JFrame
- menuPanel: MenuPanel
- infoPanel: InfoPanel
- gamePanel: GamePanel
+ initializeMenu(): void
+ displayAboutScreen(): void
+ startNewGame(): void
+ quitGame(): void
<<Interface>> Weidget
Scrollable
+scrollX()
+scrollY()
InfoPanel
- currentTurnLabel: JLabel
- playerInfoArea: JTextArea
+ updateTurnInfo(currentPlayer: PlayerInterface): void
+ showPlayerDescription(playerID: int): void
GamePanel
- graphicsElements: List<GraphicsInterface>
- worldMap: BufferedImage
+ renderWorld(): void
+ updatePlayerPosition(): void
+ showDescription(): void
+ setFeatures(Features f): void
+ displayActionResult(result: String): void
+ handleMouseClick(Coordinate clickPosition): void
MenuPanel
- newGameButton: JButton
- quitButton: JButton
- loadWorldButton: JButton
+ initializeMenu(): void
+ setMenuListener(listener: ActionListener): void
<<Interface>> GameControllerInterface
+ setView(view: GameView): void
+ movePlayer(playerID: int, newPosition: Coordinate) void
+ pickUpItem(playerID: int, itemID: int) void
+ lookAround(playerID: int) void
+ attemptAttack(playerID: int) void
+ movePet(newPosition: Coordinate) void
+ startNewGame(newWorldSpec: String) void
+ quitGame() void
<<Interface>> ControllerInterface
+ start(WorldInterface world, int maxTurns, Random random): void
+ getMaxTurns(): int
GameController
- model: WorldInterface
- view: GameView
- commandMap: Map<String, Command>
+ setView(view: GameView): void
+ movePlayer(playerID: int, newPosition: Coordinate) void
+ pickUpItem(playerID: int, itemID: int) void
+ lookAround(playerID: int) void
+ attemptAttack(playerID: int) void
+ movePet(newPosition: Coordinate) void
+ startNewGame(newWorldSpec: String) void
+ quitGame() void
<<Interface>> GraphicsInterface
+ draw(Graphics g): void
+ getBoundingBox(): Rectangle
RoomGraphics
- room: RoomInterface
- roomPosition: Coordinate
+ draw(Graphics g): void
+ getBoundingBox(): Rectangle
PlayerGraphics
- player: PlayerInterface
- playerIcon: Icon
- position: Coordinate
+ draw(Graphics g): void
+ getBoundingBox(): Rectangle
TargetGraphics
- target: TargetInterface
- targetIcon: Icon
- position: Coordinate
+ draw(Graphics g): void
+ getBoundingBox(): Rectangle
AttemptOnTargetCommand
- player: PlayerInterface
- item: ItemInterface
- target: TargetInterface
+ execute(): void
Controller
- out: Appendable
- scanner: Scanner
- commands: Map<String, Command>
+ start(WorldInterface world, int maxTurns, Random random): void
- addHumanPlayer(String playerName, RoomInterface startingRoom): void
- addComputerPlayer(String playerName, RoomInterface startingRoom): void
- initializeCommands(WorldInterface world, PlayerInterface currentPlayer): void
- addPlayersRandomly(WorldInterface world, Random random): void
<<Interface>>WorldInterface
+ turnHumanPlayer(action: String, roomInd: int, itemName: String): String
+ loadFromFile(inputSource: Readable): void
+ getName(): String
+ getRooms(): List<RoomInterface>
+ turnComputerPlayer(): String
+ getNeighbors(room: RoomInterface): List<RoomInterface>
+ getSpaceInfo(room: RoomInterface): String
+ generateWorldMap(fileDir: String): BufferedImage
+ getGraphics(fileDir: String): Graphics
+ getRowAndCol(): int[][]
+ getItems(): List<ItemInterface>
+ addPlayer(player: PlayerInterface, roomInd: int): void
+ getPlayers(): List<PlayerInterface>
+ getTurn(): PlayerInterface
+ moveTargetCharacter(): void
+ getTargetCharacter(): TargetInterface
+ getPet() : PetInterface
+ wanderPet() : void
+ getPetVisitedRooms() : Set<RoomInterface>
+ movePetTo(roomInd: int) : void
+ getTargetLocationHint() : String
+ getPlayerSpaceInfo(player: PlayerInterface) : String
+ setGameEnd(isGameEnd: boolean) : void
+ isGameEnd() : boolean
+ attemptOnTarget(player: PlayerInterface, itemName: String) : String
MockWorld
- rooms: List<RoomInterface>
- items: List<ItemInterface>
- players: List<PlayerInterface>
- pet: PetInterface
- targetCharacter: TargetInterface
- log: StringBuilder
- response: String
- isGameEnd: boolean
+ turnHumanPlayer(action: String, roomInd: int, itemName: String): String
+ loadFromFile(inputSource: Readable): void
+ getName(): String
+ getRooms(): List<RoomInterface>
+ turnComputerPlayer(): String
+ getNeighbors(room: RoomInterface): List<RoomInterface>
+ getSpaceInfo(room: RoomInterface): String
+ generateWorldMap(fileDir: String): BufferedImage
+ getGraphics(fileDir: String): Graphics
+ getRowAndCol(): int[][]
+ getItems(): List<ItemInterface>
+ addPlayer(player: PlayerInterface, roomInd: int): void
+ getPlayers(): List<PlayerInterface>
+ getTurn(): PlayerInterface
+ moveTargetCharacter(): void
+ getTargetCharacter(): TargetInterface
+ getPet() : PetInterface
+ wanderPet() : void
+ getPetVisitedRooms() : Set<RoomInterface>
+ movePetTo(roomInd: int) : void
+ getTargetLocationHint() : String
+ getPlayerSpaceInfo(player: PlayerInterface) : String
+ setGameEnd(isGameEnd: boolean) : void
+ isGameEnd() : boolean
+ attemptOnTarget(player: PlayerInterface, itemName: String) : String
<<interface>>
Command
+ execute(): void
MovePetCommand
- pet: PetInterface
- targetRoom: RoomInterface
+ execute(): void
MoveCommand
- world: WorldInterface
- player: PlayerInterface
- out: Appendable
- scanner: Scanner
+ execute(): void
PickUpCommand
- world: WorldInterface
- player: PlayerInterface
- out: Appendable
- scanner: Scanner
+ execute(): void
LookCommand
- world: WorldInterface
- player: PlayerInterface
- out: Appendable
+ execute(): void
World
- rooms: List<RoomInterface>
- items: List<ItemInterface>
- players: List<PlayerInterface>
- targetCharacter: TargetInterface
- pet: PetInterface
- rows: int
- cols: int
- currentTurnIndex: int
- worldName: String
- pixel: int (final)
- dfsStack: Stack<RoomInterface>
- visitedRooms: Set<RoomInterface>
- gameEnd: boolean
- calculateNeighbors(): void
+ turnHumanPlayer(action: String, roomInd: int, itemName: String): String
+ loadFromFile(inputSource: Readable): void
+ getName(): String
+ getRooms(): List<RoomInterface>
+ turnComputerPlayer(): String
+ getNeighbors(room: RoomInterface): List<RoomInterface>
+ getSpaceInfo(room: RoomInterface): String
+ generateWorldMap(fileDir: String): BufferedImage
+ getGraphics(fileDir: String): Graphics
+ getRowAndCol(): int[][]
+ getItems(): List<ItemInterface>
+ addPlayer(player: PlayerInterface, roomInd: int): void
+ getPlayers(): List<PlayerInterface>
+ getTurn(): PlayerInterface
+ moveTargetCharacter(): void
+ getTargetCharacter(): TargetInterface
+ getPet() : PetInterface
+ wanderPet() : void
+ getPetVisitedRooms() : Set<RoomInterface>
+ movePetTo(roomInd: int) : void
+ getTargetLocationHint() : String
+ getPlayerSpaceInfo(player: PlayerInterface) : String
+ setGameEnd(isGameEnd: boolean) : void
+ isGameEnd() : boolean
+ attemptOnTarget(player: PlayerInterface, itemName: String) : String
<<Interface>> PlayerInterface
+ moveTo(RoomInterface room): void
+ pickUpItem(ItemInterface item): void
+ lookAround(): String
+ toString(): String
+ getDescription(): String
+ getName(): String
+ getCurrentRoom(): RoomInterface
+ getInventory(): List<ItemInterface>
+ getIsComputerControlled(): Boolean
+ getMaxItems(): int
+ canSeePlayer(PlayerInterface otherPlayer): boolean
Player
- name: String
- currentRoom: RoomInterface
- inventory: List<ItemInterface>
- isComputerControlled: Boolean
- maxItems: int
- playerID: int
+ moveTo(RoomInterface room): void
+ pickUpItem(ItemInterface item): void
+ lookAround(): String
+ toString(): String
+ getDescription(): String
+ getName(): String
+ getCurrentRoom(): RoomInterface
+ getInventory(): List<ItemInterface>
+ getIsComputerControlled(): Boolean
+ getMaxItems(): int
+ canSeePlayer(PlayerInterface otherPlayer): boolean
HumanPlayer
ComputerPlayer
- random: Random
- hasLookedAround : boolean
- hasPickedUpItem : boolean
- hasMoved: boolean
+ takeTurn(): String
+ resetTurn(): void
- moveToNextRoom(): String
- tryPickUpItem(): String
<<Interface>> TargetInterface
+ move(RoomInterface room): void
+ getCurrentRoom(): RoomInterface
+ getLastRoomVisited(): RoomInterface
+ setLastRoomVisited(): void
+ getName(): String
+ getHealth(): int
+ reduceHealth(int damage): void
+ isAlive(): boolean
Target
- currentRoom: RoomInterface
- health: int
- name: String
- lastRoomVisited: RoomInterface
+ move(RoomInterface room): void
+ getCurrentRoom(): RoomInterface
+ getLastRoomVisited(): RoomInterface
+ setLastRoomVisited(): void
+ getName(): String
+ getHealth(): int
+ reduceHealth(int damage): void
+ isAlive(): boolean
<<Interface>> ItemInterface
+ getName(): String
+ getDamage(): int
Item
- damage: int
- name: String
+ getName(): String
+ getDamage(): int
<<Interface>> RoomInterface
+ addItem(Item item): void
+ myListofNeighbors(): List<Room>
+ addPlayer(PlayerInterface player): void
+ removePlayer(Player player): void
+ getItems(): List<Item>
+ getroomInd(): int
+ getName(): String
+ getCoordinateUpperLeft(): CoordinateInterface
+ getCoordinateLowerRight(): CoordinateInterface
+ getPlayers(): List<PlayerInterface>
+ getVisibleNeighbors(): List<RoomInterface>
+ setSealed(): void
+ unseal(): void
+ isSealed(): boolean
Room
- upperLeft: CoordinateInterface
- lowerRight: CoordinateInterface
- name: String
- items: List<ItemInterface>
- neighbors: List<RoomInterface>
- roominId: int
- players: List<PlayerInterface>
- isVisible: boolean
- visibleNeighbors: List<RoomInterface>
- sealed: boolean
+ addItem(Item item): void
+ myListofNeighbors(): List<Room>
+ addPlayer(PlayerInterface player): void
+ removePlayer(Player player): void
+ getItems(): List<Item>
+ getroomInd(): int
+ getName(): String
+ getCoordinateUpperLeft(): CoordinateInterface
+ getCoordinateLowerRight(): CoordinateInterface
+ getPlayers(): List<PlayerInterface>
+ getVisibleNeighbors(): List<RoomInterface>
+ setSealed(): void
+ unseal(): void
+ isSealed(): boolean
<<interface>>
PetInterface
+ moveTo(RoomInterface room): void
+ getCurrentRoom(): RoomInterface
+ getName(): String
Pet
- name: string
- currentRoom: RoomInterface
+ moveTo(RoomInterface room): void
+ getCurrentRoom(): RoomInterface
+ getName(): String
<<interface>>
CoordinateInterface
+ getX(): int
+ getY(): int
+ copy(): CoordinateInterface
Coordinate
- x: int
- y: int
+ getX(): int
+ getY(): int
+ copy(): CoordinateInterface

Welcome to Doctor Lucky!

In Doctor Lucky, players attempt to track and kill Doctor Lucky while outsmarting their opponents. Can you succeed before anyone else?

Created by Xinlai Chen

Start Game

Menu

New Game with New World Specification

New Game with Current World Specification

Quit Game

Menu

- It's xxx's turn
- You position:
- # of Nrighboring Rooms:
- Inventory:

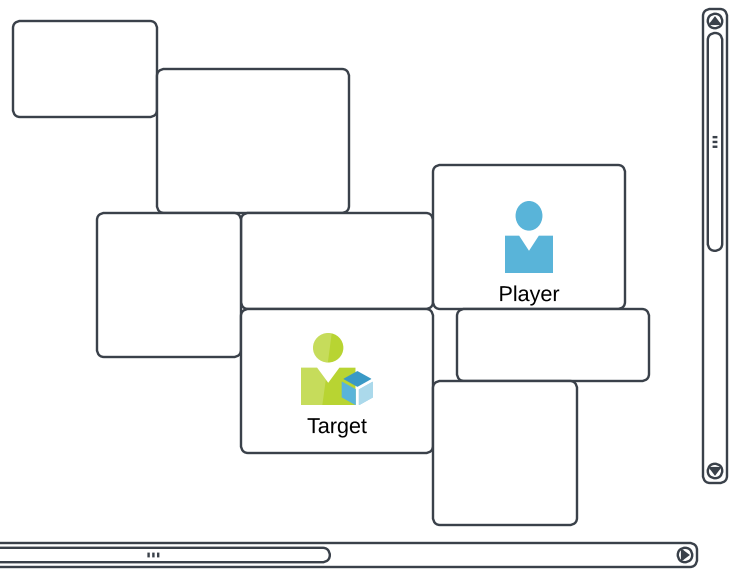
Clicking on adjacent rooms to move a player

P key: Pick up an item in the current room.

L key: Look around the current room.

A key: Attempt to attack Doctor Lucky.

Result: You picked up a revolver



Milestone 4 - Preliminary Design

Design Changes

For Milestone 4, to separate the model from the text-based controller and adapt it to the graphical view, I made several modifications to the model design. I moved the player movement logic, target character updates, and pet behavior into dedicated commands that are now implemented through interfaces. This allowed the model to interact seamlessly with either a graphical or text-based controller. Additionally, I enhanced the data structures to maintain graphical information, ensuring that the state of each element—such as players, rooms, and items—could be accessed without relying on console-based descriptions. The introduction of `GraphicsInterface` and distinct graphics classes like `RoomGraphics` and `PlayerGraphics` decouples visualization logic, allowing the game model to remain isolated from view-related responsibilities. I also designed two controllers that can be selected by the driver, allowing it to either run a text-based controller or a graphical-based one, ensuring better separation of concerns. These changes ensure that the model is independent of specific user interfaces, making it adaptable and testable in different contexts.

Testing Plan

1. GameView Class

Methods to Test

- **`initializeMenu()`**: Verify that the menu is correctly initialized, including all buttons and menu items.
- **`displayAboutScreen()`**: Ensure that the about screen is displayed properly with the correct text, including the creator's name and any credits.
- **`startNewGame()`**: Test that a new game can be successfully started, and all related UI components are reset.
- **`quitGame()`**: Validate that the game quits without errors.

Test Scenarios

- The menu should contain options for starting a new game, loading a world, and quitting.
- When the user selects "About," the about screen should display the correct message.
- Clicking "Quit" should close the game and free all resources.

2. GameController Class

Methods to Test

- `movePlayer(int playerID, Coordinate newPosition)`: Test that a player can be moved to a valid location and verify that invalid moves are prevented.
- `pickUpItem(int playerID, int itemID)`: Check that players can pick up items when available and fail gracefully if there are no items to pick up.
- `lookAround(int playerID)`: Verify that looking around shows appropriate information based on the player's current position and adjacent rooms.
- `attemptAttack(int playerID)`: Test attack attempts on the target character, ensuring attacks follow game rules (e.g., being stopped if seen by others).
- `movePet(Coordinate newPosition)`: Validate that the pet moves correctly following commands and continues its wandering behavior.
- `startNewGame(String newWorldSpec)`: Ensure that starting a new game resets the game state and loads the specified world correctly.
- `quitGame()`: Test that the quit action successfully terminates the game.

Test Scenarios

- Moving a player to adjacent rooms and verifying their updated position.
- Trying to pick up an item when no item is present.
- Attacking the target when the player is in a valid position and ensuring unsuccessful attacks when others are present.
- Moving the pet across rooms and checking if its movement is recorded correctly.

3. GamePanel Class

Methods to Test

- `renderWorld()`: Ensure that the world map is rendered correctly, with all components (players, target) visible.
- `updatePlayerPosition()`: Test that a player's icon is correctly repositioned when the player moves.
- `showDescription()`: Verify that clicking on a player displays the correct description in the designated area.
- `handleMouseClick(Coordinate clickPosition)`: Ensure that mouse clicks result in the correct action, such as selecting a player or attempting to move to a new room.

Test Scenarios

- Verifying that clicking adjacent rooms results in the correct player movement.
- Ensuring that graphical representations of players and targets are updated as expected during gameplay.

4. MenuPanel Class

Methods to Test

- `initializeMenu()`: Validate that the menu is initialized correctly, containing buttons for new game, load game, and quit.
- `setMenuListener(ActionListener listener)`: Ensure that the menu reacts appropriately when buttons are clicked.

Test Scenarios

- Testing that clicking "New Game" starts a new game and resets game state.
- Ensuring that the "Quit" button correctly closes the application.

5. InfoPanel Class

Methods to Test

- `updateTurnInfo(PlayerInterface currentPlayer)`: Ensure that turn information is correctly displayed, including which player's turn it is and their current position.
- `showPlayerDescription(int playerID)`: Validate that clicking on a player displays the correct player description.

Test Scenarios

- Clicking on a player's icon and verifying that the player's information is displayed in the information area.
- Ensuring that the turn information updates appropriately when switching players.

6. Scrollable Interface

Methods to Test

- `scrollX()`, `scrollY()`: Ensure that the view scrolls correctly when the world map is larger than the available display area.

Test Scenarios

- Test scrolling functionality by resizing the window and verifying that the user can still view all parts of the map.

7. Graphics Components Testing

RoomGraphics, PlayerGraphics, TargetGraphics

Render Accuracy

- Confirm that rooms, players, and target character are visually rendered correctly and updated in real time.

Bounding Box

- Test that each graphical element's bounding box is correctly calculated to enable accurate selection by the user.

Testing Considerations for Controller in Isolation

Mock Model Strategy

- Use a Mock Model to simulate the behavior of WorldInterface.
- Verify that commands interact correctly with the mock model, focusing on:
    - Movement
    - Picking up items
    - Looking around
    - Attacking
- Ensure the controller correctly updates the mock model without direct dependencies on the actual model.

Mock View Strategy

- Use a Mock View for isolation testing of GameController.
- Ensure commands issued by the controller correctly trigger expected view updates.
- Test the interactions of user inputs through the mock view to verify controller response.