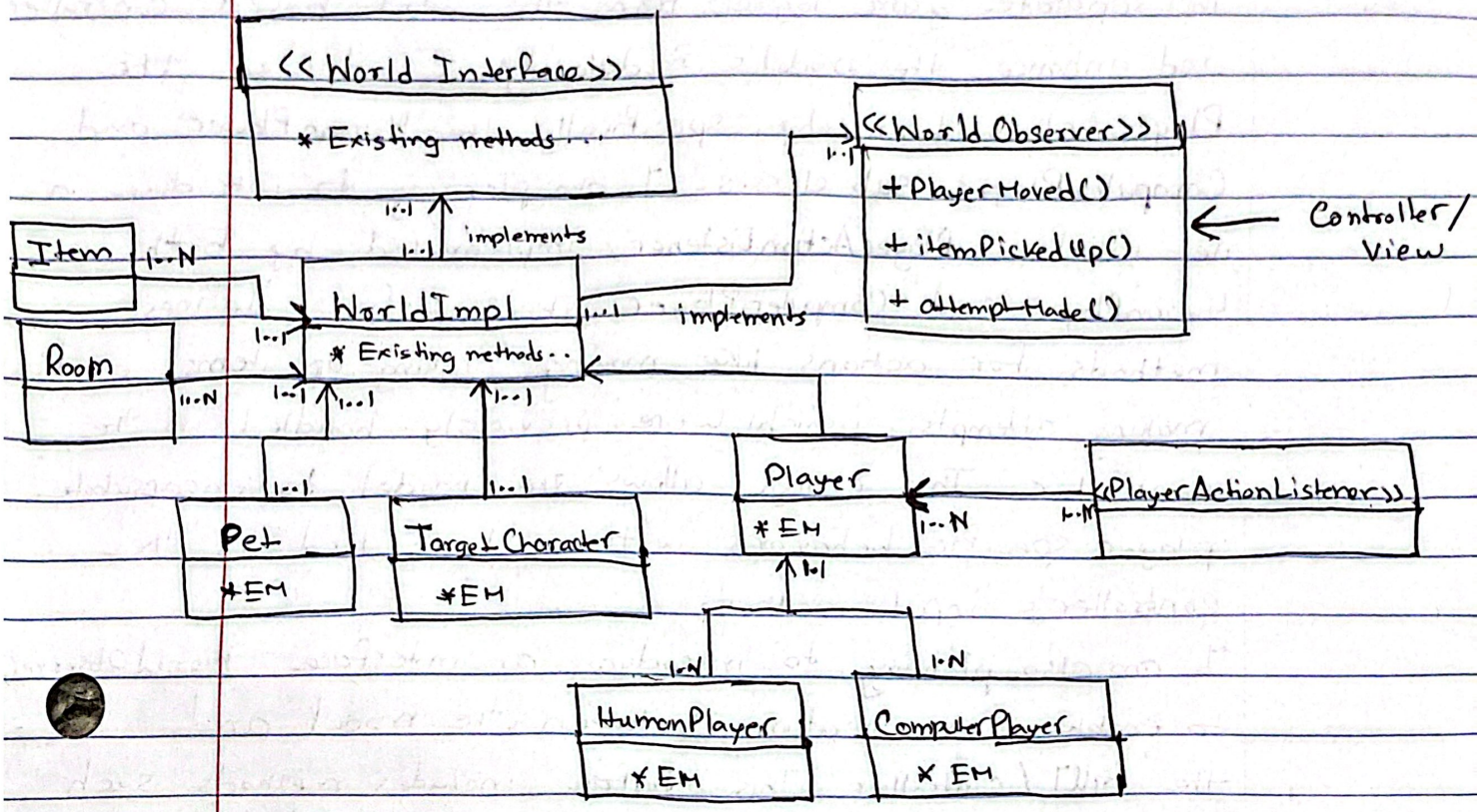
Model

To separate game logic from the text based controller and enhance the model's modularity, I adjusted the Player class hierarchy, specifically the HumanPlayer and ComputerPlayer sub classes. I am planning to introduce a new interface, PlayerActionListener, implemented by both HumanPlayer and ComputerPlayer. This interface defines methods for actions like moving, picking up items, and making attempts, which were previously handled in the controller. This change allows the model to encapsulate player specific behaviors without being tied to the controller's input methods.

I am also planning to introduce an interface, WorldObserver, to enable communication between the model and the GUI / controller. This interface includes methods such as playerMoved, itemPickedUp, and attemptMade, providing a mechanism for the model to notify observers about significant events in the game play.

*EM = Existing Methods.

Model UML



Model Test

	Test Type	Expected Output
1)	World Creation	File read successfully
2)	Rooms Creation	viewAllRooms() gives all the rooms as output.
3)	Items Creation	viewAllItems() gives all the items as output.
4)	Pet Creation	getPet() gives the valid reference to Pet Object
5)	TargetCharacter Creation	getTargetCharacter() gives the valid reference to the TargetCharacter Object. [Dr Lucky]
6)	ComputerPlayer Creation	getPlayerInfor() outputs the correct set of information for the computer player(s)
7)	HumanPlayer Creation	getPlayerInfo() outputs the information correctly as provided by the user.

8) moveTargetCharacter() — moves the target character by 1 room and when the last room is reached, dr. lucky moves to the first room.

9) movePet() — moves the pet using DFS algorithm.

10) movePlayer() — moves the player as per the user input.

11) lookAround() — shows the list of all the neighboring rooms.

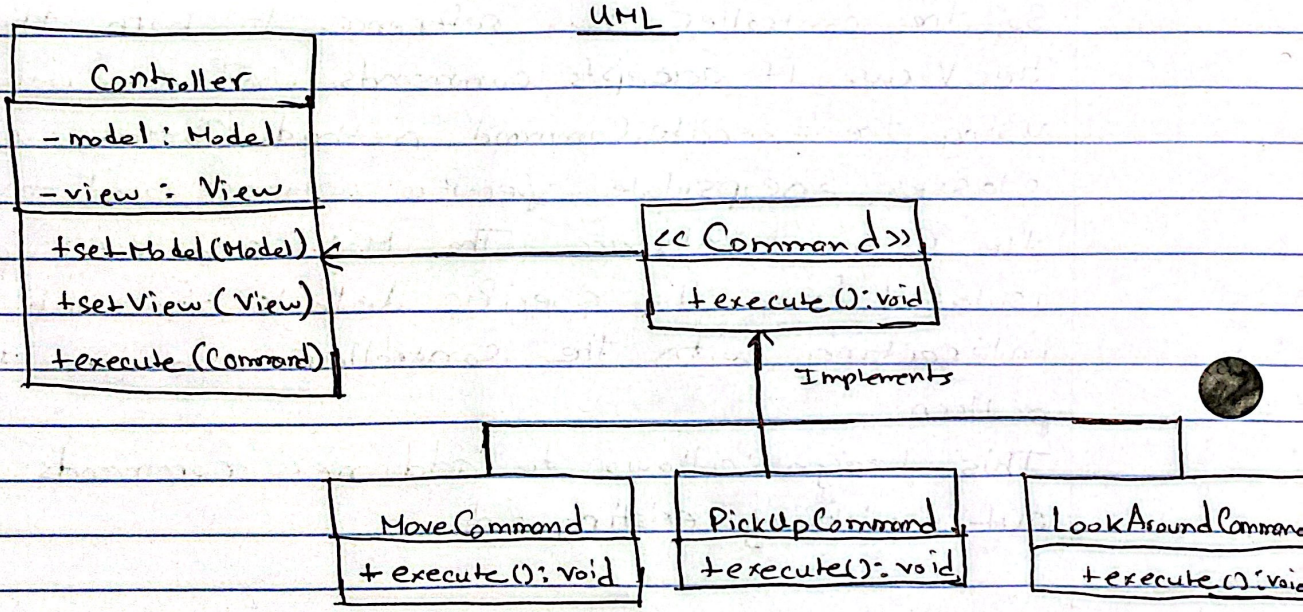
12) pickUpItem() — picks up the item as per the users choice.

13) makeAttempt() — Attempt should not reduce health points if the pet or two players are in same room. If both the conditions are false then an attempt is made and health point reduces by 1 if poked or by the hit points of the item used.

Controller

I will use command design pattern like before. Each command object represents a specific action, and the controller can invoke these commands without having knowledge of their specific details.

UML



Key points:

* Controller: The main controller that interacts with the model and the view to communicate with them.
* Command Interface: An interface defining the execute method that concrete command classes must implement.
* MoveCommand, PickUpCommand, LookAroundCommand: Concrete command classes representing specific actions. Each implements the 'Command' interface.

* Model: The model component, representing the game state and logic.

* View: The view component, responsible for displaying the game state.

So, the controller has reference to both the Model and the View. It accepts commands and executes them using the 'executeCommand' method. Concrete command classes encapsulate specific actions and implement the Command interface. The Model and View are kept isolated from the specific details of each action by interacting with the controller through the command pattern.
This design allows to add new commands easily without modifying existing code.

Controller Test

1) Test moving the Player: The player should be moved to the specified room and the view should be updated accordingly.

2) Test moving the Player to invalid room: The player should not be moved and the view should display an error message

3) Test Player Pickup: The player's inventory should contain an item ~~in a room~~ that was picked up.

4) Test Player pickup in a Room with no Item: The player's inventory should remain unchanged and the view should display a message indicating no items to pick up.

5) Test Player LookAround: The view should display information about the neighboring rooms and items in the current room.

6) Test Player Attempts on Target's life: The target's health should decrease and the view should reflect the change.

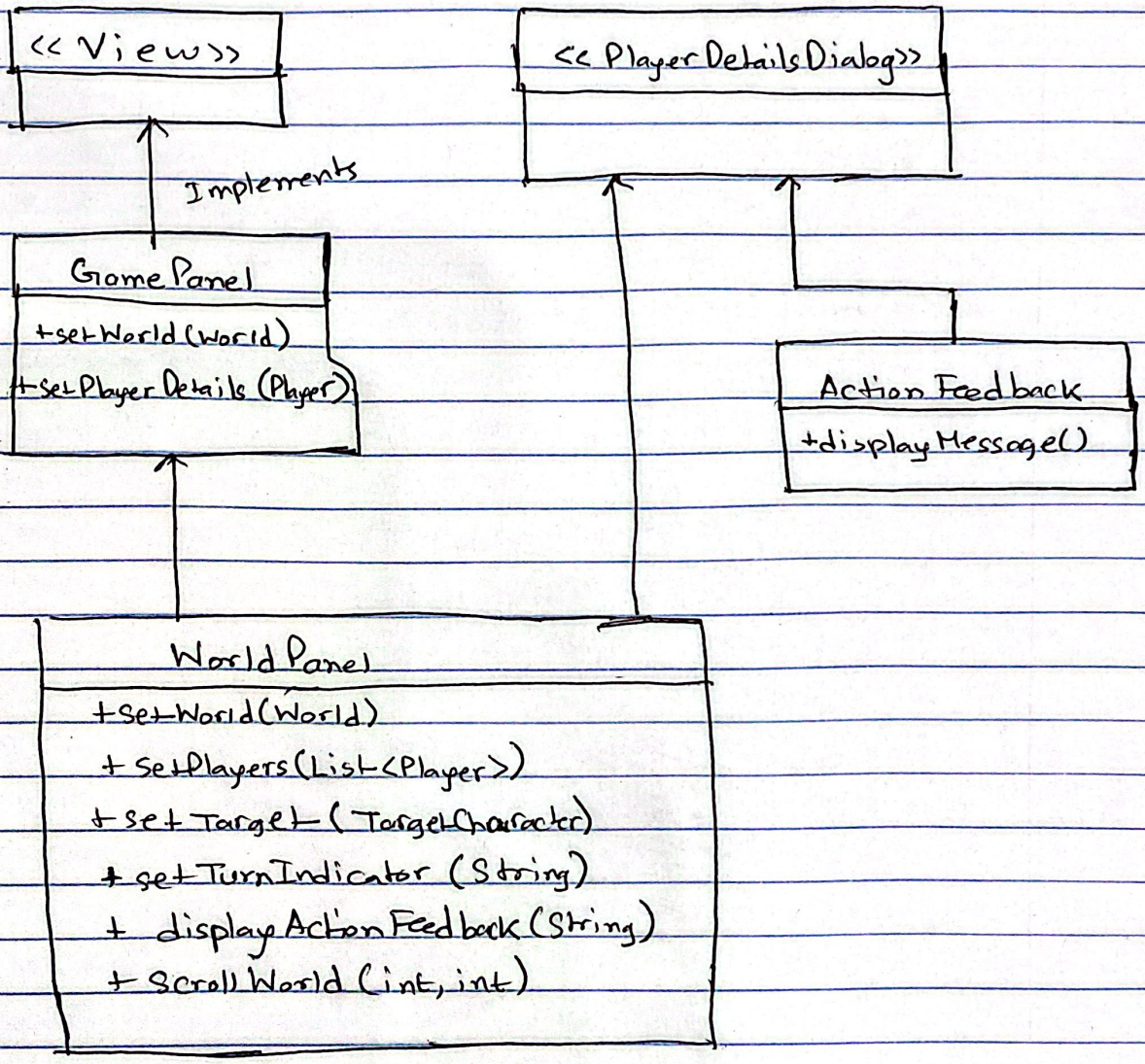
7) Test Player Attempts on Target's life without items: The target's health should decrease by 1 and the view should reflect the change.

8) Test Pet move: Pet should move to a room using DFS principals and the view should reflect the change.

9) Test Character move: Target character's index should be updated by 1 and the view should reflect the change

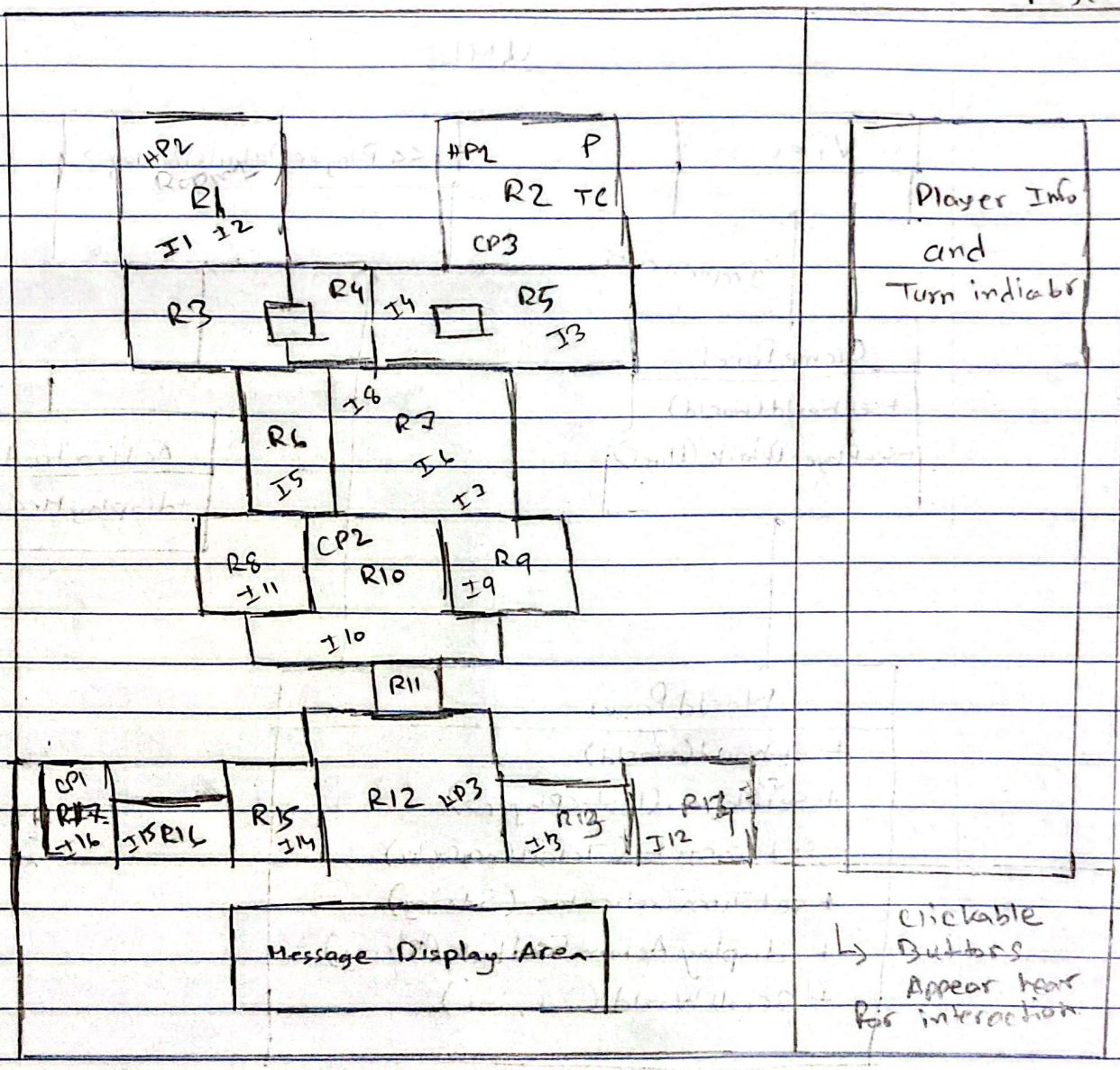
View

UML


<<View>>
<<PlayerDetailsDialog>>
Implements
GamePanel
+setWorld(World)
+setPlayerDetails(Player)
ActionFeedback
+displayMessage()
WorldPanel
+setWorld(World)
+setPlayers(List<Player>)
+setTarget(TargetCharacter)
+setTurnIndicator(String)
+displayActionFeedback(String)
+ScrollWorld(int,int)

P -> Pet
TC -> Target
R -> Room
I -> Item
HP -> Human player
CP -> Computer player

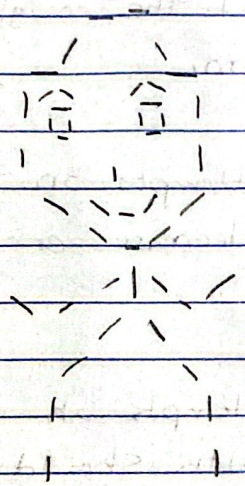
UI Sketches



600x600

Game Screen

Welcome to Dr Lucky's World



1. New Game
2. Continue
3. Quit

600 X 600

Welcome Screen