Observer Design Pattern

The purpose of this pattern is to show how alternative player displays are updated whenever there is a change to the draw and discard pile display in the center of the board. Class descriptions and how they correlate to the structure of an Observer Pattern are found below.

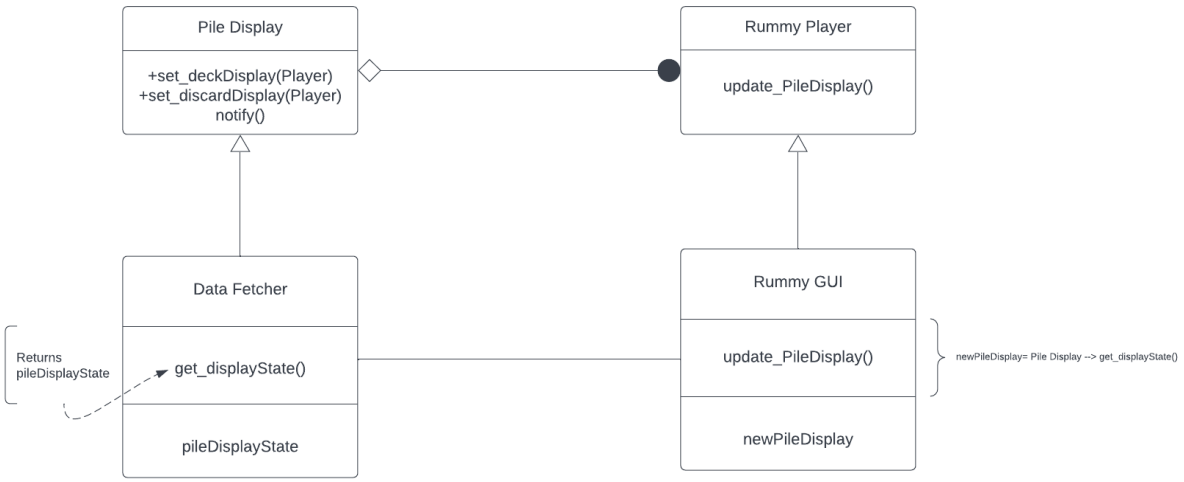
- Names:
    - Subject: “Pile Display”
        - This class takes in a “+*set_deckDisplay(Player)”* method, a “*+set_discardDisplay(Player)”* method which set the changes made to the deck/discard display by the player and a “*notify()”* action
    - Concrete subject: “Data Fetcher”
        - Gets the state of the current “Pile Display” through a *“get_displayState()”* method and passes this as a *“pileDisplayState”* return value.
    - Observer: “Rummy Player”
        - Receives “Pile Display” updates and displays it to the Rummy Players in the game
    - Concrete Observer: “Rummy GUI”
        - Gets subject state from “Data Fetcher” and updates the display for Rummy Player using the *“update_PileDisplay()”* method and returns it to Rummy Player as *“newPileDisplay”*.

- Problem Description:
    - Need to update all rummy players (observers) in game whenever the deck/discard pile is changed

- Solution Description:
    - The “Pile Display” will set a new display once a player draws or discards a card. These changes are grabbed by the “Concrete Subject” which returns the current state of “Pile Display” and sends it to “Rummy GUI” which will then take in those display changes and update all of the “Rummy Players”.

Justification: Need a display update of the center deck/discard pile because without it, players would not be able to see any cards that another player draws or discards thus making it impossible for the game to progress.

Observer Pattern Diagram



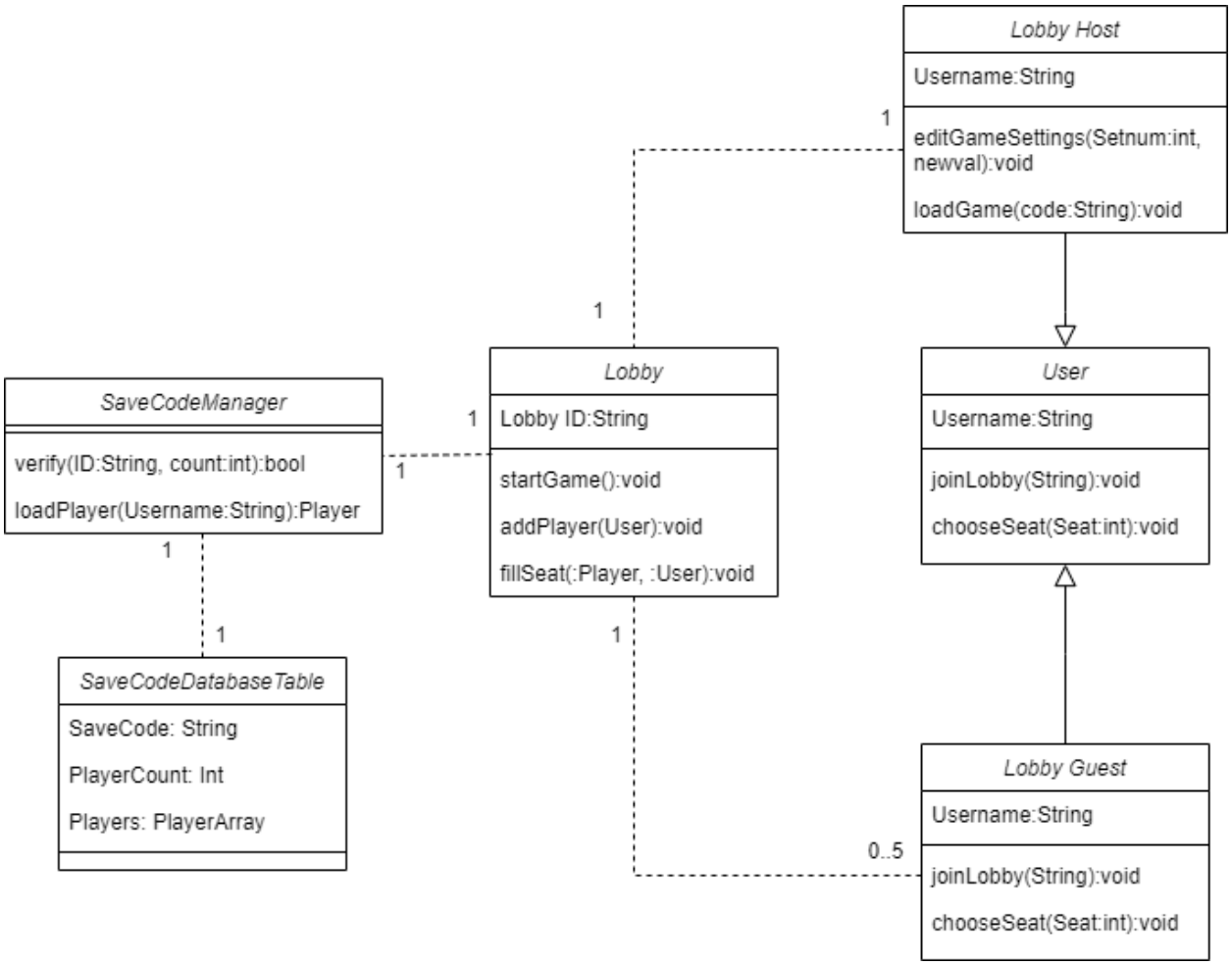
DCD Part 1: Lobby

Lobby Host, User, Lobby Guest, and Lobby reprise their roles from the Lobby Domain Model.

- Lobby has gained the fillSeat(:Player, :User):void method so that loaded player data can be given to a User.

SaveCodeManager combines the classes of PlayerChecker, SaveCodeController and SaveCodeVerifier from Sequence Diagram 5:2 and the Data Fetcher and Data Verifier classes from Sequence Diagram 5:4, and serves to assist in the loading of previously saved games from the database.

- The verify method fills the role of PlayerChecker, SaveCodeController and SaveCodeVerifier
- The loadPlayer method filles the role of DataFetcher and DataVerifier.



DCD Part 2: Gameplay

The functionality of the TurnChecker class from the Sequence Diagrams has been given to the Board class in the form of the isTurn(:Player):bool function.

The functions of the LegalMoveController and IllegalMoveController have been split between the Board and Player classes, where Player has been given the attempt(:moveEvent):void method, and Board has been given the verifyMove(:moveEvent):bool method.