Specification dev

Team Dev

February 2025

1 Introduction

This specification outlines the front-end requirements for the "6 Takes!" card game. In "6 Takes!", players are dealt numbered cards and must strategically choose which card to play to avoid picking up cards that carry penalty points. The game combines careful planning with a bit of luck, providing a fun and engaging experience. This document sets the guidelines for creating an intuitive, responsive user interface that brings the game's dynamics to life.

2 Specification

2.1 Types

2.1.1 Card

Operations :

- **createCard(value : int, bullHeads : int) → Card**
  - **Description** : Creates a card with the specified value and number of bull heads.
  - **Precondition** : $1 \leq$ value ≤ 104 and bullHeads ≥ 0.
  - **Postcondition** : Returns a valid Card instance with the specified attributes.
- **displayCard(card : Card) → void**
  - **Description** : Displays the card on the game screen.
  - **Precondition** : The card is valid.
  - **Postcondition** The card is correctly displayed on the screen.

2.1.2 Player

Operations :

- **chooseCard() → Card**
  - **Description :** Allows the player to select a card to play from their hand using the user interface.
  - **Preconditions :** The player's hand must contain at least one card.
  - **Postconditions :** Returns the selected card to be played in the current round.
- **chooseRow(card : Card, board : Board) → int**
  - **Description :** If the played card is the smallest among those played and no valid row is available for placing it, this operation allows the player to choose the index of the row they wish to pick up.
  - **Precondition :**
    - The card is the smallest among those played in the round.
    - No row in the board accepts the card (i.e., for every row, the last card has a higher value than the played card).
  - **Postcondition :** Returns the index of the row selected by the player to be picked up.
- **logIn(username : string, password : string) → Player**
  - **description :** Authenticates and connects a player to the game system by verifying their credentials via the authentication mechanism.
  - **Precondition :** The username and password must be non-empty and match the stored credentials in the system's database.
  - **Postcondition :** Returns the connected Player instance if authentication is successful ; otherwise, raises an authentication error.
- **quitGame() → void**
  - **Description :** Allows the player to exit the current game. This operation updates the game state to remove the player and free the resources associated with their session.
  - **Preconditions :** The player must be currently connected and participating in a game.
  - **Postconditions :** The player is removed from the game, and their state is updated to reflect that they are no longer active.
- **logOut() → void**
  - **description :** Disconnects the player from the game system, en-

ding their authentication session.
  - **Precondition :** The player must be connected to the system.
  - **Postcondition :** The player is disconnected, and their connection state is updated accordingly.
- **joinLobby() → void**
  - **description :** Allows the player to join an existing game lobby.
  - **Precondition :** A lobby must exist and be open for new players.
  - **Postcondition :** The player is added to the lobby.

2.1.3 Host (Child of Player class)

Operations :
- **CreateLobby() → void**
  - **Description :**Allows the host to create a new game lobby so that other players can join the game.
  - **Preconditions :**The host must be authenticated and not already in an existing lobby or game.
  - **Postconditions :** A new lobby, with default parameters, is created and associated with the host.
- **setParameters(parameters : Object) → void**
  - **description :** Allows the host to define or modify game parameters (such as the maximum number of players, difficulty level, time limit, etc.).
  - **Precondition :** The new parameters must be valid
  - **Postcondition :** These parameters are displayed to players joining the lobby.
- **startGame() → void**
  - **description :** Allows the host to officially launch the game once all start conditions are met.
  - **Precondition :**
    - The lobby must exist and all necessary parameters must be defined.
    - The minimum number of players (if required) must be present.
  - **Postcondition :**
    - The game starts and the lobby state changes to "In Progress" or "Started."
    - All players in the lobby are notified or redirected to the active game interface.

— The game resources and states are initialized for gameplay

2.1.4 Hand

The Hand type represents the collection of cards held by a player.
Operations :

- **removeCardFromHand(hand : Hand, card : Card) → Hand**
  - **Description :** Removes a card from the player's hand when it is played.
  - **Preconditions :** The card must be present in the hand.
  - **Postconditions :** The hand no longer contains the specified card.
- **displayHand(hand : Hand) → void**
  - **description :** Displays all the cards in the player's hand.
  - **Precondition :** The hand is valid (even if empty).
  - **Postcondition :** The cards in the hand are displayed on the screen.
- **addCardToHand(hand : Hand, card : Card) → Hand**
  - **description :** Adds a card to the player's hand (useful during dealing).
  - **Precondition :** The hand is valid.
  - **Postcondition :** The card is added to the hand.

2.1.5 Board

The Board type models the game table as rows of cards. It is structured as a list of rows, each being an ordered list of cards in ascending order.
Operations :

- **initializeBoard() → Board**
  - **Description :** Creates a new board with 4 initialized rows, each containing a starting card.
  - **Preconditions :** None
  - **Postconditions :** Returns a board with 4 valid rows ready for gameplay.
- **addCardToRow(board : Board, card : Card, rowIndex : int) → bool**
  - **description :** Adds the card to the end of the specified row on the board, maintaining ascending order.

- **Precondition :** The card must be greater than the last card in the specified row.
- **Postcondition :** If the addition is valid, the card is added to the row and returns true ; Otherwise, returns false.

— **replaceRow(board : Board, rowIndex : int, player : Player, card : Card) → void**
- **description :** When a row is full, the player picks up the row (accumulating the bull heads from the cards) and the row is reset with the new played card.
- **Precondition :** The row must be full according to game rules.
- **Postcondition :**
  - The player's score is updated with the total bull heads from the row ;
  - The row now contains only the new card.

— **displayBoard(board : Board) → void**
- **description :** Displays the current state of the board by showing the cards in each row.
- **Precondition :** The board is valid.
- **Postcondition :** The board state is correctly displayed on the screen.

2.1.6 Game

The Game type encapsulates the overall state of the game. It integrates the board, the players, and the complete deck of cards.

Operations :

— **displayScores() → void**
- **Description :** Displays the scores of all players.
- **Preconditions :** The game is either finished or in progress with scores calculated.
- **Postconditions :** The scores are displayed on the screen.

— **displayLobbies() → void**
- **description :** Displays the available game lobbies.
- **Precondition :** At least one lobby must exist.
- **Postcondition :** The list of lobbies is displayed on the screen.

3 UML Diagram and Description

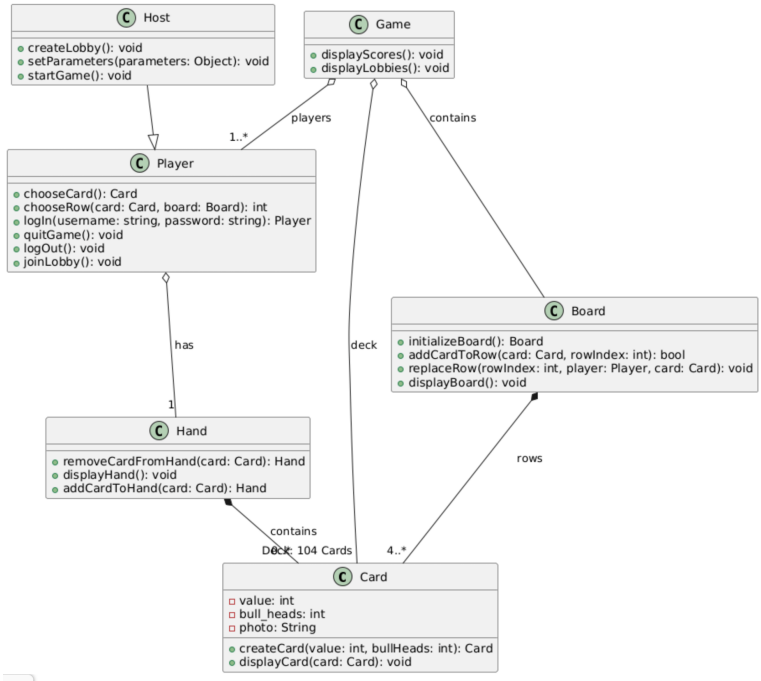

FIGURE 1 – UML diagram of the main classes and their relationships.

3.1 Class Descriptions

Card :
This class represents the data model of a card. It contains the essential attributes (`value`, `bull_heads`, `photo`) as well as methods to create and display a card.

Hand :
Represents a player's hand, that is, the collection of cards they possess. The relationship indicates that a Hand contains zero or more cards.

Player :
Models a game player and provides operations such as choosing a card or a row. Each player has a Hand, as indicated by the has association.

Host :
A specialization of the Player class that includes additional functions to create a lobby, set parameters, and start the game.

Board :
Represents the game board in the form of rows of cards (ordered by value). It offers operations to initialize the board, add a card to a row, replace a full row, and display the board.

Game :
This class encompasses the overall state of the game by aggregating the Board, the Players (each with a Hand), and the complete deck of cards. It also contains operations to display scores and lobbies.

This UML diagram highlights the key relationships : each player has a hand that contains cards, the board is made up of several rows of cards, and the overall game integrates these different components to form the complete state.