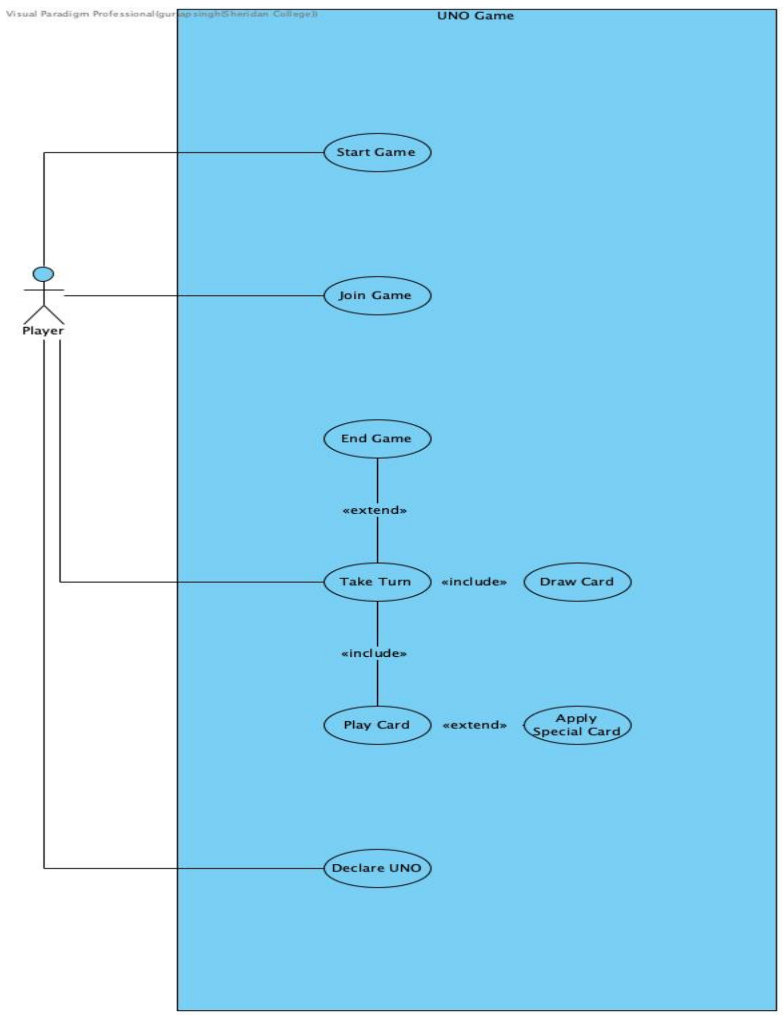
Deliverable - 2

Group – Code Crusaders

Members:

- Gurjap singh – 991814037
- Sankalp Taneja – 991819580
- Jaspinder Kaur - 991810405

Use Case Diagram:



Use Case Narrative

1: Register Players

Use Case Name: Join Game
Primary Actor: Player
Goal: Add all players to the game before it starts.

Preconditions:

- The program is running.
- No game has been started yet.

Postconditions:

- All players are added to the game with their names.

Main Flow:

1. The system asks how many players will play.
2. The user enters a number (minimum 2).
3. The system asks for each player's name.
4. The user types each name.
5. The system creates a player object for each name and stores it.
6. The system shows a message saying that all players are registered.

Alternate Flow:

- If the entered number is less than 2, the system shows an error and asks again.

2: Start Game and Deal Cards

Use Case Name: Start Game
Primary Actor: Player
Goal: Start the UNO game and deal cards to each player.

Preconditions:

- All players are registered.
- The UNO deck is created.

Postconditions:

- Each player has 7 cards.
- The discard pile has one starting card.
- The first player is selected.

Main Flow:

1. The player chooses “Start Game”.
2. The system shuffles the deck.
3. The system deals 7 cards to each player.
4. The system places the next card from the deck on the discard pile.
5. The system shows which player will start first.

Alternate Flow:

- If the starting card is Wild Draw Four, the system draws another card as the first discard card.

3: Take Turn

Use Case Name: Take Turn
Primary Actor: Current Player
Goal: Allow the player to play a card or draw a card.

Preconditions:

- The game has already started.
- It is this player’s turn.

Postconditions:

- A card is either played or drawn.
- The turn moves to the next player.

Main Flow:

1. The system shows the player's hand and the top discard card.
2. The player chooses a card to play.
3. The system checks if the card is valid.
4. If valid, the card is placed on the discard pile.
5. If the player has one card left, the system asks them to say “UNO”.
6. If the player has zero cards, the game ends.
7. If not, the turn goes to the next player.

Alternate Flow 1 – Invalid Card:

- If the card does not match, the system shows an error and asks again.

Alternate Flow 2 – No Playable Card:

1. The player chooses to draw a card.

2. The system gives the player one card from the deck.
3. The turn ends.

4: Play Special Card

Use Case Name: Play Special Card
Primary Actor: Player
Goal: Apply the effect of a special UNO card.

Preconditions:

- A special card has just been played.

Postconditions:

- The game updates turn order or card effects based on the special card.

Main Flow:

1. The system checks which special card was played.
2. If it is Skip, the next player loses their turn.
3. If it is Reverse, the direction of play changes.
4. If it is Draw Two, the next player draws two cards.
5. If it is Wild, the current player chooses a color.
6. If it is Wild Draw Four, the player chooses a color and the next player draws four cards.
7. The system moves to the correct next player.

5: Declare UNO and End Game

Use Case Name: Declare UNO
Primary Actor: Player
Goal: Let the player declare UNO when they have one card, and end the game if they win.

Preconditions:

- The player now has one card after playing.

Postconditions:

- UNO is declared or skipped.
- The game may end if the player wins.

Main Flow:

1. The system sees the player has one card.
2. The system asks the player to declare "UNO".
3. The player declares UNO.
4. The game continues normally.

Alternate Flow – Player Forgets:

- If the player does not declare UNO, the system simply continues the game (no penalty in our version).

Winning Flow:

1. When the player plays their last card, the system checks their hand.
2. If the hand is empty, the system announces the winner.
3. The game ends.

Class Diagram

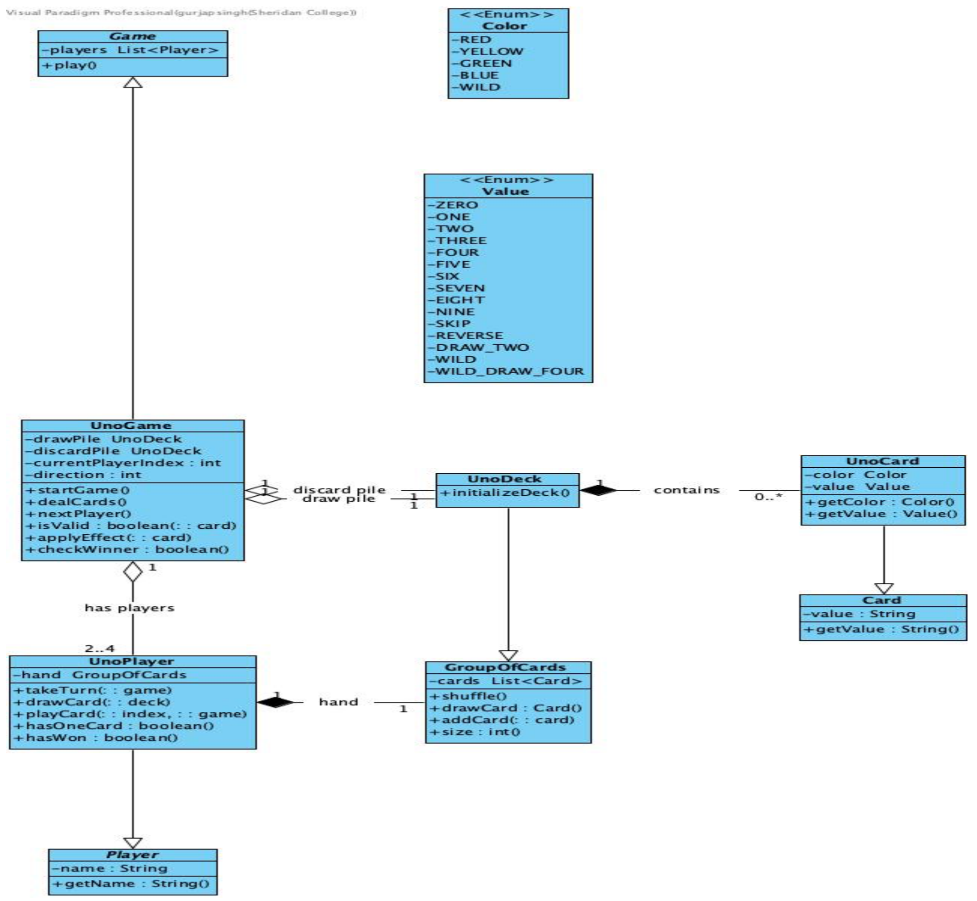


Figure 1

syst 17796 Deliverable 2

design document template

Overview

Project Background and Description

- We are developing a console card game based on SYST17796 starter code as a console game on UNO. The initial code provides us with such basic classes as Card, GroupOfCards, Player and Game. To develop this design in Deliverable 2, we are contributing more specific classes and rules that belong to UNO.

- UNO game will enable 2-4 players to participate. Each opponent is issued with 7 cards. Players, in turn, **must** play a card matching the color or value with the discard pile. In case, they are incapable of playing, they **must** draw a card. Special cards such as Skip, Reverse, Draw Two, Wild and Wild Draw Four are also incorporated in this game. When one of the players has no cards to play, the game is over.

- Deliverable 2 aims to explicitly outline the application of the use case and class design and then write the complete program in Deliverable 3.

Design Considerations

- The class diagram of our UNO game is presented in figure 1. On the top we retain the original structure Game has a list of Player objects and a play() method. Based on this framework we generate UNO-specific classes. UnoGame is an extension of Game, UnoPlayer is an extension of Player, UnoDeck is an extension of GroupOfCards and UnoCard is an extension of Card. UnoCard uses enums Color and Value to store the colour of the card (RED, YELLOW, GREEN, BLUE, WILD) and the value (09 special values SKIP, REVERSE, Drawtwo, wild, WILDDRAWFOUR).
- The key associations include the following. Auxiliary to e.g. one UnoGame object are a few UnoPlayer objects; the degree of multiplicity adjacent to the UnoPlayer is 2..4, i.e. an object of type UnoGame must have two to four players, and there is a label indicating the relationship between UnoGame and UnoPlayer, which is the has label. A GroupOfCards is the hand of each UnoPlayer (composition: when removing the player the hand also goes). GroupOfCards contains a set of Card objects and can offer such facilities as shuffle(), drawCard and addCard.
- UnoGame also has an association with UnoDeck through aggregation. Each UnoGame uses one draw pile and one discard pile (both being represented by UnoDeck objects, the diagram labelling this as the draw pile and the discard pile on the UnoGame side). Each UnoDeck contains 0 UNOCards- there are objects of deck, i.e. a deck can include any number of UNO cards. UnoCard is tied to both the Color and Value enums via its color and value properties namely color and value respectively. These associations and multiplicities are significant in that a single running game can have 2-4 players, and each player has a hand of many cards, and that the game has two decks composed of an excessive number of UNO cards.

- **Encapsulation:**
  All attributes are private. Players do not directly edit deck lists. Cards in the hand can only be changed using methods like addCard() or removeCard().

- **Delegation:**
  UnoGame controls the main game flow but asks UnoPlayer to decide which card to play. UnoDeck handles shuffling and drawing.

- **Cohesion:**

Every class has one job:

- UnoCard only stores card info.
- UnoDeck manages card collections.
- UnoPlayer manages the hand and decisions.
- UnoGame runs the overall game.

- **Coupling:**

  Classes communicate only through methods. UnoGame does not directly access a player's cards list.

- **Inheritance:**

UNO classes extend the base classes:

- UnoCard → Card
- UnoDeck → GroupOfCards
- UnoPlayer → Player
- UnoGame → Game

- **Aggregation:**
  UnoGame contains multiple UnoPlayers (but players could exist separately).

- **Composition:**
  A player's hand belongs completely to that player.
  If the player is removed, their hand is also removed**.**

- **Flexibility/Maintainability**:
  The design is easy to update later.
  New cards or rules can be added by modifying UnoCard or UnoGame without changing the whole program.