University of Lethbridge
CPSC 2720
2021

Ace of Bits
Florence Squad

Sai Kovur, Neil West, Trevor Niggli, Mackenzie Broughton

University of Lethbridge
CPSC 2720
2021

Contents

University of Lethbridge
CPSC 2720
2021

Introduction

Ace of Bits is software that will implement three different card games: Go Fish, Crazy 8's, and Uno.

Go Fish

This is a game with simple rules targeted at children, which uses a standard 52 card deck. The objective of the game is to collect the highest number of pairs among all the players.

Game Start:

- If there are 2 or 3 players, each player is dealt 7 cards
- If there are 4 or 5 players, each player is dealt 5 cards
- All the cards left over in the deck are placed in "the pool".

Within a turn:

- On a player's turn, they can ask another player if they have a card with a specific number on it.
    - If the questioned player has a card of that value, then they must hand it over to the player who asked them
    - If the questioned player does not have a card of that value, then the player who asked must draw a card from the pool
- If a player finds a pair of cards in their hand with the same face, they can remove that pair from their hand and increase their pairs count.

End of the game:

- The game ends as soon as the pool is empty
- Players can then count any more pairs they have within their hand, and whoever has the most pairs total wins.

Crazy 8's

This is another simple game using the standard 52 card deck. The goal of the game is to remove all the cards within your hand.

Game Start:

- Each of the 2-5 players is dealt 5 cards from the deck to their hands, and then the deck is placed in the middle of the table, and the top card is revealed as the starter card.

Within a turn:

- A player can place cards on top of the starter card pile if their card
    - Matches the value of the top card in the starter card pile
    - Matches the suit of the top card in the starter card pile (unless that card is an 8, in which case they must match the suit specified by the player who placed it.
    - Has the face value of 8, in the case of which they get to specify the suit that other players can play on.
- If a player cannot place any card, they have to draw from the deck either until they find a card that they can place, or they draw 3 cards they are unable to place.

End of Round

- A round ends when a player runs out of cards in their hand.
- At the end of a round each player calculates the number of points they gain from the remaining cards in their hand based on the following values:
    - 8 = 50 points
    - King, Queen, Jack, or 10 = 10 points
    - Other cards = face value
- If any player has 100 or more points, then the match ends and the player with the fewest points wins.
- If the game hasn't ended, then start a new round while keeping track of the previously existing scores.

UNO

A shedding game similar to Crazy 8's, however with it's own special deck of 102 cards, and with the opposite goal of getting the most points. This implementation will allow for the defensive use of +4 and +2 cards.

Game start:

- The game starts with 2-7 players
- Each player is dealt 7 cards
- The remaining cards in the deck become the draw pile
- The top card from the draw pile is drawn and made the first card in the discard pile
    - If the card is a +4 card, then put it back in the draw pile and reshuffled
    - If the card is a wild card, the first player can choose what colour it is (essentially allowing them to place any card they want)
- The first turn then goes to a random player.

Turn rules:

- On a players turn, they can place a card of their choosing onto the discard pile if
    - Their chosen card matches the face value of the top card on the discard pile (number, action, ect.)
    - Their chosen card matches the colour value of the top card on the discard pile.
    - Their chosen card is a wild card, or a +4 card. In this case they must also specify a colour that their card will represent.
- The player's turn ends when they place their card.
- If the player is unable to place their card, then they must draw a card from the draw pile

Special cards:

- There are 5 types of special cards, each with a unique property
    - Reverse cards change the direction of the turn order.
    - Skip cards skip the next player's turn
    - Wild cards can be placed on any colour, and the player placing the wild card can pick the colour the wild card represents
    - +2 cards force the next player to draw 2 cards
    - +4 cards force the next player to draw 4 cards, and acts as a wild card
- As a house rule, a player can counter a +2 or a +4 card by placing a +2 or +4 card on it, thus adding to the required number of cards the next player must draw.

Uno:

- One of the core rules of the game uno is the requirement for a player to declare uno when they have a single card left in their hand.
- Since this will be a command line game, a player needs to say uno at the end of the command that puts down their second last card. Failure to do so will most likely result in one of the other players calling uno on them, and forcing them to draw two cards.

Round end:

- A round ends as soon as a player discards all the cards in their hand.
- Once a round is over, the winner will receive points according to the cards remaining in their opponents' hands based on the following values:
    - Numbered cards = face value
    - +2, skip, reverse = 20 points
    - Wild, +4 = 50 points
- Once a single player has a certain number of points or more, they are the winner of the game. This value will be determined by the human player of the game.

University of Lethbridge
CPSC 2720
2021

Project Planning

Team Roles

Team Member	Design - Draft	Design - Final	Implementation - Basic	Implementation - Final
Sai Kovur	Phase Lead	QA Lead	Reporting Lead	Design Lead
Neil West	Design Lead	Phase Lead	Design Lead	QA Lead
Trevor Niggli	QA Lead	Design Lead	Phase Lead	Reporting Lead
Mackenzie Broughton	--N/A--	Reporting Lead	QA Lead	Phase Lead

Role Responsibilities

- **Phase Lead**
  - Distributes tasks to be done among group members
  - Keeps track of what is done and what needs to be done
  - Ensure all team members are effectively working on their tasks
  - Takes lead in group meetings
  - Primary repo manager. Handles merge requests
- **Design Lead**
  - Takes lead in software design and implementation
  - Creates and maintains class and sequence diagrams for project planning
  - Primarily handles implementation and bug fixes
  - Ensures designs follow principles taught in class
- **QA Lead**
  - Takes lead in software testing procedures
  - Primary test writer and issue reporter
  - Ensures tests are written by way of procedures taught in class
  - Ensures all necessary aspects of the software are tested
- **Reporting Lead**
  - Writes up report to be submitted at the end of each phase
  - Organizes the work to be done for the report(s).
  - Collects team's contributions to the reports (e.g. design diagrams from Design
  - Lead)
  - Records the team's ideas/action items during meetings.
- **All Roles**
  - Take on what tasks they can when necessary
  - Aren't afraid to ask for help when struggling

- Ask instructors questions and report back if the answer is pertinent to the whole group
- Have a good time

Risk Management

- Requirements/Design/Estimation
  - Features will be established in a prioritisation hierarchy. Features deemed absolutely necessary to the game's functionality will be implemented and tested before features the game could go without, i.e. games behave properly on a round-by-round basis before a point-scoring system is considered. This will prevent time going into a feature that ends up taking too long or too much effort to implement.
  - Classes will be designed such that code can be reused from the first game type in the two subsequent ones.
  - Branches will be utilised for version control when more than one member is working on the project at any given time.
  - Tests and implementation will be written in accordance to design choices in this document.
- People
  - It is expected that all group members will have communication tools at hand so as to stay up to date and active with group tasks, inquiries, and progress.
  - If it is apparent that one group member will be unable to complete a task in an absolute amount of time, it may fall to another group member to finish what they started depending on the task's priority.
  - Should one of the three group members withdraw from the class, the remaining two will open a dialogue with the course instructors to hopefully find a solution if possible.
  - If one of the group members encounters a health issue, it will have to be rectified at the member's own expense. We don't even have dental.
  - Group members will be subject to "crunch" hours if necessary. We don't even have a union.
  - Should a group member wish to file a formal complaint they will take it up with our human resources department (course instructors).
- Learning & Tools
  - If a group member needs more time on a task and it can be afforded to them, it will be given to them before they are assigned another task.

- If a group member simply needs help with completing one of their tasks, it is expected that all group members will help if they are able to.
- Tasks will be assigned to members in accordance with their perceived proficiency and ability.

Development Process

- **Code Review**
  - Code will be implemented after unit tests for that code have been created to assure that it operates as specifications declare.
  - The QA lead will check each for bugs in previously implemented code and file bug reports accordingly.
- **Communication Tools**
  - Primary mode of communication will be a private direct messaging group on Discord. As mentioned above, it will be expected that all group members will stay up to date on unraveling discourses.
  - All members will also have changes in the repository logged in their email so that they may be alerted about the status of the repository even if they are not looking at their discord.
- **Change Management**
  - It will primarily be the phase leader's responsibility to manage the project repository. They will handle merge requests and closing issues.
  - We will create a new branch for each functional component of the program to make it easier for the Phase leader to see what changes have been made before adding them to the main project, which will be rigorously tested before merging back to the master branch.
  - Features will be implemented in order of dependance, with base level classes like the card class being implemented first, and top level components like specific game operations classes last.

Software Design

Design Description

Fundamentally, each of the card games being implemented all have a very similar system for facilitating the exchange of cards between smaller decks of cards. Therefore, the primary focus will be on creating a smooth, versatile, and expendable deck system. Otherwise, there are only minor problems such as scoring (which is unique for each game), and then parsing user input. Each class will be handling output messages individually.

The Card class has three children classes to allow for polymorphic use of the comparator function. This function is declared in the Card class as a virtual function. Faces and suites aren't really particularly different between uno cards and a standard deck of playing cards, so other than using characters to declare the suits (which conveniently all contain unique first letters even if we include uno suits), we can declare the face values with a simple integer value that is consistent with a logical pattern.

The Deck class exists not only to hold a vector of cards, but also to facilitate the majority of the card exchanges required for the games. Any deck can take a card from any other deck or place a card from itself into any other deck. This means a player's hand can simply be handled as a deck of cards, as well as any other place that a known list of cards is needed.

The Player class mostly operates as an interface class for its own instance of the Deck class, but also has a few other attributes to itself. Each of its children classes contain specific functions for each of the three games, as well as any additional information that needs to be kept for the rules of those games. For example, the UnoPlayer and Crazy8sPlayer children classes each have their own implementation of a getScore function, since the way scores are tallied up in each game is different. The GoFishPlayer child class primarily handles keeping tabs on the pairs a player accumulates.

The Players class acts as a convenient storage medium for the Game class to keep track of and handle the players in the game.

The Game class and its children manage the rules of each game, keeping the similar functions in the Game class and allocating any specialized functionality to the children classes.

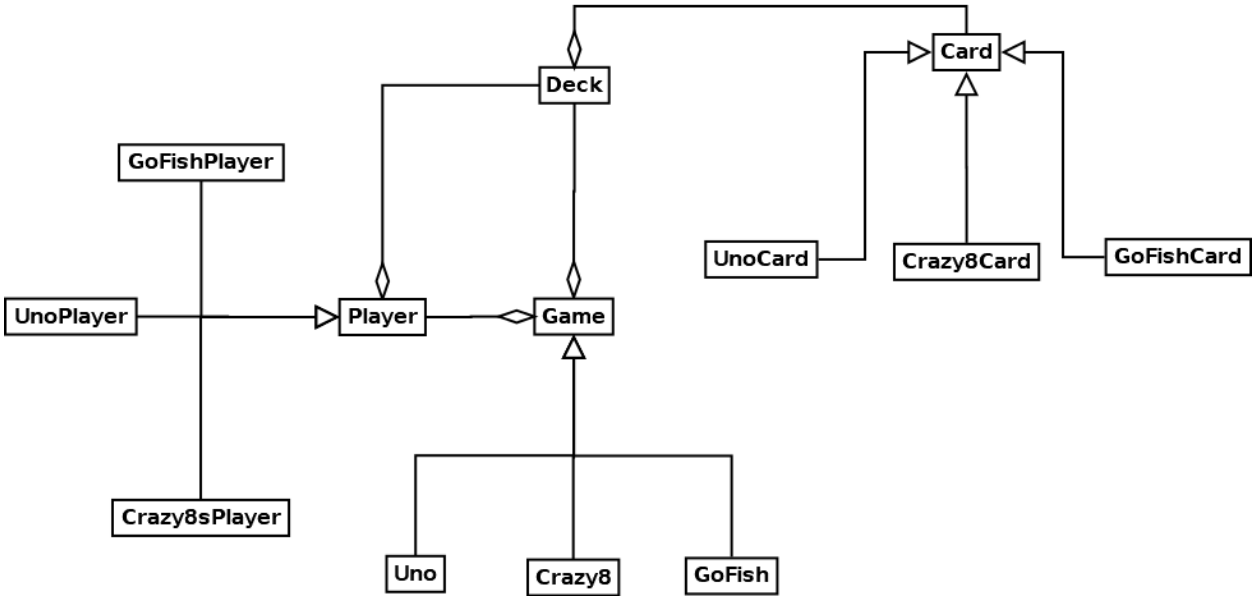
Design Rationale

The class and sequence diagrams below outline our development plans. Our objective is to deliver as quality of card-gaming product as text-driven games can afford.
We will be using polymorphism for our objects wherever reasonable in an effort to stick to the DRY principle in our design. Since we are to be using test-driven development techniques, we are designing our project with a top down approach. We will begin by creating the deck, player, and card systems for the GoFish game. Afterwards, artificial players will be implemented. Once we have that first game working as intended, we will have a general framework to abide by for the two subsequent game modes. This design is a work in progress and may be subject to change.

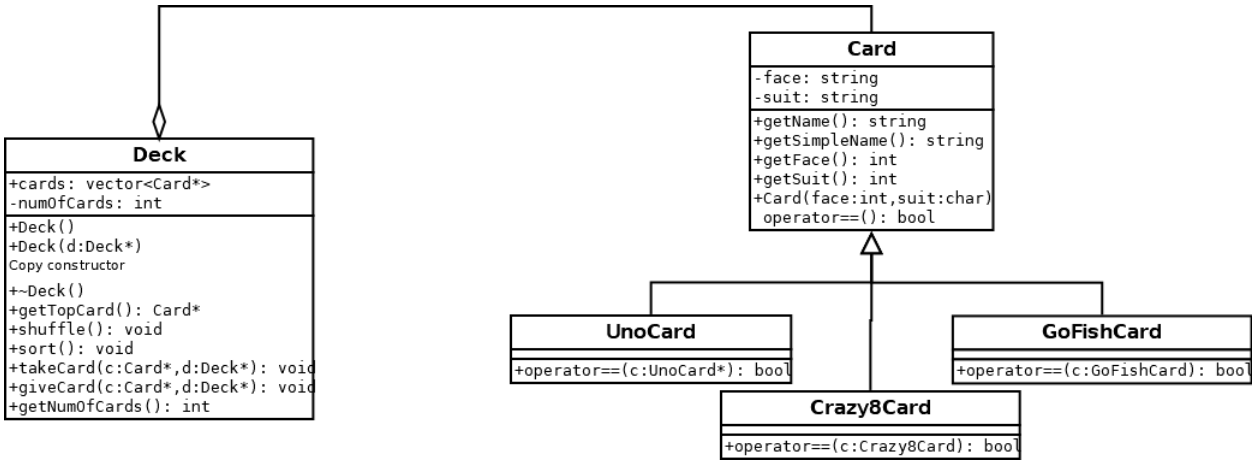
Appendices

APPENDIX A: CLASS DIAGRAMS

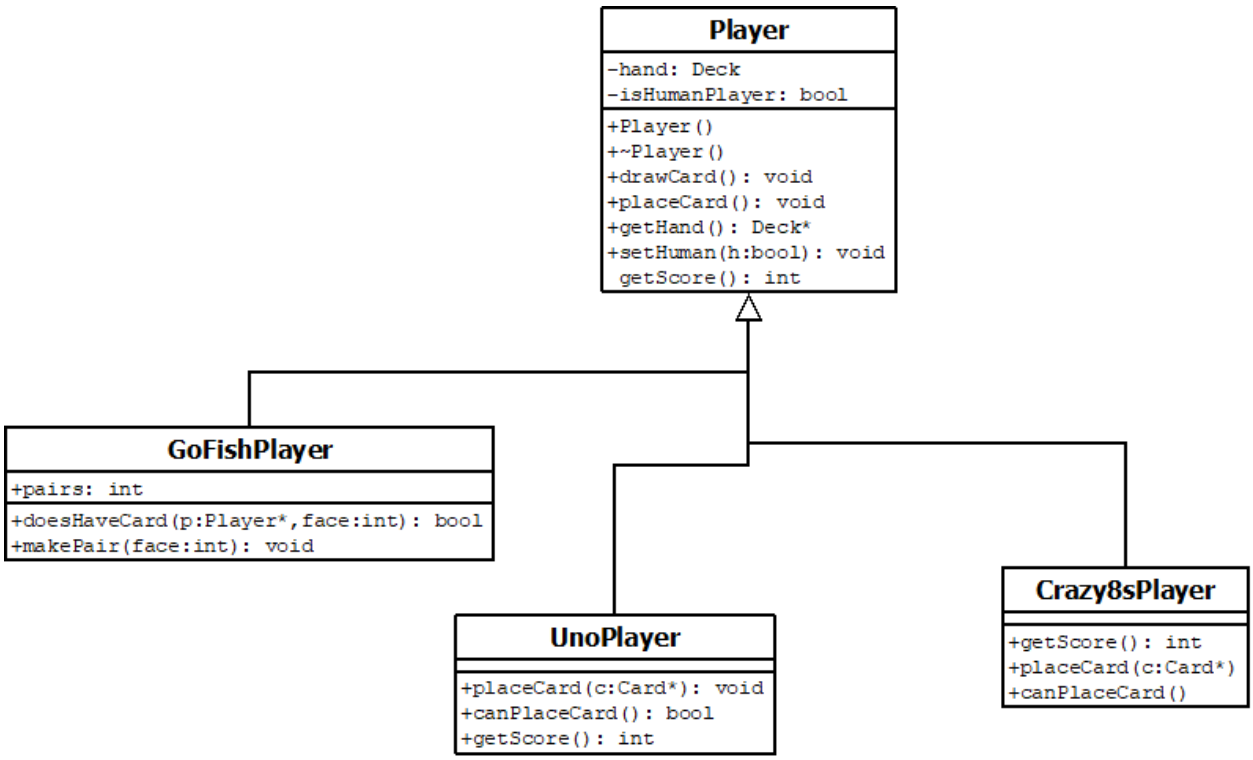
1. Overview



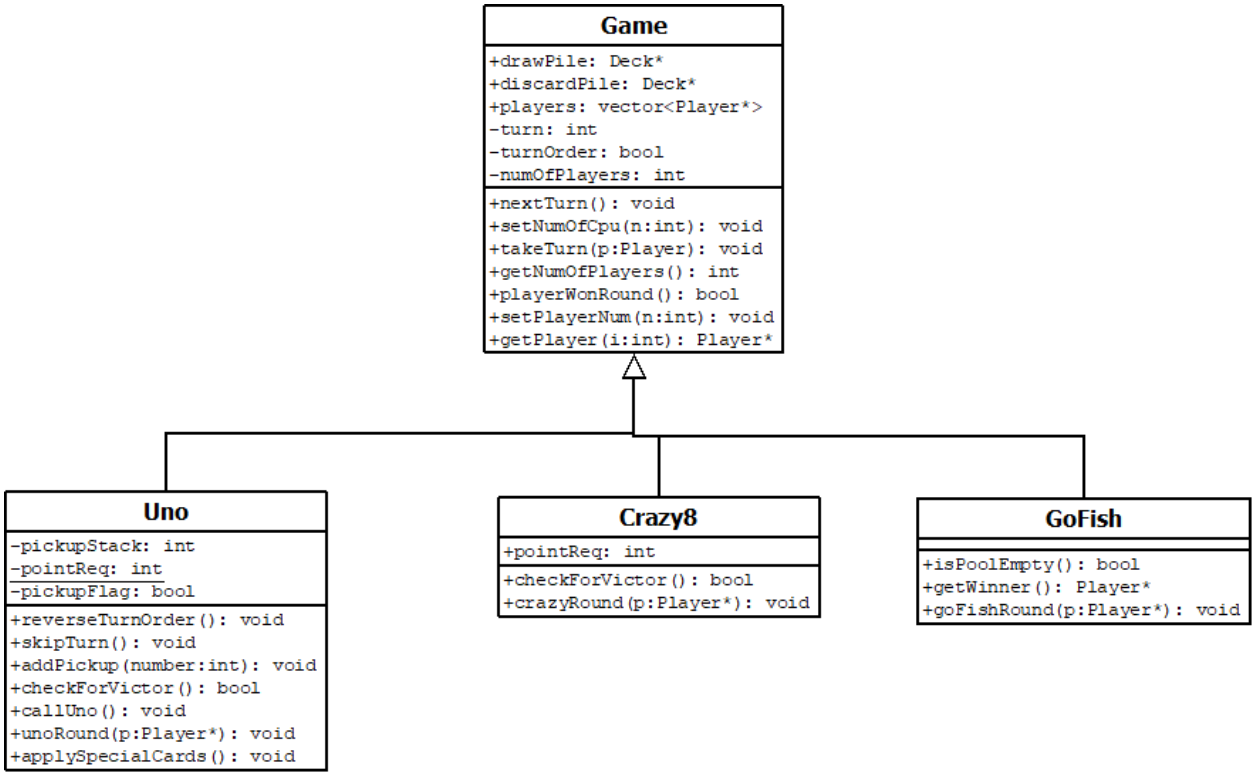
2. Card and Deck Classes



3. Player Handling

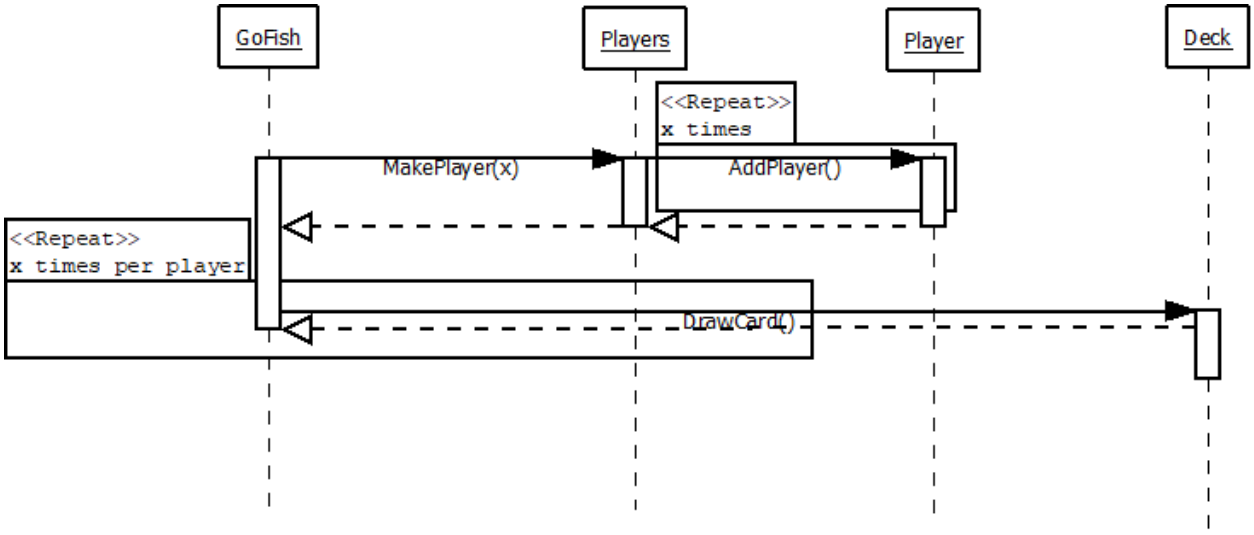


4. Game Handling

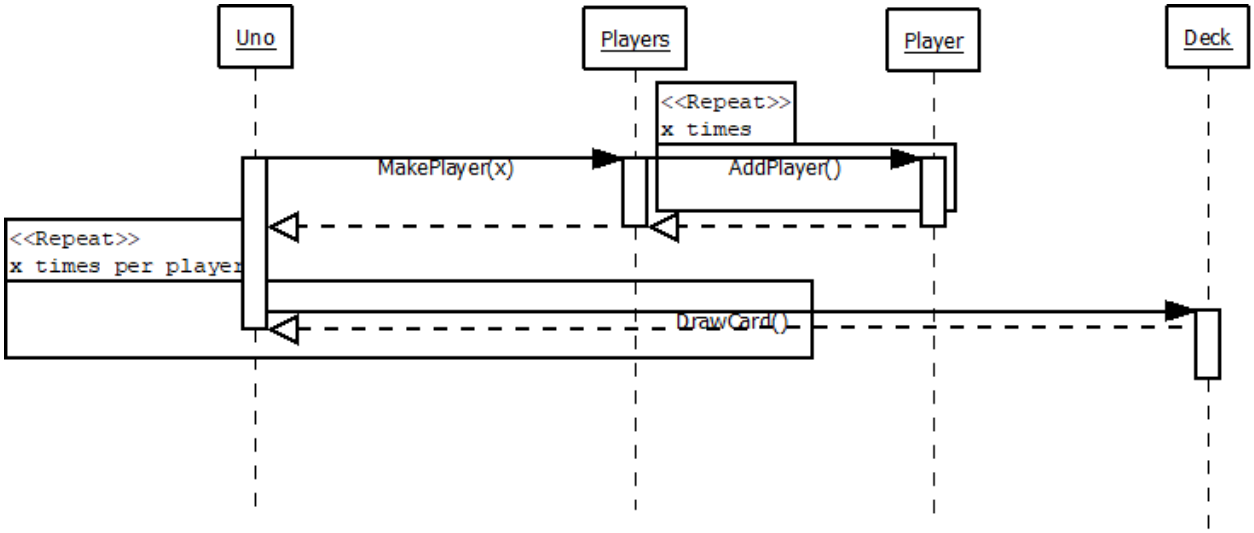


APPENDIX B: SEQUENCE DIAGRAMS

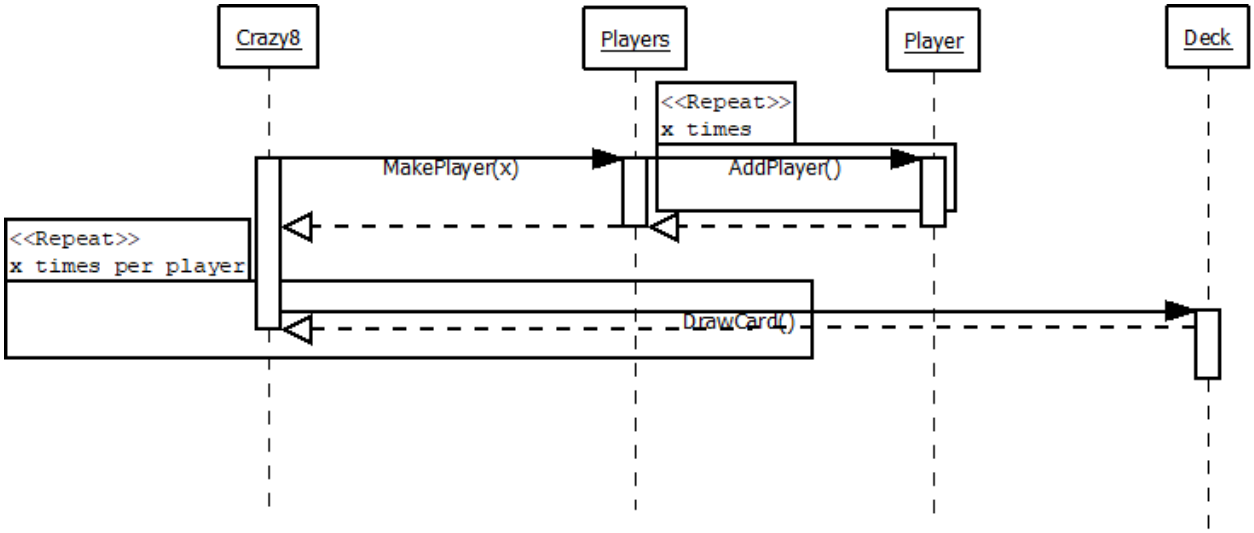
1. GoFish Start



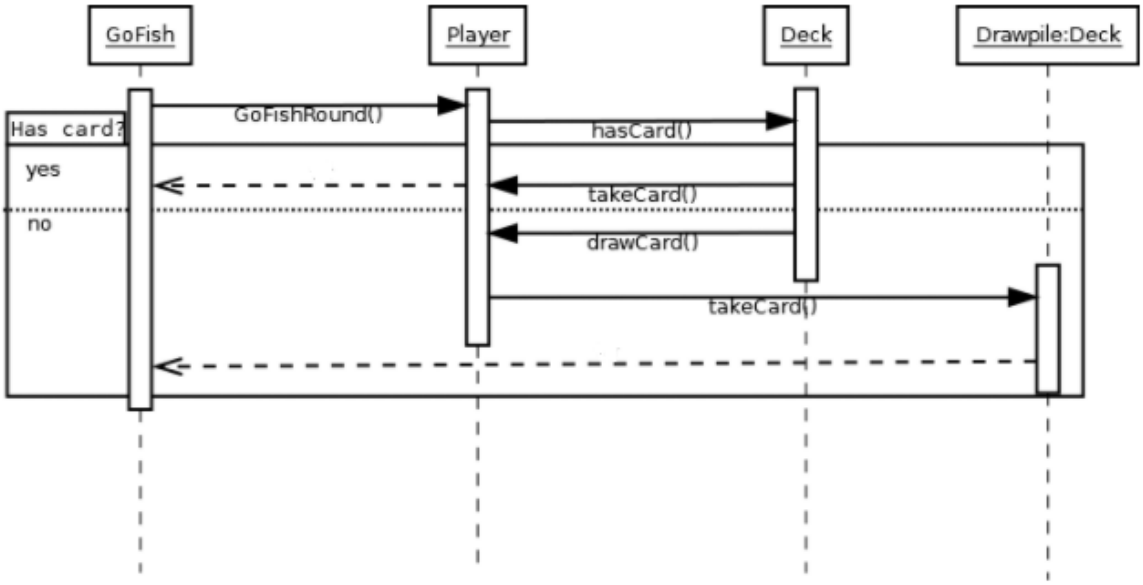
2. Uno Start



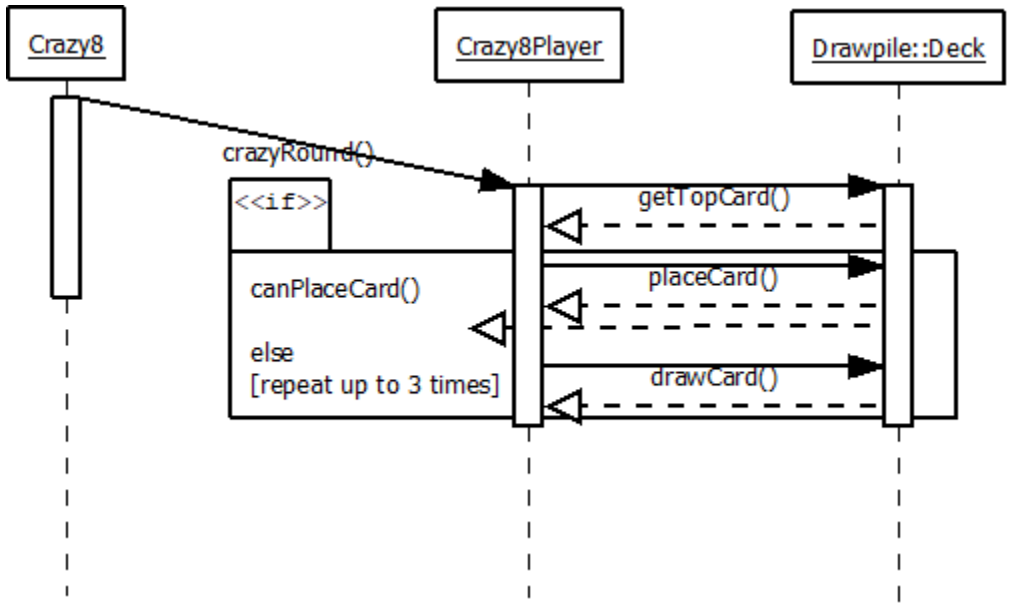
3. Crazy8's Start



4. GoFish Turn



5. Crazy8's Turn



6. Uno Turn