CPSC 2720 2021

[D.U.N.C]

DEVELOPERS (without) UNDERGRADS IN CS

Hutash

Cadence Meszaros
David Awosoga
Noel Roemmele
Ugonna Osuji

Feb 15, 2021

TABLE OF CONTENTS

INTRODUCTION

Welcome to the D.U.N.C. software developed by Develops (without) Undergrads in CS. Created by 2 Math majors, an Applied Stats major, and an MRU transfer student, this software will implement three card games - “Crazy 8’s”, “Go Fish”, and “Jungle Speed”.

Crazy 8’s

This is a solo game, where the objective is to have the highest number of points after the completion of the rounds. The number of rounds is determined by the player. At the end of each round the players points will be shown, with the total points summed and displayed after the game concludes. The player with the highest score at the end of the game will be declared the winner.

Pre Game:

- Played with a standard deck of 52 cards.
- 5 cards are dealt per player.
- The remaining cards are put into a stockpile
- The top card from the stockpile will be revealed and put into a separate pile, called the play pile.

Round Structure:

- The player and the computer will alternate placing a card onto the play pile. In order to play the player must have a card in their hand that matches either the face/number or suit of the top card in the play pile.
    - The player may play an eight instead. This acts as a wild card, where the player can change the suit to whatever they desire.
- If a player is unable to play (no matching suit, number/face, or eight), then they must draw from the stockpile.
    - The player will continue to pick up from the stockpile until they are able to play.
- A player may choose to pick up a card from the stockpile as strategy, even if they do have a playable card in their hand, but they must place a card on the play pile at the end of their turn.
    - The amount of cards they may pick up is unlimited (until the stockpile is empty).
- A round ends when a player runs out of cards. At this time the scoring procedure is put in place.
- After a winner has been declared then the cards are reshuffled and redealt.

Scoring:

- The winner received the sum of the points from the loser’s hand, delegated as such:
    - Each eight = 50 points.
    - Each K, Q, J, or 10 = 10 points.
    - Each ace = 1 point.
    - Every other card is its face value.

Go Fish

This is a solo game, where the objective is to get the most “books”. A book is 4 of a kind, based on face/number value. The number of rounds is determined by the player. At the end of each round the winner will be declared, along with the number of books they collected. After the winner has been declared a prompt will appear asking the player if they would like to continue the game (play another round) or exit. If they choose to continue, another round will be played and their scores from the previous game will be carried over. Therefore, the books can accumulate over the rounds.

Pre Game:

- Played with a standard deck of 52 cards.
- 7 cards are dealt per player.
- The remaining cards are placed into a stockpile, face down.

Round Structure:

- A player will ask their opponent for the rank of a card that they want.
    - Requests can only be made if the player has said rank of card in their hand.
- If the player from which the card has been requested has the requested card, they must give up the card to their opponent. If the player has multiple of that card, they must give up all of them to the player who made the request.
- If the player from which the card has been requested does not have the card, they must say “Go Fish”, at which point the player that made the request will draw a card from the stockpile and place it in their hand.
    - Their turn then ends.
- If an affirmative request was made, then the player makes another card request. They may request the same rank as the previous request or a different rank.
    - They may continue to make requests until they are told to “Go Fish”, then their turn will end
- A book will be automatically counted once there are four of a kind in a player's hand.
    - A message will be sent to confirm that a book has been created, and what the rank of the book is.
- If a player runs out of cards, they must draw from the stockpile and make a request based on the card that they drew.
- The game ends when all 13 books have been collected. The winner is the player with the most books.

Jungle Speed

This is a solo game, where the objective is to get rid of all the cards in your hand. The number of rounds is determined by the player. At the end of each round the player who gets rid of their hand first is declared the winner, and the user will get a prompt asking if they would like to play another round or exit the game.

Pre Game:

- Played with a special deck of 72 cards which consists of different colours and patterns.
- The entire deck is divided between the players, face down, and the players cannot look at their cards.
- The totem is placed in between the players, in arms reach.

Round Structure:

- Both players (user and AI) will alternate flipping the top card from their hand and placing it into a communal discard pile. This process continues until two flipped cards match in shape.
- Once two flipped cards match in shape, there will be a duel between the players.
    - The first player to press the key preselected by the player (analogous to grabbing the totem) will be declared the winner of that round.
    - The loser of that round will take the cards from both discard piles and place them at the bottom of their hand.
    - The player who lost the duel will start the card flipping process again
- The game ends when a player no longer has any cards in their hand.
    - This player will be declared the winner.

Special Rules:

- If a player accidentally presses the key when the flipped cards do not match, they automatically lose the round and take the cards from both discard piles, placing them into their hand.

Other Notes:

Time permitting, the D.U.N.C. software will implement the following:

- Different skill levels of the AI whom they are playing against.
    - For Crazy 8's and Go Fish different common strategies will be added to the levels of AI,
    - variable reflex speeds for the AI will be enabled in Jungle Speed.
- Dialogue between the player and AI may also be implemented.
- Ascii Output of cards
- The ability to play against more than two players
- For Jungle Speed, the following special cards will be reintroduced:
    - If an "All In" card is played, both players must press the totem key as quickly as possible, and the same rules apply for the loser of the duel.
    - An "All Out Card" will have all players flip a card from their hand
    - A "Colours in" card will change the matching criterion from shapes to colours

PROJECT MANAGEMENT

TEAM ORGANIZATION

Team Member	**Design- Draft**	**Design- Final**	**Implementation-Basic**	**Implementation-Final**
Cadence	Reporting Lead	Design Lead	QA Lead	Phase Lead
David	Phase Lead	Reporting Lead	Design Lead	QA Lead
Noel	QA Lead	Phase Lead	Reporting Lead	Design Lead
Ugonna	Design Lead	QA Lead	Phase Lead	Reporting lead

TEAM ROLES

Phase Lead:

- Understands what needs to be accomplished in the phase.
- Attend office hours to get help for the group, regarding clarification/questions etc.
- In charge of tasks identification and delegation with approval from group members.
- Maintains communication with team members to remain on track.
- Initiate dialogue on identification and resolution of problems.
- Ensure task deadlines are met.

Design Lead:

- Compilation and collection of group design ideas.
- Ensure that team members' software design contributions meet the 2720 guidelines.
- Create and constantly refer back to UML diagrams to ensure that they represent the intended software design.

Quality Assurance Lead:

- Implement rigorous testing procedures.
- Ensures that all code meets requirements before being merged into master branch.
- In charge of test driven development for a respective phase of implementation.
- Uses boundary value analysis and equivalence partitioning for test cases.

Reporting Lead:

- Ensures that the reports are completed and properly assigned to team members.
- In charge of collecting the team members contributions for the reports.
- Scribes team meetings and disseminates actions items during meetings.
- Document team report, in accordance with team members.

RISK MANAGEMENT

Requirements/Design/Estimation

- The team planned a project that is too large.
    - To avoid this issue we will prioritize the classes deemed “essential” over the classes that we declared as “features or extras”
    - We will focus on the essentials first and then the extras, time permitting.
        - Starting with the foundation and working outwards.
- The team underestimated how long parts of the project would take.
    - In the case we run into a part that is larger than we expected we will make sure we are a flexible team. Meaning that the flexible team members are able to be assigned as an aid to that particular branch to ensure its completion.
    - In addition, we can re-evaluate each part to determine if it is essential or an extra.
- Major changes to design are needed during implementation.

    - In this case we will schedule extra meetings throughout the week, or make existing meetings longer to compensate for the major changes.
    - In addition, the team will figure out why we had to make that change. This is for the prevention of making the same mistake.
        - Learning from our mistakes.
- Card games are too simple.
    - If we get to a point where we have finished early, but the card games are too simple we can add extra features to add complexity to the games.
        - This will require extra meetings, or longer meeting times.

People

- Addition or loss of team members.
    - In the case of losing a team member, we would retain our roles and reassign the jobs of the lost member.
    - In the case of gaining a team member, we would prep them with an orientation of our project, along with our progress to date.
        - We would then reassign jobs to even the workload among team members.
- Unproductive team member(s).
    - If we run into the issue of an unproductive team member we would:
        - Assess the reason why the team member is not contributing as much as they are expected to and figure out how to fix the issue.
            - If the team member does not like the jobs they were assigned, or struggling with them, then we can reassign them to a better fitting team member.
- Team member(s) lacking expected technical background.
    - If there are team members with less experience we can assign them to easier jobs that don't require as much technical knowledge.
        - These jobs could be:
            - Testing
            - Brainstormer/idea person
        - This allows the team member to participate in a different way, and still help the team towards the final goal.
    - Another option would be to mentor them.
        - By showing the team member an example of how to do a particular task, they may be able to follow the example and finish the task they were assigned.
- Major life events.
    - We would expect the team member to give good notice for life events that are predictable. This way we can prepare for the change.
    - In the case of an unexpected life event, we expect the team member to be transparent and communicate to the team what they need.

        - We could lighten the load or reassign some jobs to ensure that the team member can work through their personal situations.
- Conflict between team members.
    - This conflict could be a conflict of ideas, or someone taking complete control of the project.
    - To prevent conflicts from happening the team needs to have good communication and discuss issues as they arise.
        - This means retaining an atmosphere of respect and openness.
    - If a conflict cannot be avoided, we will settle disputes as quickly, and efficiently as possible.
    - A solution for idea conflicts would be to make decisions based on (1) consensus, (2) majority, or lastly (3) leadership.
        - In the third case the phase leader will get the final say.

Learning & Tools

- Inexperience with new tools.
    - If the tool is recommended by Dr. Anivk or presented to us during the lab we will refer to official docs to try to understand how to use the tools.
        - After we read over the docs, if there are still questions the Phase Lead will attend office hours and ask questions on behalf of the team.
    - If a minority of the team are inexperienced with the tools they will be mentored by experienced team members.
    - In addition, we can create test programs to help us understand how the tools/programs work.
- Learning curve for tools steeper than expected
    - If the learning curve is steeper than expected the team will take time outside of class to practice how to use the tools, in order to keep on track with the project.
    - There are online resources that the team can also use to help understand the tools.
    - In addition, the team can work together to help each other understand the tools.
        - This will ensure that the entire team is on the same page.
- Tools don't work together in an integrated way.
    - If the tools we are using are those suggested by Dr.Anvik or Nicole, the Phase Lead will attend office hours and ask questions on how to integrate the tools better.
    - If the suggested tools are not compatible with the teams experience, we can extend to alternative tools that may integrate better.
- Lecture topics don't align with the project.
    - If there are requirements for a phase that havent been covered in lecture yet, the team can read ahead and learn the content on their own time.
    - The Phase Lead can attend office hours and ask questions regarding topics that are unknown to the team.
    - In addition, the team can use external resources to understand the topics.

DEVELOPMENT PROCESS

CODE REVIEW PROCESS

- For push requests:
    - Anyone can push but the changes need to be reviewed by the Quality Assurance lead.
- For merge requests:
    - They will be handled by the Design Lead.
- For triaging bugs:
    - They will be handled by the Reporting Lead.
- Phase Lead will do intermittent checks of code to ensure progress and that the code is meeting requirements.
- If there are major changes that need to be done, these changes will need to be brought up during a meeting and will be decided on by the team.

COMMUNICATION TOOLS

- Use iMessage group chat to discuss minor aspects of the project, and to arrange zoom meetings.
- Every week we have 4 scheduled zoom meetings. Each meeting will be 2 hours long.
- We can also use the Issue tracker on gitlab to communicate.

CHANGE MANAGEMENT

- If we run into a bug, the person who was in charge of that branch will be responsible to fix it. If they cannot fix it then other team members can assist.
- After the bug has been “fixed” the code will be reviewed by the QA lead.
- Issues will be closed by the member who opened them.
- In the case of bugs resulting from incompatible code, this will be addressed by the respective team members who created it.
    - They will then work together to make code compatible.
- Status of a bug will be changed in the issue tracker by the reporting lead.
    - This will require an acceptable tag, indicating that the bug has been fixed, along with a small description/explanation.

SOFTWARE DESIGNS

DESIGN – CLASS DIAGRAMS

We will create these later, but here is how we will structure the UML diagrams.

Diagram 1: All classes

Diagram 2: Players and AI's

Diagram 3: Game and it's children

Diagram 4: Deck and it's children

Diagram 5: Card and it's children

DESIGN – SEQUENCE DIAGRAMS

Refer to Appedencies.

CLASS DESCRIPTIONS

Game:

Public Methods

- **Default Constructor(int players, int decks)** will create a card game with a specific number of players and decks, and will call **createPlayer** to create the player objects
- **~Destructor**
- **void printRules()** will be in charge of displaying the rules of the selected game.
- **player* createPlayer()** will call **setName()** to create a player object and return it to the constructor.
- **void setName():** is the function that will allow the player to input a name of their choice. This can be letters or numbers.
- **string getName(player* player):** returns the name of the player.
- **void printScore()** will output the scores of each player.
- **void printWinner(vector <player*>):** will output the game rankings, using the player names.
- **dealCards(vector <player*>, int handSize):** will deal the cards out accordingly, taking in an int parameter of how many cards need to be dealt to each player.
- **virtual void preGame()=0:** initializes game set up
- **virtual void round()=0:** controls how a round starts, functions, and ends.
- **Virtual void generateAI()=0:**

- **void transferCards(vector <card*> location, vector <card*> destination):** will take in the initial location of the card and the final destination. This function will be in charge of picking up cards, giving cards away, putting down cards, and will be called by dealCards.
- **int chooseGame(int whichGame):** determines which game the player wants to play by taking in int and returning an int to the playGame function.
- **void playGame(int gameNumber):** will take in an int, either 1,2 or 3 which are assigned to a specific game. This will create and run the game, calling the game constructor first

Protected Members
- **int numPlayers**
- **int numDecks**

Deck:

Public Members:
- **Deck(int sizeOfDeck):** Initialize an empty array.
- **virtual Deck generateDeck()=0:** Fill the array with the appropriate type of cards.
- **void shuffleDeck(vector <Card*>):** will randomize the cards before they are dealt.
- **void setDeckSize(int d)**: setter function for deckSize
- **int getDeckSize():** getter function returns size of deck
- **~Deck**

Protected members:
- **int deckSize**
- **stack <Card *> deck;**

StandardDeck : public Deck

Public Members:
- **StandardDeck(int size) :** Deck(deckSize = size) {}
- **~StandardDeck**
- **void generateDeck():** generates a standard deck of playing card

NonStandardDeck : public Deck

Public Members:
- **NonStandardDeck(int size) :** Deck(deckSize = size) {}
- **~NonStandardDeck**
- No methods, acts as an interface.

JungleSpeedDeck: public NonStandardDeck

Public Members:

- **JungleSpeed(int size) :** NonstandardDeck(deckSize = size) {}
- **~JungleSpeed**
- **void generateDeck()**: generates a Jungle Speed deck

Card: A base class

Public Member Functions:

- **Card()**
- **~Card**

StandardCard: public Card

Public Member Functions:
enum Suit {DIAMONDS = 1, HEARTS, CLUBS, SPADES};

- **StandardCard(Suit nameOfSuit, int cardNumber, int CardRank):**
    - Where number is from 1-13
    - CardRank determines which cards are deemed “high”
        - Ex. Ace is low in crazy 8’s
- **~StandardCard**
- **void setValue:** sets the value of the card.
- **void setRank():** sets the rank of a card.
- **void setSuit()** sets a suit of a card.
- **int getValue():** will return the value of a card.
- **suit getSuit ():** will return the suit of a card, by calling suitToString.
- **int getRank()**: will return the rank of the card.
- **string displayNumber():** returns a string implementation of the number/face value of a card
- **string suitToString(Suit)**
- **Void displayCard():** display suit, number/face

Private Members:

- **Suit suitName;**
- **int number;**
- **int rank;**

JungleSpeedCard: public Card

enum Colour {RED = 1, BLUE, YELLOW, GREEN};
Public Member Functions:

- **JungleSpeedCard(string cardName, Colour cardColor):**
- **~JungleSpeedCard**
- **void setColour():**
- **Void setName():**
- **Colour getColour():**
- **String getName():**
- **String ColourToString(Colour c);**
- **void displayCard():** displays card name and colour.

Private Members:

- **String name**
- **Colour colour**

Player:

Public member functions:.

- **Player(string playerName, int playerID):**
- **void displayHand():** shows a player's hand.
- **bool getMyTurn():** returns the value from myTurn.
- **void setMyTurn()**: sets the value of myTurn.
- **void emptyHand():** uses a for loop to delete all card pointers within the hand vector.
- **void setPoints():** will allow us to assign points to a player.
- **int getPoints():** returns the total points earned by a player.
- **Vector<Card *> Hand :** needs to be public so that transferCards can use it as a location or destination
- **void playTurn()**
    - Contains all turn information from turn function from Round from the specific game class

Private member functions:

- **bool myTurn():** will return true or false depending on if it is that specific players turn or not.
- **int points**
- **string name**
- **int ID**

AI: public Player

Public member functions:

Enum Level {EASY =1, NORMAL, HARD}

- **AI(Level AILevel, string playerName, int playerID): Player(playerName, playerID)**
- **~AI**
- **void setDifficultyLevel():** will set the playing level of the AI.
- **Level getDifficultyLevel():** will return the playing level of the AI.
- **void virtual Strategy()=0:**
- **Level stringToLevel():** converts a string to a Level type

Protected members:

- **Level aiLevel;**

JungleSpeed: public Game

Public Member Functions

- **JungleSpeed();**
- **~JungleSpeed**
- **void Rules():** contains the rules of Jungle Speed.
- **void preGame(vector <Player*> Players):** defines preGame set up.
- **void round(vector <Player*> Players):** defines a round.
- **void generateAI():** creates the AI with the selected difficulty level.
    - Calls setLevel
- **double reactionTime():** implement the timer function to determine the players reaction time. Returns total time.
- **Player* declareWinner(vector <Player*> Player, double reactionTime):**
    - If the player's time is less than the AI, then the player will be declared the winner of that round.
    - Else if the player's time is equivalent to the AI, then it'll be a tie.
    - Else, the AI is declared the winner.
- **Void Duel():**
    - Calls reactionTime
    - Calls declareWinner
    - Duel will handle the card transfer.
    - After the card transfer is complete, it will check if any hands are empty.

CrazyEights: public Game

Public Member Functions

- **CrazyEights():**
- **~CrazyEights**
- **void Rules():** will contain the rules of Crazy 8's.
- **void preGame(vector <Player*> Players):** defines pregame set up.
- **void round(vector <Player*> Players):** defines a round.
- **void generateAI():** creates the AI with the selected difficulty level.
    - Calls setLevel.
- **void scoringSystem(vector <Player*> Players):** called at the end of each round.

Private Member Functions:

- **StandardDeck stockPile;**
- **StandardDeck mainDeck;**

Go Fish Class: public Game

Public Member Functions:

- **GoFish():**
- **~GoFish**
- **void Rules():** contains the rules of Go Fish.
    - It will call printRules to print them to the screen
    - A player can ask for the rules at anytime through the game
- **void preGame(vector <Player*> Players):** defines pregame set up.
- **void round(vector <Player*> Players):** defines round.
- **void generateAI():** create the AI with the selected difficulty level
    - Calls setLevel.
- **void makeBook(vector <Player*> Players):** "creates" books for each player
- **void makeRequest(vector <Player*> Players)**: will allow a player to make a card request to their opponent.
- **void goFish(vector <Player*> Players):** when a request is denied, goFish will call transfer cards and take a card from the stockpile and place it in the player's hand.

Private Member Functions:

- **StandardDeck stockPile;**
- **StandardDeck mainDeck;**

JungleSpeedAI: public AI

Public Member Functions:

- **JungleSpeedAI(double response, string PlayerName, Level playerLevel, int playerID) : AI(playerLevel, playerName, playerID)**:
- **~JungleSpeedAI**
- **Void strategy():** sets the strategy depending on the difficulty level of the AI
- **double getResponseTime():** will return the response time of the AI.
- **void setResponseTime:** sets the response time of the AI depending on its level.

Private members:

- **double responseTime**

CrazyEightsAI: public AI

Public Member Functions:

- **CrazyEightsAI(string PlayerName, Level playerLevel, int playerID) : AI(playerLevel, playerName, playerID)**:
- **~CrazyEightsAI**
- **Void strategy():** sets the strategy depending on the difficulty level of the AI

GoFishAI: public AI

Public Member Functions:

- **GoFishAI(string PlayerName, Level playerLevel, int playerID) : AI(playerLevel, playerName, playerID):**
- **~GoFishAI**
- **Void strategy():** sets the strategy depending on the difficulty level of the AI.

APPENDICES

APPENDIX A: FIGURES AND TABLES

Figure 1: How to Start a Standard Deck Game

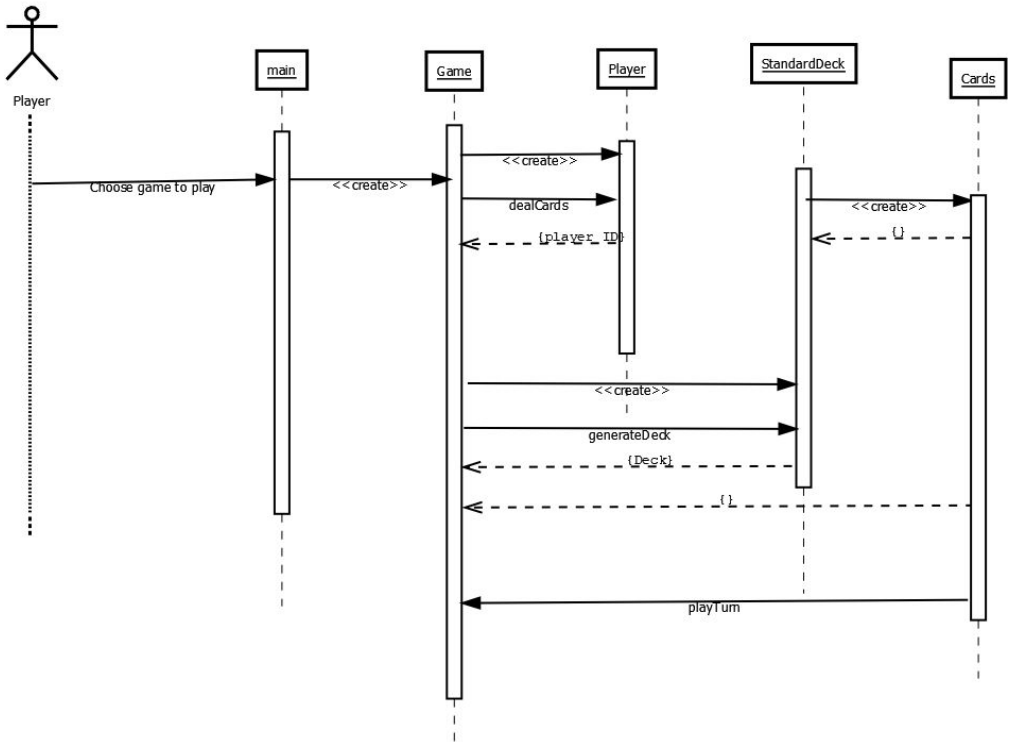

Figure 2: Sequence for one round of Crazy 8's

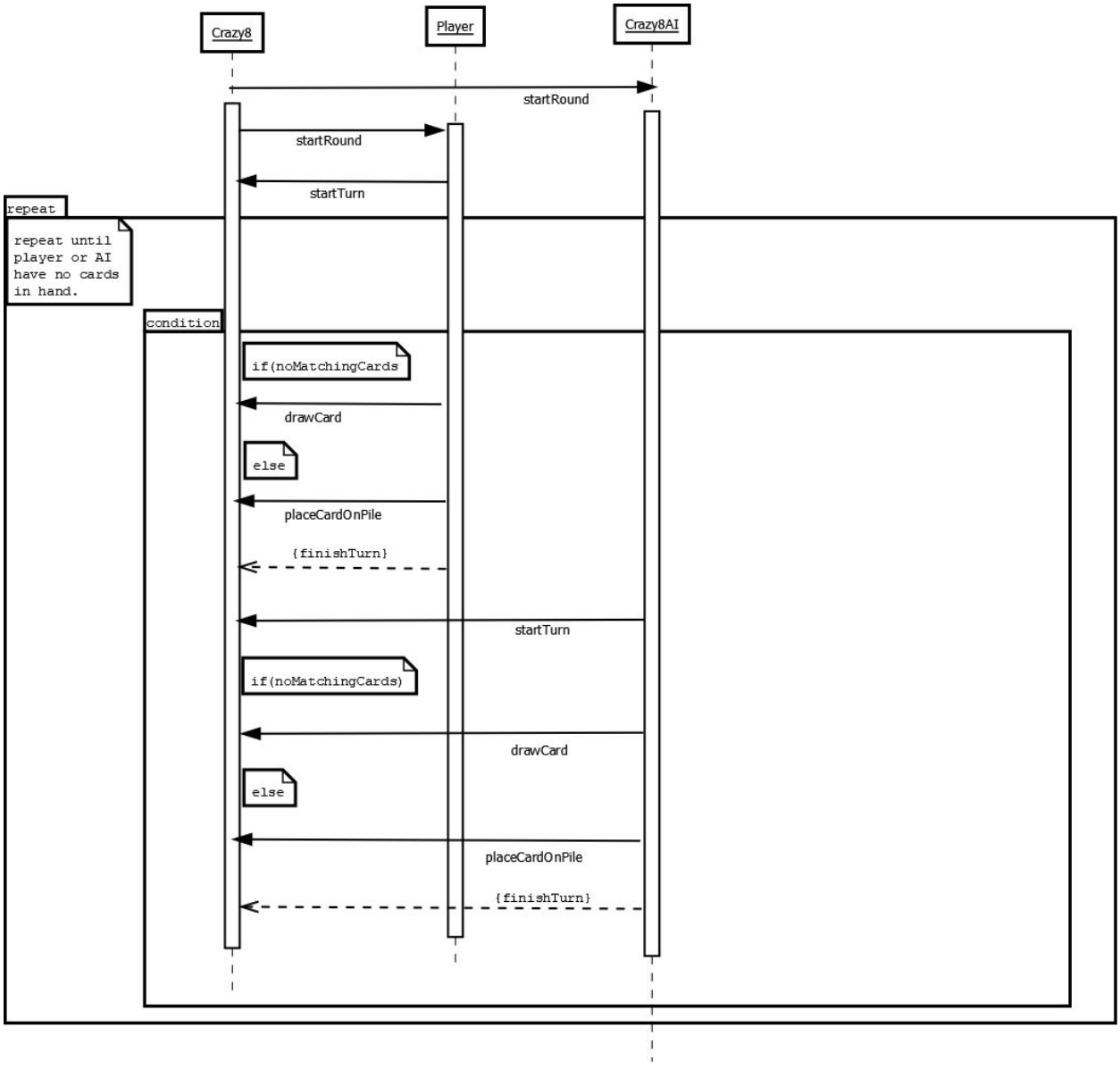
Crazy8
Player
Crazy8AI
startRound
startRound
startTurn
repeat
repeat until player or AI have no cards in hand.
condition
if(noMatchingCards
drawCard
else
placeCardOnPile
{finishTurn}
startTurn
if(noMatchingCards)
drawCard
else
placeCardOnPile
{finishTurn}

Figure 3: Sequence Diagram for one round of Go Fish

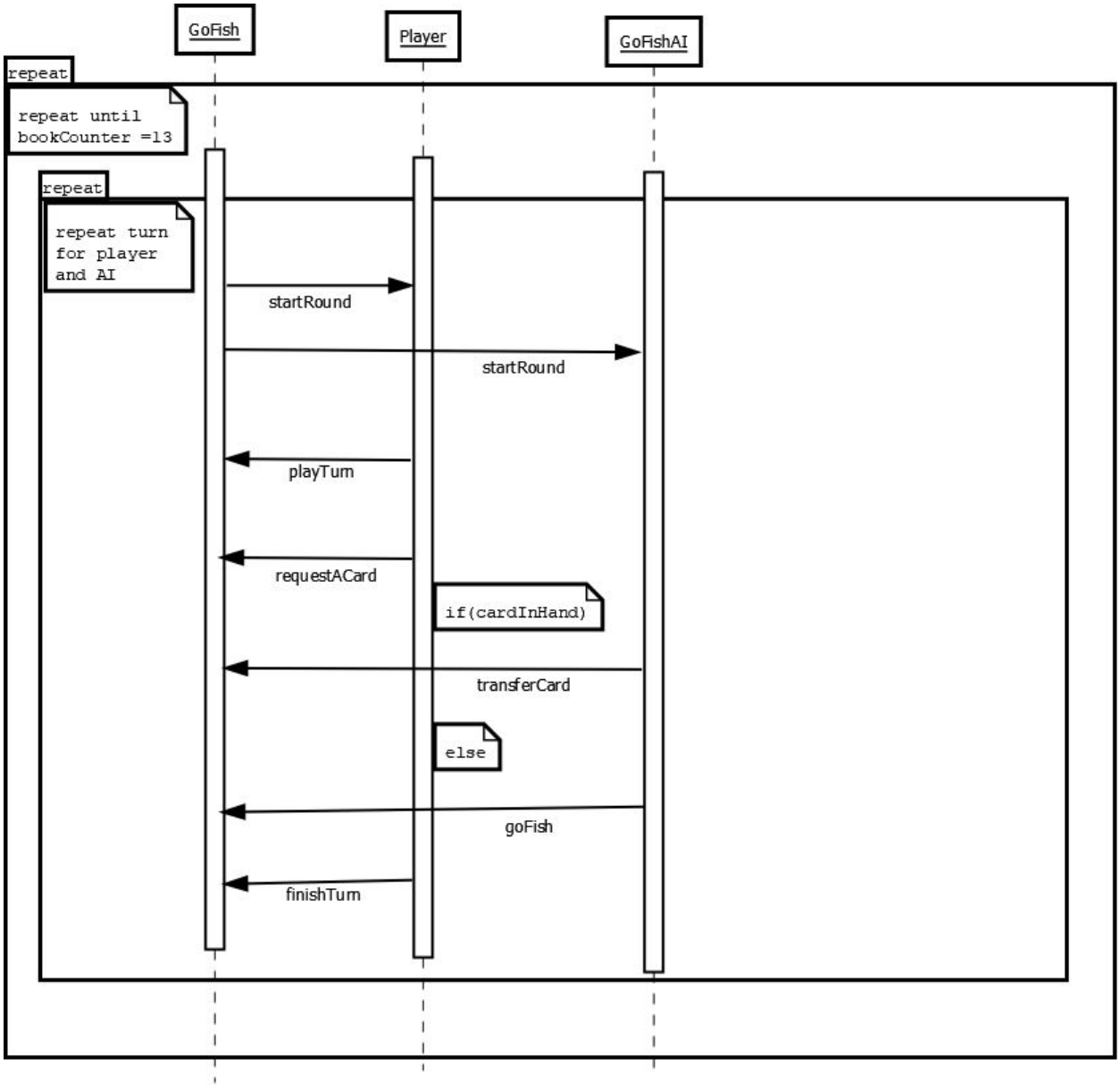

Figure 4: Sequence Diagram for one round of Jungle Speed

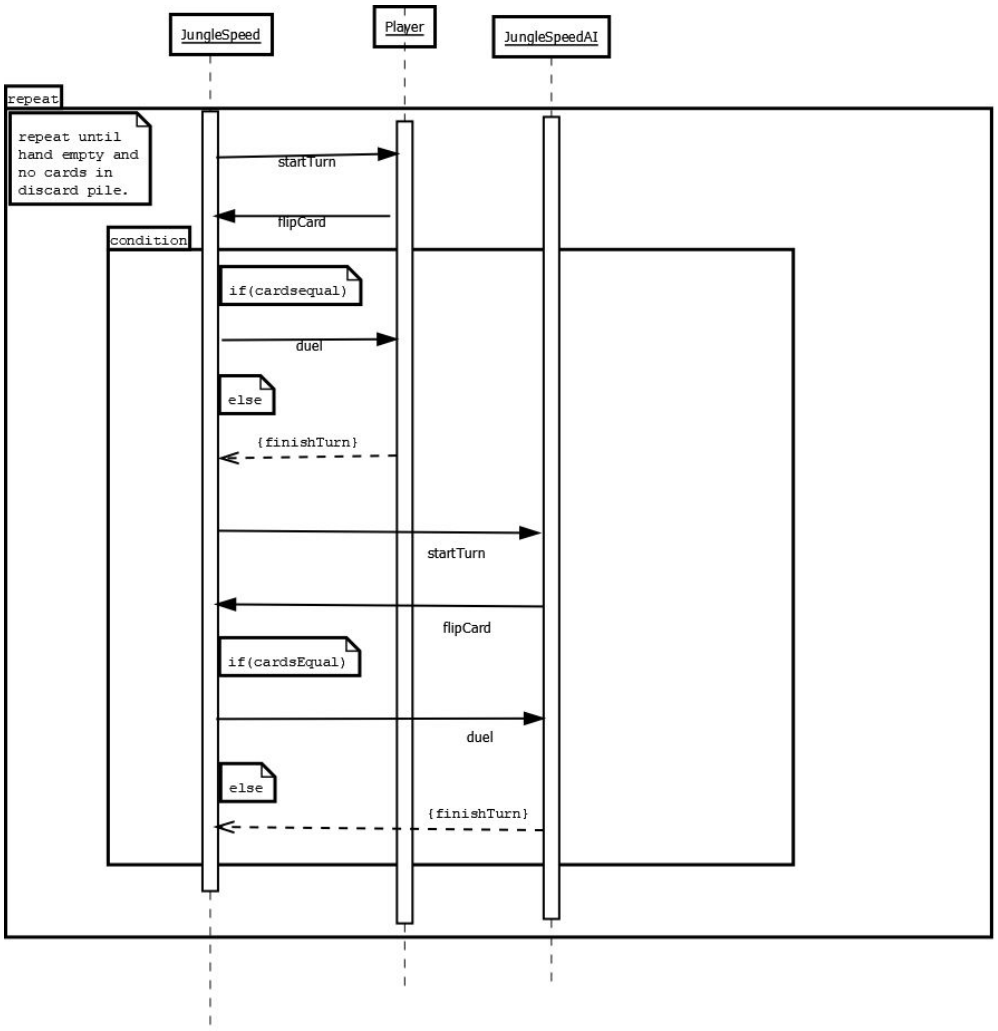

Figure 5: Sequence Diagram for after a game ends

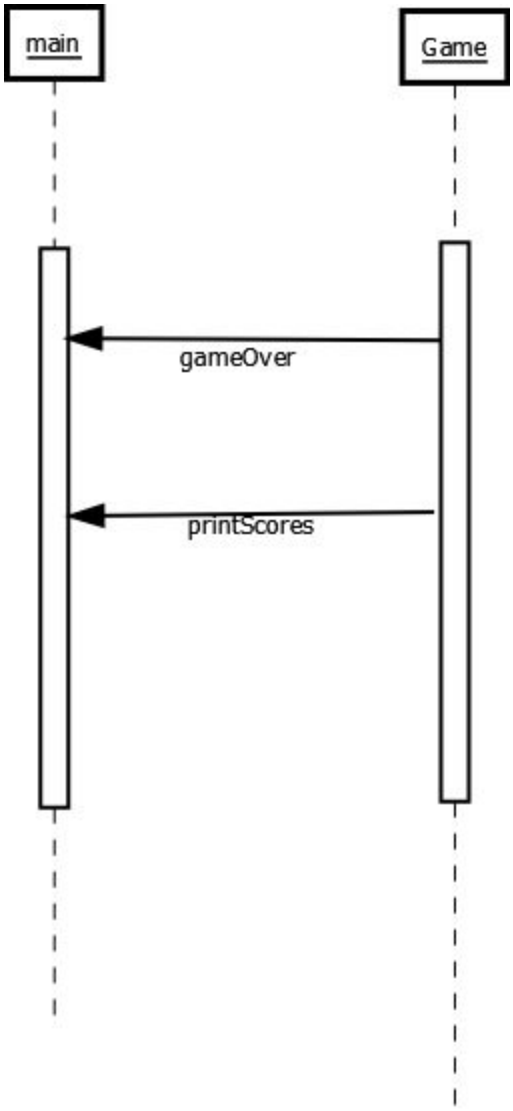

Figure 6: Classs Diagram of General Classes