Final Project

CSE/IT 213

NMT Department of Computer Science and Engineering

"In theory there is no difference between theory and practice. In practice there is."

— Yogi Berra

"A ship in port is safe, but that's not what ships are built for."

— Grace Hopper

"If the automobile had followed the same development as the computer, a Rolls Royce would today cost $100 and get a million miles per gallon, and explode once a year killing everyone inside."

— Robert X. Cringely

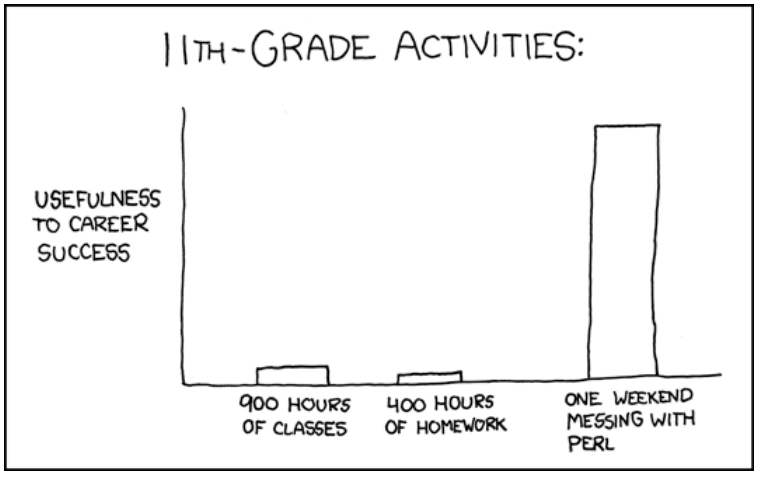

Figure 1: `https://xkcd.com/519/`

Introduction

For the final project, you may choose one (and only one) of two options.

The first option is implementing a Go Fish game. Here you must support a human player and three simple AI players for a total of four players. Optionally you may allow four human players to play over the network for 10 points of extra credit.

The second option is implementing a version of the card game Uno. Here, you must be able to handle turn order changes that sometimes occur in Uno. This option is worth more points, and like Go Fish has the possibility for 10 points of extra credit if you implement network play capabilities.

Projects

Option 1: Go Fish (100 Points)

For this option, you will implement the card game Go Fish. In Go Fish, each player is given seven cards of random values. The players take turns (in a clockwise order) asking a specific player if they have a specific card value in their hand. If that player does have the card, they relinquish it to the player who asked them. If not, the player who is asked for a card says "go fish" and the original player who asked for the card must draw a new card to add to their hand. If a player asks another player for a card and receives it, they may continue asking for cards from either the same player or a different one until they ask a player for a card that said player does not have. When a player has a book, meaning four cards of the same value and different suites (e.g. an ace of hearts, spades, diamonds, and clubs), they set that book aside in a pile. The game ends when all 13 books have been won, and whoever won the most books wins the game. For more details on the rules please see `https://bicyclecards.com/how-to-play/go-fish/`.

The game behavior for your project must be as follows:

- The application must accept 4 players, one human, three AI.
- The application will have a GUI feature to allow the player to begin the game, such as a "play now" button.
- Upon starting the game, each player is delt a hand of seven randomly chosen cards.
- Each player is given a unique name to differentiate them (This can be something simple like player1, player2, etc.)
- The game order will go clockwise in the order starting with the human player.
- Upon a player's turn, they will be able to choose any one of the other players to ask for a card. If they receive that card, their turn repeats.
- If a player asks for a card and the player does not have that card in their hand, the game logic automatically indicates that "go fish" has occurred and the turn is moved to the next player.

- When a player gets a book, the cards that compose it are automatically removed from the player's hand and their book count increases by one.
- The game ends when the combined total of books on the board is 13, after which the player is asked if they would like to play again, and either a new game will begin or the program will terminate if the player indicates they do not want to play again.

The AI you implement will be a simple dummy AI. At minimum, it can simply ask for a random card from the values it holds in its hand from a random player, and the results of this (receiving the cards or needing to go fish) will be handled automatically by the game logic.

Optionally, you may implement network play capability for 10 points of extra credit. The game will be the same as above but modified to accomodate network players as follows:

- The server must accept up to 4 clients.
- The clients will have a GUI feature to inform the server when they are ready to play.
- The game does not begin until at least 2 clients are connected and both have indicated that they are in a ready state.
- Upon starting the game, each player is delt a hand of seven randomly chosen cards.
- Each client is given a unique name to differentiate them (This can be something simple like client1, client2, etc.)
- The game order will go clockwise in the order from the first client who indicated a ready state to the last one
- Upon a player's turn, they will be able to choose any one of the other players to ask for a card. If they receive that card, their turn repeats.
- If a player asks for a card and the player does not have that card in their hand, the server automatically gives a signal to the clients that "go fish" has occurred and the turn is moved to the next player.
- When a player gets a book, the cards that compose it are automatically removed from the player's hand and their book count increases by one.
- The game ends when the combined total of books on the board is 13, after which players are asked if they would like to play again, and either a new game will begin with every player that indicated they wanted to play again or the program will terminate if no more than one player wanted to play again.

There will be a class `Card.java`, which will be an object to store information on a card. It must have an enum with the four suites "HEARTS, CLUBS, DIAMONDS, SPADES", and another enum with the value "ACE, ONE, TWO, ..., TEN, JACK, QUEEN, KING". The constructor for this object must set it's suite and value from the enum values. You must also have a class `Deck.java` which holds an ArrayList<Card> with all the cards in the deck. It must also have a function getCards(int n) which retrieves n cards from the ArrayList, chosen at random, and returns them, deducting those cards from the deck's ArrayList. Remember there will be 52 cards in the deck, one of each value in each suite. For the Client or human player GUI portion you may use as many classes

as you feel necessary. You may design the layout of the GUI yourself, as long as it meets the following requirements:

- All of the components must be laid out in a manner that is easily clear and readable
- You must have a component to show all the cards the player has
- You must have a component to show all the players currently in the game, and allow the player to select one to ask for a card from during their turn
- You must have a component (or set of components) to let the player choose a card to ask for.
- You must have a component to keep track of the number of books each player has.
- You must have a component to indicate when the game is over, and to show which player won.
- You must have a component to allow players to select to replay when the game is over.

Of course, your project must also meet the submission requirements in the submission section as well.

Option 2: Uno (120 Points)

The Uno project is similar to the Go Fish one, except with, of course, Uno. Again in this project you must allow four players to play, one human and three AI. Every player is delt 7 cards (which should be randomly chosen from the deck). One card from the remaining deck should be exposed and put in a separate "discard" pile. In a clockwise order, each player must attempt to play one card that shares a color, number, or symbol. A player could also play one of the following special cards:

- Skip - The next player in the sequence misses a turn
- Reverse - The order of play switches direction (from clockwise to counterclockwise, or vice versa, in a two player game it acts like a skip card)
- Draw Two (+2) - The next player in the sequence draws two cards and misses a turn
- Wild - Whoever plays this can choose the next color to be matched in the discard pile
- Wild Draw Four - This can only be played if the current player doesn't have any cards of the current color in the discard pile. It allows whoever plays it to choose the next color to be matched in the discard pile, and causes the next player to draw four cards and skip their turn.

The player could also draw a card from the top of the deck instead, in which case they must either play the card, or if that is not possible, their turn is forfeit and they must keep the card they drew. There are further rules that you can find here: `https://en.wikipedia.org/wiki/Uno_(card_game)#Official_rules`. Note that you do not need to implement house rules, only the official ones. The game ends when one player earns 500 points. Points are earned by a player when they have no further cards in their hand, and are counted from the values of the cards in other player's hands. Number cards are valued based on their number, action cards count for 20, and wild/wild

draw four cards count for 50 points. You do NOT need to implement rules that require vocal interaction from players, or other more nuanced interactions, such as calling "Uno" or challenging the Wild Draw Four card.

Like Go Fish, you must also have `Deck.java` and `Card.java`, which will behave nearly identically. Here, `Card.java` will have two enums, one for color and one for the symbol/value (i.e. TWO, THREE, REVERSE, etc.). These values will be stored in the Card object through its constructor, like in Go Fish. Let `Deck.java` behave like in Go Fish. Remember that in Uno there are 108 cards, four Wild, four Wild Draw Four, and 25 cards of each color. Each color will have one zero, two each of the numbers 1-9, and two each of Skip, Draw Two, and Reverse. The colors are red, yellow, green, and blue. The AI can also be very simple, where it randomly chooses a valid card to play, or if it has no valid card, draws one from the deck and plays it if possible.

As with the Go Fish project, you can implement network play for 10 points of extra credit. For this, you must have classes for the client networking side, `Client.java` and `ClientThread.java`. You must also have classes for the server networking side, `Server.java` and `ServerThread.java`. You may create other classes around these for handling game logic and components, but the core networking component should be located in these classes.

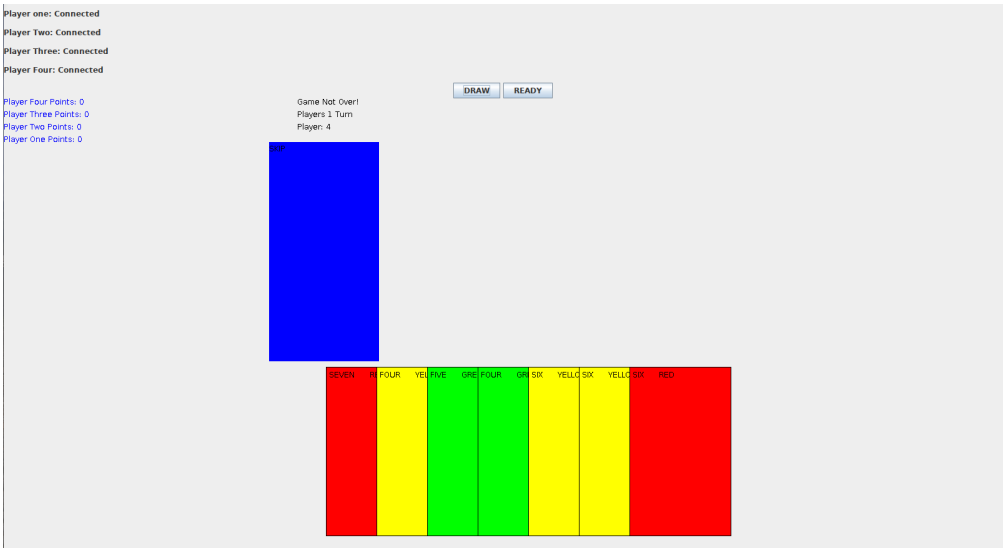

Figure 2: An example of what your project could look like. Please note it can look different, as you are free to create any assets or use any graphics technique you wish, as long as it has all the required components.

Like in the Go Fish project, the game behavior will consist of the human player(s) selecting when they are ready, and choosing if they want to play again at the end of a game. The rules of course will be for Uno rather than Go Fish. You may design the GUI of the game, and use as many classes as you feel necessary, as long as your GUI:

- is clearly organized and readable
- has a component to allow a player to indicate that they are ready
- has a component (or set of components) to show the cards in the player's hand

- has a component to show the card at the top of the discard pile
- has a component to show how many cards each player has
- has a component to allow the player to choose a card to play (and have the game logic (on the server side, if you implement network play) handle whether or not the player may play that card and if so what should happen as a result)
- has a component to indicate the amount of points each player has (which of course must be updated by the server side game logic with network play)
- has a component to indicate when the game is over and who won, and a component to allow the user(s) to opt into a rematch if they desire.

Your project must also meet the submission requirements in the submission section as well.

Submission

Make sure that you have Javadoc style comments for every class and method in your source code, as described in Homework 0. Document any unresolved bugs in the Javadoc comments for each of your classes. You must also include UML diagrams for every class you write! You can include these in one PDF file, be sure to label every class and attribute clearly. If you implement network play, include two JAR files in your submission, one to run the client aspect of your project and one to run the server aspect. Otherwise just include one JAR file to launch the application. When you are satisfied that your code is complete, create a TAR file containing all of the source code for this assignment called:

`cse213_<firstname>_<lastname>_final.tar.gz`

Upload your submission to Canvas before the due date. Please relay any questions about this lab to the head TA for the class. Contact information can be found on Canvas.