# Algorithm-Jitsu

Jack Donnelly, Michael Bishai, Max Pugh

Card-Jitsu

- Club Penguin was one of the first online games we played as kids, so there is some sentimental motivation behind this project.

- The Card-Jitsu game is complex enough to involve strategy but simple enough that we have time to develop our own version and still work on the card choice algorithms.

Beats

Beats

Beats
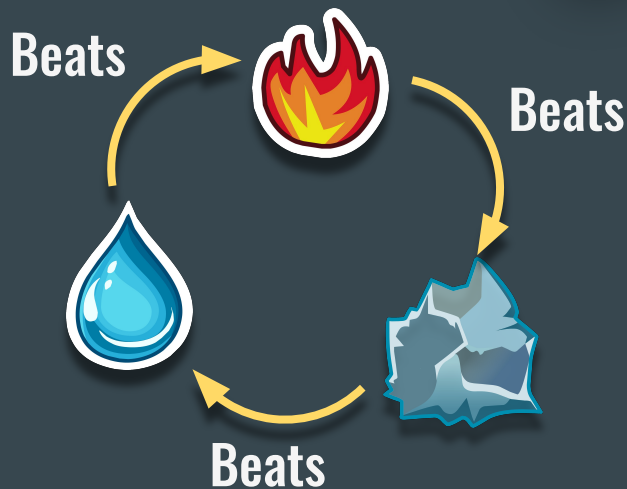
Architecture

- We utilized a Client - Server architecture for its modularity and simplicity.
  - The Client chooses between playing as a person or an easy, medium, or hard AI.
  - The Server waits for two clients to connect, deals cards, and facilitates communication between the two clients as well as the relevant game logic that must be handled.
- The Server is also responsible for logging the games and storing the information in a text file, which we use to develop and test our AI.
- The Client is only responsible for determining which card is picked via user interface or the selected opponent algorithm.
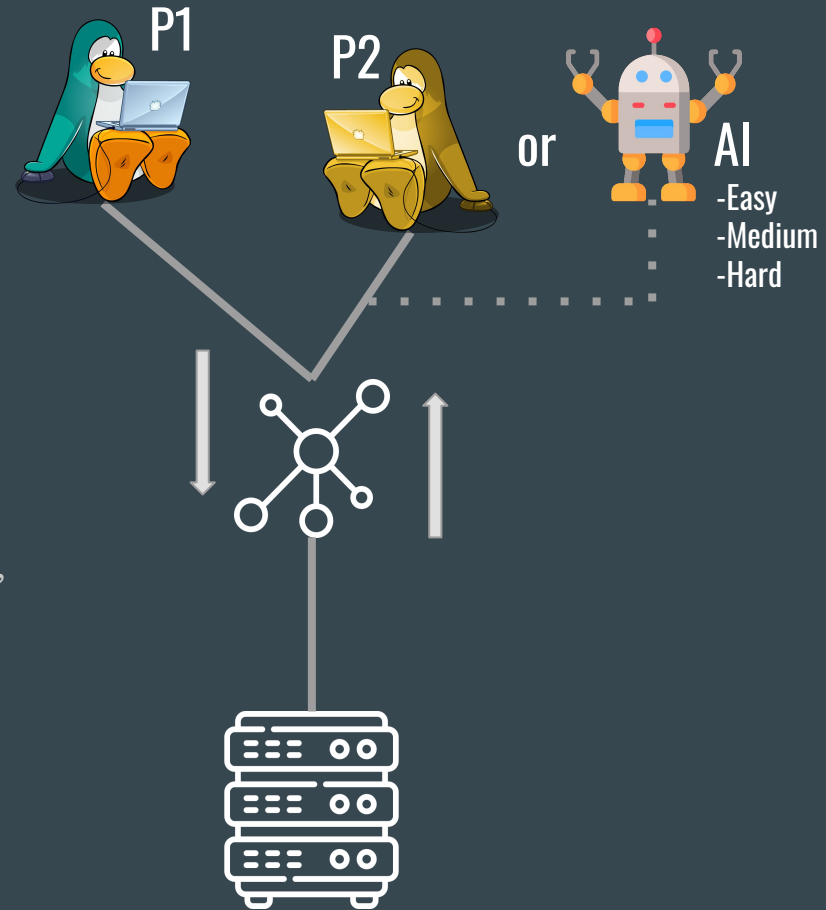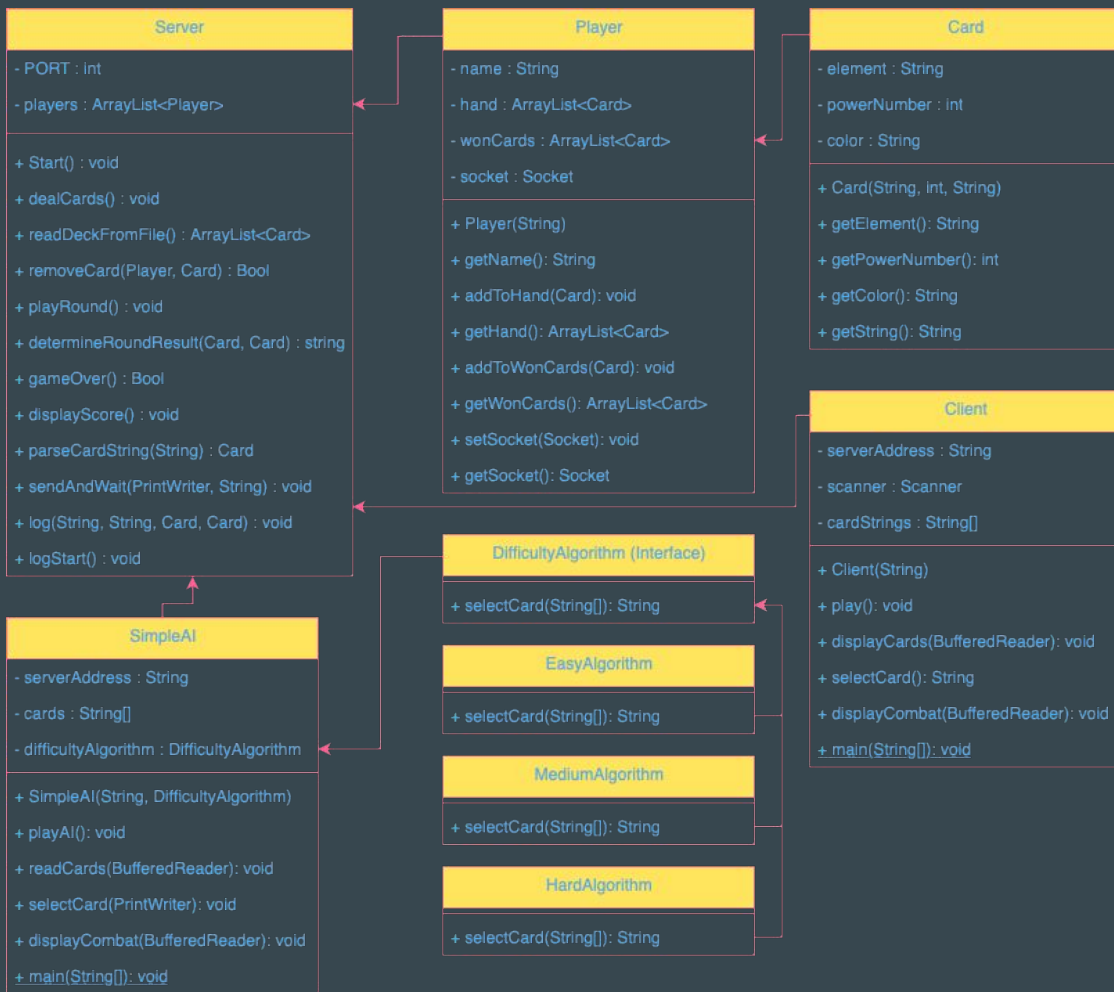
Implementation

- Java was our programming language of choice because of its portability and straightforward socket support.
- The Server opens a socket on a specific port for the clients to connect to, creating each player object and their cards as well as dealing them.
- Each round the server checks for win conditions, sends card information to players, and receives card choices via messages sent over sockets.
  - The first word of the message indicates how it is to be interpreted by the client.
  - The rest of the message is the data that comes along with the message.
- Each Client works similarly, but leaves card choice up to the given algorithm or human opponent.

Class Diagram

Server

- PORT : int
- players : ArrayList<Player>

+ Start() : void
+ dealCards() : void
+ readDeckFromFile() : ArrayList<Card>
+ removeCard(Player, Card) : Bool
+ playRound() : void
+ determineRoundResult(Card, Card) : string
+ gameOver() : Bool
+ displayScore() : void
+ parseCardString(String) : Card
+ sendAndWait(PrintWriter, String) : void
+ log(String, String, Card, Card) : void
+ logStart() : void

Player

- name : String
- hand : ArrayList<Card>
- wonCards : ArrayList<Card>
- socket : Socket

+ Player(String)
+ getName(): String
+ addToHand(Card): void
+ getHand(): ArrayList<Card>
+ addToWonCards(Card): void
+ getWonCards(): ArrayList<Card>
+ setSocket(Socket): void
+ getSocket(): Socket

Card

- element : String
- powerNumber : int
- color : String

+ Card(String, int, String)
+ getElement(): String
+ getPowerNumber(): int
+ getColor(): String
+ getString(): String

Client

- serverAddress : String
- scanner : Scanner
- cardStrings : String[]

+ Client(String)
+ play(): void
+ displayCards(BufferedReader): void
+ selectCard(): String
+ displayCombat(BufferedReader): void
+ main(String[]): void

DifficultyAlgorithm (Interface)

+ selectCard(String[]): String

SimpleAI

- serverAddress : String
- cards : String[]
- difficultyAlgorithm : DifficultyAlgorithm

+ SimpleAI(String, DifficultyAlgorithm)
+ playAI(): void
+ readCards(BufferedReader): void
+ selectCard(PrintWriter): void
+ displayCombat(BufferedReader): void
+ main(String[]): void

EasyAlgorithm

+ selectCard(String[]): String

MediumAlgorithm

+ selectCard(String[]): String

HardAlgorithm

+ selectCard(String[]): String

Data Structures/Algorithms

- Most of the game objects are stored in Java ArrayLists, but we utilize Java HashSets to check win conditions without counting duplicate cards.
- Algorithms play a card for the AI Client which is built very close to human client.
    - EMH: Identifying a card worth playing
    - MH: Considering goals of overarching game
    - H: Identifying losing streaks and playing preventative measures

Easy Algorithm/SimpleAI

SimpleAI.java

- SimpleAI acts as the AI client
- SimpleAI receives a hand from the server class and selects a card to be played each round.
- Getting this to work was a crucial step in development because we could finally focus on the card choice algorithms.

Easy Algorithm

- A random card is selected from SimpleAI's hand
- No special justification nor judgement
- This was implemented to be able to test SimpleAI.java

Medium Algorithm

Offensive Strategy

- Checks the amount of each element it has in hand
- Identifies the element with most quantity
- Play that card type until common element is not in hand
- Rerun Medium Algorithm for new element numbers (repeat)

Hard Algorithm

- Improvement to 'Lack of Variety' in Medium Algorithm
- New Additions
  - Loss Count (understanding if the AI is losing to opponent)
  - selectedCard: scoring of value of cards to help pick the best option for winning
  - useAlternateStrategy: Swap between two strategies focused on game winning
    - Strategy 1: Focuses on getting 3 unique elements + colors
    - Strategy 2: Focuses on getting 3 matching elements w/ unique color
- Compares scores of cards to find best score
- If loss count reaches 2 in a row, swap strategy.

Work Division

Michael

- Core game and server implementation
  - Server, Client, Player classes
- DifficultyAlgorithm.java Interface
- HardAlgorithm.java and MediumAlgorithm.java
- Debugging

Jack

- AI interface implementation
  - SimpleAI.java
  - EasyAlgorithm.java
- Polishing Mechanics
  - Index based card selection
  - Difficulty selection
  - Game logic bug fixes
- Debugging

Max

- Log System implementation
- Polishing Mechanics
  - Cards pulled from deck after playing
  - Update deck state.
- MediumAlgorithm.java, HardAlgorithm.java
- Shell Scripts
- Debugging

Results/Future Direction

- Changing the game to be multiplayer over the internet would be an interesting way to expand this project.
    - Finding a way to connect players peer to peer would also be a very interesting path to go down.
    - (write an algorithm for matchmaking, and menu for offline mode)
- Another way to make the game more appealing would be to implement a graphical user interface.
    - Consider porting the game to javascript and implement HTML5 + CSS for graphics + lightweight
- Servers should be deployed from a main server, with a configuring client deployment automatically
    - In tandem with GUI recommendations, make a website for the game to be freely played on.

Literature Review

references to related work and theoretical background.

Rules: https://clubpenguin.fandom.com/wiki/Card-Jitsu

Original Game: https://en.wikipedia.org/wiki/Club_Penguin

Presentation Images: https://www.clipartmax.com/