Documentation for Canadian Fish

# Introduction

Hi, this is my submission for KPCB Engineering Design Challenge.

Try turning on Navigation Pane in Word for better navigation of this documentation page.

Comments are also included in the code, and variable names are very descriptive.

## Rules

<http://bantha.org/~develin/cardgames.html#ch9>

## Running the program

You have several options:

* You can directly run the .exe included in the folder
* You can copy my .cpp file and compile it in the IDE of your choice
* You can use my visual studio solution file

Or you can watch some of the demo screen recordings!

* /demos/demo1 is with all thinkingPlayers
* /demos/demo1 is with team2 as all cheatingPlayers
* Email me ([deanzhou@stanford.edu](mailto:deanzhou@stanford.edu)) if you need more demos

## Playing the game

Press any when it is on someone else’s turn.

Use the arrow keys and space when it is your turn.

In the main() class, you can change the type and name of players, as well as edit any hyperparameters.

# Structures/Classes

## Player

One of the 6 players of the game, contains various quality-of-life helper functions.

Contains reference to the overall Game class.

Generally a superclass that will be inherited for more specificity.

When developing a subclass of a Player, it must contain

takeTurn() and endgame()

## Game

One overall structure that contains everything inside of it.

Sort of the admin class that contains the location of the cards, and controls whose turn it is.

## RandomPlayer

A computer player who just makes decisions at random.

## CheatingPlayer

A computer player who directly access Game information and always “guesses” correctly.

## HumanPlayer

Structure for user input. Directly uses keypresses and predetermined options, rather than having to sift & QA input.

## ThinkingPlayer

A computer player that uses logic to best determine which player has which cards

## Main

Instantiates the Game, instantiates players.

Adds players to Game and has Game deal the cards.

While the game is not finished, it will keep running through turns.

After the game finishes, it will print the result.

# Terminology/Conventions

We are using a 48-card deck, with 8’s and Jokers removed.

Fish can only be called on a player’s turn.

HalfSuits consist of 6 cards, with the low halfSuits 2-7 of a suit, and the high halfSuits 9-A of a suit.

0 = low Clubs, 1 = high Clubs, 2 = low Diamonds, 2 = high Diamonds, 3 = low Hearts, 4 = high Hearts, 5 = low Spades, 6 = high Spades (alphabetical order of suits)

Game consists of 6 people, 3 on each team.

Game is coded such that Player 1 is the human player and is always on team 1 with player ID 0.

Endgame is the scenario in which one team has exhausted all of their cards, so the other team must (silently and without discussion) declare Fish on the remaining suits. I have it coded that the member on the team with the most cards must give their best guess. In the case that team 1 still has cards, I have it coded that the Player must select who on their team must give their best Guess.

# Algorithms/Hacks

## Game.dealCards()

Creates a vector of cardIDs, shuffles them, and distributes them evenly to the players

## Game.takeTurn()

When a player runs out of cards, the turn automatically passes to the next clockwise teammate

## HumanPlayer.takeTurnUtil()

Essentially a state machine for user input.

Uses recursive stack.

MenuOption controls what part of the menu they are in.

Class variables currentSelection, halfSuitSelection, and askSelection preserve data previously selected by the user.

### While(true) \_getch() loop

Loop that takes in user input. While(true) to wait for input, and manipulates the result of \_getch() to edit class variables when certain keys (left arrow, right arrow, space) are selected

i.e. ln 761-793, but used many many times for user input

## ThinkingPlayer.generateKnowledge()

Uses several logic to best “think”

### vector<vector<int> cardLocationKnowledge;

List of players who possibly could have card i.

Begins full, with every player ID.

If I have the card, then cardLocationKnowledge[cardID] consists of only me.

### vector<vector<int>> halfSuitKnowledge;

Minimum number of cards player i has in halfSuit j

If a player asks for a card, they have at least a card in that halfSuit

### Getting pastEpoch state– ln 1268-1280

Variable pastEpoch is how back in history our memory parameter allows us to look.

We set our cardLocationGuess to what we knew in pastEpoch

If a card doesn’t exist in currentEpoch, we ignore the card

### Gathering Knowledge from Card Asks

If a cardAsk is successful, we know that cardLocationKnowledge[cardID] only consists of the asker (they have the card now)

The asker’s cardLocationKnowledge in that halfSuit increments

The askee’s cardLocationKnowledge in that halfSuit deincrements

If a cardAsk is unsuccessful, we know that cardLocationKnowledge[cardID] has neither the asker nor the askee.

The asker’s cardLocationKnowledge is at least 1.

### Gathering Knowledge from Fish Declares

We purposely ignored cards that don’t exist in currentEpoch, so we ignore Fish Declares

### Gathering Knowledge from cardLocationKnowledge

If some subset of players’ cardLocationKnowledge adds up to 6 (say Alice, Bob, and Xavier have 2 each), then we know that the other players (Player, Yadier, Zachary) cannot have any of that halfsuit.

Therefore, we can remove Player, Yadier, and Zachary from cardLocationKnowledge for every card in that halfSuit.

This actually gains us a lot of information.

# Other Methods/Variables

Most of the other methods are self-documenting with descriptive names.

If you really don’t understand a method, feel free to email me at deanzhou@stanford.edu

# Notes/Possible Extensions

This entire project was really a quick hack for my own enjoyment and I don’t expect anyone to use it seriously, and I don’t expect myself to develop it further.

That being said, I tried to code without hard numbers, setting global const variables such as SUIT\_MAX, NUMBER\_MAX, and NUM\_PLAYERS. Code could be quickly edited (in theory) to support play with a different deck of cards of a different amount of players.

However, currently, it supports 6 players with the human player being player 1.

Here are the reasons (that you must change if you want to change the setup):

* Game prints the hand of player 1 every turn.
* Endgame gives the option for player 1 to choose who declares Fish

However, you can very easily change the computer players (2-6). You can do the following:

* Change the type of player (RandomPlayer, CheatingPlayer, ThinkingPlayer)
* Change the memory parameter of ThinkingPlayer (right now set to 1000, functionally infinity)
* Create your own type of player (just extend Player and implement takeTurn() and endGame()

Also try playing this game with your friends! It’s much more fun with people ☺

Thanks for reading! – Dean Zhou