CS106L Handout #15
Fall 2007 October 31, 2007

Hearts Contest

Due November 14, 11:59 PM

Introduction

By popular demand, we'll be holding an AI hearts contest! While the tournament will not count for a grade, it will still give you a chance to combine your C++ skills with your AI savvy to build a pretty cool piece of software.

For this project, you will not be implementing the entire hearts program. Instead, you'll be writing a class called `HeartsPlayer` that exports methods that interact with a central hearts program I'll provide. Your goal is to fill out the class methods with whatever code you see fit, as long as it's all standard C++! If you want, you can even use a modified version of the lecture code.

Once I've received all the submissions, I'll wire them all together into a tournament program that will pit all the AIs against one another in a multi-round tournament. Unlike the hearts player from lecture, I'll use some nonstandard C++ to animate the cards so we can watch the AIs duke it out! At the end of the tournament, if you submitted the winning entry, you'll be crowned the CS106L Card Shark and will receive a $20 iTunes gift card for your valorous effort.

Rules for Hearts

Because there are many variants on the rules of hearts, for the hearts contest we will treat the following list of rules as the "official" rules for hearts:

1. At the start of each round, each player is dealt thirteen random cards. Players do not pass cards – whatever cards you're dealt are the cards you have to work with.
2. At the start of the round, the player with the two of clubs goes first. The very first card played must be the two of clubs. Thus, if you have the two of clubs and it's the first round, you must play it.
3. You can play "point cards" (hearts and the queen of spades) on any trick, including the first.
4. If it's your turn to lead, you cannot play a heart unless you have no other choice or someone has played a heart on a previous trick (usually because that player didn't have any of the led suit)
5. When it's your turn to follow suit, you must play a card of the proper suit if you have one. Otherwise you can play any card from your hand.
6. At the end of each round, players total up the points they've accumulated in the round. The queen of spades is worth 13 points, and each heart is worth 1 point.
7. A player who takes all twenty-six points in a round "shoots the moon" and takes zero points. Each other player then takes twenty-six points instead. Thus if you manage to take all of the point cards, you put all other players at a serious disadvantage.
8. The game ends whenever a player gets 100 or more points. Players don't reset to zero points if they have exactly 100 points.

Please make sure you understand these rules before writing the player! I have seen many variations on the rules of hearts and hope everyone is on the same page for the tournament.

Deliverables

For the purposes of the contest `suitT`, `valueT`, `cardT`, and `deckT` are defined as:

```
enum suitT {Hearts, Diamonds, Clubs, Spades};
enum valueT{Two, Three, Four, Five, Six, Seven,
            Eight, Nine, Ten, Jack, Queen, King, Ace};

typedef pair<suitT, valueT> cardT;
typedef deque<cardT> deckT;
```

Note that while in lecture `cardT` was its own `struct`, here **`cardT` is simply a `typedef` for a `pair`**. This allows you a bit more flexibility with the type, since you can store `pair`s in a `set`, test them for equality, or even cram them in a `map`.

Your submission will be an implementation of the `HeartsPlayer` object, which is defined below.

```
class HeartsPlayer
{
    public:
        explicit HeartsPlayer(int index); // See footnote
        ~HeartsPlayer();

        void setHand(const deckT &cards);

        cardT chooseFirstCard();
        cardT chooseFollowingCard(const deckT &deck);

        void endTrick(const deckT &playedCards);
        void endRound(const deckT &scores);
};
```

The member functions are discussed in more detail below:

Constructor: When your hearts player is created, it will be stored in an array containing the other computer players. This array is guaranteed not to change throughout the course of the game (unlike the example in lecture, we won't `rotate` the players around), so it's possible to refer to each player by their index in the array. When your player is constructed, it will receive as a parameter to the constructor its index in the players array. You should store this number since later functions will assume you know what position you're in. You can assume this number is between 0 and 3, inclusive.

`void setHand(const deckT &)`: The computer will call the `setHand` function on your `HeartsPlayer` object at the start of each round, passing in a `deckT` object representing your hand. The cards will be provided in random order, so do not expect that they will be presorted. You can assume that your hand will contain thirteen unique cards.

`cardT chooseFirstCard()`: This function is called when it's your turn to lead a card at the start of a trick. The function should return which card you intend to play and should remove that card from your hand.

We'll cover the `explicit` keyword next week. For now you can ignore it.

`cardT chooseFollowingCard(const deckT &)`: This function will be called when it's your turn to play a card that must follow suit. The parameter is a `deckT` representing the cards that have already been played this round with the first card in the `deckT` representing the first card played this trick. This function should return which card you're playing and should also remove that card from your hand.

`void endTrick(const deckT &)`: Once all four players have played their cards, the program will call the `endTrick` function on your `HeartsPlayer` and provide what cards were played this round. These cards will be provided to you in the order that players are stored in the master player index, which means that your card will always be at the index specified in the constructor.

`void endRound(const vector<int> &)`: Once all thirteen cards have been played, the computer will call your `endRound` function and provide a `vector<int>` representing each player's total score at the end of the round. As with `endRound`, the numbers will be provided based on the order of the players.

You are free to *add* to this interface, but you must **not** modify the public interface provided to you.

Note that the `HeartsPlayer` object will only be created <u>once</u> per game of hearts, not once per thirteen-card round. This means that your `HeartsPlayer` must be able to play several consecutive games, provided that it is reset by a call to `setHand` at the start of each game. Your `HeartsPlayer` class will be destroyed when opponents change, which means that each `HeartsPlayer` object can assume it is playing against the same opponents between rounds.

Good luck, and have fun!