MelvinLecoy / gitcode Private

Code Issues Pull requests Actions Projects Security Insights Settings

master

gitcode / p3-euker copy / Player.cpp

Kwan Ting Lau gitcode History

0 contributors

321 lines (281 sloc) | 8.99 KB

```
// Project UID 1d9f47bfc76643019cfbf037641defe1

#include "Player.h"
#include "Card.h"

#include <algorithm>
#include <cassert>
#include <iostream>
#include <ratio>
#include <string>
#include <vector>

using namespace std;

std::ostream &operator<<(ostream &os, const Player &p) {
  os << p.get_name();
  return os;
}

// START HELPER FUNCS----------------------------------------

int get_card_index_in_hand(Card card, vector<Card> &HAND) {
  int card_index = 0;

  for (int i = 0; i < int(HAND.size()); i++) {
    if (card == HAND[i]) {
      return card_index;
    }
  }

  assert(false);
  return 0;
}

void remove_card_from_hand(Card card, vector<Card> &HAND) {
```

```
  for (int i = 0; i < int(HAND.size()); i++) {
    if (card == HAND[i]) {
      HAND.erase(HAND.begin() + i);
      return;
    }
  }
}

// TODO: IMPLEMENT
vector<Card> filter_out_trumps(string trump_suit, vector<Card> HAND) {
  vector<Card> filtered_vector = {};
  for (int i = 0; i < int(HAND.size()); i++) {
    if (HAND[i].get_suit(trump_suit) != trump_suit) {
      filtered_vector.push_back(HAND[i]);
    }
  }
  return filtered_vector;
}

Card get_highest_card(vector<Card> v, string TRUMP, Card LED) {
  int highest_card_index = 0;
  for (int i = 0; i < int(v.size()); i++) {
    if (Card_less(v[highest_card_index], v[i], LED, TRUMP)) {
      highest_card_index = i;
    }
  }
  return v[highest_card_index];
}

Card get_lowest_card(vector<Card> v, string TRUMP) {
  int lowest_card_index = 0;
  for (int i = 0; i < int(v.size()); i++) {
    if (Card_less(v[i], v[lowest_card_index], TRUMP)) {
      lowest_card_index = i;
    }
  }
  return v[lowest_card_index];
}

// END HELPER FUNCS-------------------------------------------

class SimplePlayer : public Player {
private:
  string player = "";
  vector<Card> hand = {};

public:
  // constructor
  SimplePlayer() {}

  // constructor override
  SimplePlayer(string name) { player = name; }

  // EFFECTS returns player's name
```

```
  const std::string &get_name() const { return player; }

  // REQUIRES player has less than MAX_HAND_SIZE cards
  // EFFECTS  adds Card c to Player's hand
  void add_card(const Card &c) { hand.push_back(c); }

  // REQUIRES round is 1 or 2
  // MODIFIES order_up_suit
  // EFFECTS If Player wishes to order up a trump suit then return true and
  //   change order_up_suit to desired suit.  If Player wishes to pass, then do
  //   not modify order_up_suit and return false.
  bool make_trump(const Card &upcard, bool is_dealer, int round,
                  std::string &order_up_suit) const {

    if (round == 1) {
      int x = 0;
      for (int i = 0; i < int(hand.size()); i++) {
        if (hand[i].is_face() &&
            hand[i].get_suit(upcard.get_suit()) == upcard.get_suit()) {
          x++;
        }
      }
      if (x >= 2) {
        order_up_suit = upcard.get_suit();
        return true;
      } else {
        return false;
      }

    } else if (round == 2) {
      int x = 0;
      for (int i = 0; i < int(hand.size()); i++) {
        if (hand[i].is_face() &&
            hand[i].get_suit(Suit_next(upcard.get_suit())) ==
                Suit_next(upcard.get_suit())) {
          x++;
        }
      }

      if (is_dealer || x >= 1) {
        order_up_suit = Suit_next(upcard.get_suit());
        return true;
      }
    }
    // assert(false);
    return false;
  }

  // REQUIRES Player has at least one card
  // EFFECTS  Player adds one card to hand and removes one card from hand.
  void add_and_discard(const Card &upcard) {
    hand.push_back(upcard);
    int smaller = 0;
    for (int i = 0; i < int(hand.size()); i++) {
```

```
      if (Card_less(hand[i], hand[smaller], upcard.get_suit()))
        smaller = i;
    }
    hand.erase(hand.begin() + smaller);
  }

  // REQUIRES Player has at least one card, trump is a valid suit
  // EFFECTS  Leads one Card from Player's hand according to their strategy
  //   "Lead" means to play the first Card in a trick.  The card
  //   is removed the player's hand.
  Card lead_card(const std::string &trump) {
    vector<Card> non_trump_cards = {};
    Card highest_card = hand[0];
    non_trump_cards = filter_out_trumps(trump, hand);

    if (non_trump_cards.size()) {
      Card temp = non_trump_cards[0];
      for (int i = 0; i < (int)non_trump_cards.size(); i++) {
        if (!Card_less(non_trump_cards[i], temp, trump)) {
          temp = non_trump_cards[i];
        }
      }
      highest_card = temp;
    } else {
      Card temp = hand[0];
      for (int i = 0; i < (int)hand.size(); i++) {
        if (hand[i].is_trump(trump) && !Card_less(hand[i], temp, trump)) {
          temp = hand[i];
        }
      }
      highest_card = temp;
    }
    remove_card_from_hand(highest_card, hand);
    return highest_card;
  }

  // REQUIRES Player has at least one card, trump is a valid suit
  // EFFECTS  Plays one Card from Player's hand according to their strategy.
  //   The card is removed from the player's hand.
  Card play_card(const Card &led_card, const std::string &trump) {
    Card card_to_play;
    vector<Card> led_cards = {};
    vector<Card> non_led_cards = {};

    for (int i = 0; i < int(hand.size()); i++) {
      if (hand[i].get_suit(trump) == led_card.get_suit(trump)) {
        led_cards.push_back(hand[i]);
      } else {
        non_led_cards.push_back(hand[i]);
      }
    }

    if (int(led_cards.size()) >= 1) {
      card_to_play = get_highest_card(led_cards, trump, led_card);
```

```
    } else {
      card_to_play = get_lowest_card(non_led_cards, trump);
    }

    remove_card_from_hand(card_to_play, hand);
    return card_to_play;
  }
};

class HumanPlayer : public Player {
private:
  string player = "";
  vector<Card> hand = {};

public:
  // constructor
  HumanPlayer() {}

  // constructor override
  HumanPlayer(string name) { player = name; }

  // EFFECTS returns player's name
  const std::string &get_name() const override { return player; }

  // REQUIRES player has less than MAX_HAND_SIZE cards
  // EFFECTS  adds Card c to Player's hand
  void add_card(const Card &c) override {
    hand.push_back(c);
    sort(hand.begin(), hand.end());
  }

  void print_hand() const {
    for (int i = 0; i < int(hand.size()); i++) {
      // print in ascending order
      cout << "Human player " << player << "'s hand: [" << i << "] "
           << hand[i].get_rank() << " of " << hand[i].get_suit() << "\n";
    }
  }

  // REQUIRES round is 1 or 2
  // MODIFIES order_up_suit
  // EFFECTS If Player wishes to order up a trump suit then return true and
  //   change order_up_suit to desired suit.  If Player wishes to pass, then do
  //   not modify order_up_suit and return false.
  bool make_trump(const Card &upcard, bool is_dealer, int round,
                  std::string &order_up_suit) const override {
    string input = "";
    print_hand();
    cout << "Human player " << player
         << ", please enter a suit, or \"pass\":\n";
    cin >> input;
    if (input == "pass") {
      // cout << player << " passes\n";
      return false;
```

```
    } else if (input == "Spades" || input == "Clubs" || input == "Hearts" ||
               input == "Diamonds") {
      order_up_suit = input;
      // cout << player << " orders up " << input << "\n";
      return true;
    }
    return false;
  }

  // REQUIRES Player has at least one card
  // EFFECTS  Player adds one card to hand and removes one card from hand.
  void add_and_discard(const Card &upcard) override {
    int input = 0;
    // hand.push_back(upcard);
    print_hand();
    cout << "Discard upcard: [-1]\n";
    cout << "Human player " << player << ", please select a card to discard:";
    // hand.erase(hand.begin() + input);
    cin >> input;
    if (input == -1) {
      return;
    } else {
      hand.erase(hand.begin() + input);
      hand.push_back(upcard);
    }
    sort(hand.begin(), hand.end());
  }

  // REQUIRES Player has at least one card, trump is a valid suit
  // EFFECTS  Leads one Card from Player's hand according to their strategy
  //   "Lead" means to play the first Card in a trick.  The card
  //   is removed the player's hand.
  Card lead_card(const std::string &trump) override {
    int input = 0;
    print_hand();
    cout << "Human player " << player << ", please select a card:";
    cin >> input;
    cout << "\n";
    Card card = hand[input];
    hand.erase(hand.begin() + input);
    sort(hand.begin(), hand.end());
    return card;
  }

  // REQUIRES Player has at least one card, trump is a valid suit
  // EFFECTS  Plays one Card from Player's hand according to their strategy.
  //   The card is removed from the player's hand.
  Card play_card(const Card &led_card, const std::string &trump) override {
    int input = 0;
    print_hand();
    cout << "Human player " << player << ", please select a card:";
    cin >> input;
    cout << "\n";
    Card card = hand[input];
```

```
    hand.erase(hand.begin() + input);
    sort(hand.begin(), hand.end());
    return card;
  }
};

Player *Player_factory(const std::string &name, const std::string &strategy) {
  if (strategy == "Simple") {
    return new SimplePlayer(name);
  }
  if (strategy == "Human") {
    return new HumanPlayer(name);
  }
  assert(false);
  return nullptr;
}
```

Give feedback