


 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

...

```
1 // Project UID 1d9f47bfc76643019cfbf037641defe1
2
3 #include "Player.h"
4 #include "Card.h"
5
6 #include <algorithm>
7 #include <cassert>
8 #include <iostream>
9 #include <ratio>
10 #include <string>
11 #include <vector>
12
13 using namespace std;
14
15 std::ostream &operator<<(ostream &os, const Player &p) {
16     os << p.get_name();
17     return os;
18 }
19
20 // START HELPER FUNCS-----
21
22 int get_card_index_in_hand(Card card, vector<Card> &HAND) {
23     int card_index = 0;
24
25     for (int i = 0; i < int(HAND.size()); i++) {
26         if (card == HAND[i]) {
27             return card_index;
28         }
29     }
30
31     assert(false);
32     return 0;
33 }
34
35 void remove_card_from_hand(Card card, vector<Card> &HAND) {
```

```
36     for (int i = 0; i < int(HAND.size()); i++) {
37         if (card == HAND[i]) {
38             HAND.erase(HAND.begin() + i);
39             return;
40         }
41     }
42 }
43
44 // TODO: IMPLEMENT
45 vector<Card> filter_out_trumps(string trump_suit, vector<Card> HAND) {
46     vector<Card> filtered_vector = {};
47     for (int i = 0; i < int(HAND.size()); i++) {
48         if (HAND[i].get_suit(trump_suit) != trump_suit) {
49             filtered_vector.push_back(HAND[i]);
50         }
51     }
52     return filtered_vector;
53 }
54
55 Card get_highest_card(vector<Card> v, string TRUMP, Card LED) {
56     int highest_card_index = 0;
57     for (int i = 0; i < int(v.size()); i++) {
58         if (Card_less(v[highest_card_index], v[i], LED, TRUMP)) {
59             highest_card_index = i;
60         }
61     }
62     return v[highest_card_index];
63 }
64
65 Card get_lowest_card(vector<Card> v, string TRUMP) {
66     int lowest_card_index = 0;
67     for (int i = 0; i < int(v.size()); i++) {
68         if (Card_less(v[i], v[lowest_card_index], TRUMP)) {
69             lowest_card_index = i;
70         }
71     }
72     return v[lowest_card_index];
73 }
74
75 // END HELPER FUNCS-----
76
77 class SimplePlayer : public Player {
78 private:
79     string player = "";
80     vector<Card> hand = {};
81
82 public:
83     // constructor
84     SimplePlayer() {}
85
86     // constructor override
87     SimplePlayer(string name) { player = name; }
88
89     // EFFECTS returns player's name
```

```
90     const std::string &get_name() const { return player; }
91
92     // REQUIRES player has less than MAX_HAND_SIZE cards
93     // EFFECTS adds Card c to Player's hand
94     void add_card(const Card &c) { hand.push_back(c); }
95
96     // REQUIRES round is 1 or 2
97     // MODIFIES order_up_suit
98     // EFFECTS If Player wishes to order up a trump suit then return true and
99     // change order_up_suit to desired suit. If Player wishes to pass, then do
100    // not modify order_up_suit and return false.
101    bool make_trump(const Card &upcard, bool is_dealer, int round,
102                   std::string &order_up_suit) const {
103
104        if (round == 1) {
105            int x = 0;
106            for (int i = 0; i < int(hand.size()); i++) {
107                if (hand[i].is_face() &&
108                    hand[i].get_suit(upcard.get_suit()) == upcard.get_suit()) {
109                    x++;
110                }
111            }
112            if (x >= 2) {
113                order_up_suit = upcard.get_suit();
114                return true;
115            } else {
116                return false;
117            }
118
119        } else if (round == 2) {
120            int x = 0;
121            for (int i = 0; i < int(hand.size()); i++) {
122                if (hand[i].is_face() &&
123                    hand[i].get_suit(Suit_next(upcard.get_suit())) ==
124                    Suit_next(upcard.get_suit())) {
125                    x++;
126                }
127            }
128
129            if (is_dealer || x >= 1) {
130                order_up_suit = Suit_next(upcard.get_suit());
131                return true;
132            }
133        }
134        // assert(false);
135        return false;
136    }
137
138    // REQUIRES Player has at least one card
139    // EFFECTS Player adds one card to hand and removes one card from hand.
140    void add_and_discard(const Card &upcard) {
141        hand.push_back(upcard);
142        int smaller = 0;
143        for (int i = 0; i < int(hand.size()); i++) {
```

```
144     if (Card_less(hand[i], hand[smaller], upcard.get_suit()))
145         smaller = i;
146     }
147     hand.erase(hand.begin() + smaller);
148 }
149
150 // REQUIRES Player has at least one card, trump is a valid suit
151 // EFFECTS Leads one Card from Player's hand according to their strategy
152 // "Lead" means to play the first Card in a trick. The card
153 // is removed the player's hand.
154 Card lead_card(const std::string &trump) {
155     vector<Card> non_trump_cards = {};
156     Card highest_card = hand[0];
157     non_trump_cards = filter_out_trumps(trump, hand);
158
159     if (non_trump_cards.size()) {
160         Card temp = non_trump_cards[0];
161         for (int i = 0; i < (int)non_trump_cards.size(); i++) {
162             if (!Card_less(non_trump_cards[i], temp, trump)) {
163                 temp = non_trump_cards[i];
164             }
165         }
166         highest_card = temp;
167     } else {
168         Card temp = hand[0];
169         for (int i = 0; i < (int)hand.size(); i++) {
170             if (hand[i].is_trump(trump) && !Card_less(hand[i], temp, trump)) {
171                 temp = hand[i];
172             }
173         }
174         highest_card = temp;
175     }
176     remove_card_from_hand(highest_card, hand);
177     return highest_card;
178 }
179
180 // REQUIRES Player has at least one card, trump is a valid suit
181 // EFFECTS Plays one Card from Player's hand according to their strategy.
182 // The card is removed from the player's hand.
183 Card play_card(const Card &led_card, const std::string &trump) {
184     Card card_to_play;
185     vector<Card> led_cards = {};
186     vector<Card> non_led_cards = {};
187
188     for (int i = 0; i < int(hand.size()); i++) {
189         if (hand[i].get_suit(trump) == led_card.get_suit(trump)) {
190             led_cards.push_back(hand[i]);
191         } else {
192             non_led_cards.push_back(hand[i]);
193         }
194     }
195
196     if (int(led_cards.size()) >= 1) {
197         card_to_play = get_highest_card(led_cards, trump, led_card);
```

```
198     } else {
199         card_to_play = get_lowest_card(non_led_cards, trump);
200     }
201
202     remove_card_from_hand(card_to_play, hand);
203     return card_to_play;
204 }
205 };
206
207 class HumanPlayer : public Player {
208 private:
209     string player = "";
210     vector<Card> hand = {};
211
212 public:
213     // constructor
214     HumanPlayer() {}
215
216     // constructor override
217     HumanPlayer(string name) { player = name; }
218
219     // EFFECTS returns player's name
220     const std::string &get_name() const override { return player; }
221
222     // REQUIRES player has less than MAX_HAND_SIZE cards
223     // EFFECTS adds Card c to Player's hand
224     void add_card(const Card &c) override {
225         hand.push_back(c);
226         sort(hand.begin(), hand.end());
227     }
228
229     void print_hand() const {
230         for (int i = 0; i < int(hand.size()); i++) {
231             // print in ascending order
232             cout << "Human player " << player << "'s hand: [" << i << " ] "
233                  << hand[i].get_rank() << " of " << hand[i].get_suit() << "\n";
234         }
235     }
236
237     // REQUIRES round is 1 or 2
238     // MODIFIES order_up_suit
239     // EFFECTS If Player wishes to order up a trump suit then return true and
240     // change order_up_suit to desired suit. If Player wishes to pass, then do
241     // not modify order_up_suit and return false.
242     bool make_trump(const Card &upcard, bool is_dealer, int round,
243                    std::string &order_up_suit) const override {
244         string input = "";
245         print_hand();
246         cout << "Human player " << player
247              << ", please enter a suit, or \"pass\":\n";
248         cin >> input;
249         if (input == "pass") {
250             // cout << player << " passes\n";
251             return false;
```

```
252     } else if (input == "Spades" || input == "Clubs" || input == "Hearts" ||
253               input == "Diamonds") {
254         order_up_suit = input;
255         // cout << player << " orders up " << input << "\n";
256         return true;
257     }
258     return false;
259 }
260
261 // REQUIRES Player has at least one card
262 // EFFECTS Player adds one card to hand and removes one card from hand.
263 void add_and_discard(const Card &upcard) override {
264     int input = 0;
265     // hand.push_back(upcard);
266     print_hand();
267     cout << "Discard upcard: [-1]\n";
268     cout << "Human player " << player << ", please select a card to discard:";
269     // hand.erase(hand.begin() + input);
270     cin >> input;
271     if (input == -1) {
272         return;
273     } else {
274         hand.erase(hand.begin() + input);
275         hand.push_back(upcard);
276     }
277     sort(hand.begin(), hand.end());
278 }
279
280 // REQUIRES Player has at least one card, trump is a valid suit
281 // EFFECTS Leads one Card from Player's hand according to their strategy
282 // "Lead" means to play the first Card in a trick. The card
283 // is removed the player's hand.
284 Card lead_card(const std::string &trump) override {
285     int input = 0;
286     print_hand();
287     cout << "Human player " << player << ", please select a card:";
288     cin >> input;
289     cout << "\n";
290     Card card = hand[input];
291     hand.erase(hand.begin() + input);
292     sort(hand.begin(), hand.end());
293     return card;
294 }
295
296 // REQUIRES Player has at least one card, trump is a valid suit
297 // EFFECTS Plays one Card from Player's hand according to their strategy.
298 // The card is removed from the player's hand.
299 Card play_card(const Card &led_card, const std::string &trump) override {
300     int input = 0;
301     print_hand();
302     cout << "Human player " << player << ", please select a card:";
303     cin >> input;
304     cout << "\n";
305     Card card = hand[input];
```

```
306     hand.erase(hand.begin() + input);
307     sort(hand.begin(), hand.end());
308     return card;
309 }
310 };
311
312 Player *Player_factory(const std::string &name, const std::string &strategy) {
313     if (strategy == "Simple") {
314         return new SimplePlayer(name);
315     }
316     if (strategy == "Human") {
317         return new HumanPlayer(name);
318     }
319     assert(false);
320     return nullptr;
321 }
```

[Give feedback](#)