

```

1 /**
2  * @file PlayerBot.js
3  * @description This file contains functions for the PlayerBot to deal cards
  during the game
4  * @author Senni Tan
5  * @version Latest edition on April 10, 2021
6  */
7
8 import * as Rules from './Rules.js'
9
10 /**
11  * @function BotPlayCards
12  * @description A function that takes the input of all cards that the playerBot
  has and
13  * an input of the cards last dealt by last player, and returns the selected
  cards for playerBot
14  * @param {card[]} cards
15  * @param {card[]} last
16  * @returns {card[]} selectedCards
17  */
18 export function BotPlayCards(cards, last) {
19     Rules.sortCardsValue(cards)
20     Rules.sortCardsValue(last)
21     var selectedCards
22
23     if (last.length === 1){
24         selectedCards = BotSelectSingle(cards, last)
25     } else if (last.length === 2){
26         selectedCards = BotSelectPair(cards, last)
27     } else if (last.length === 5){
28         selectedCards = BotSelectFive(cards, last)
29     } else {
30
31     }
32
33     return selectedCards
34 }
35
36 /**
37  * @function BotStartingTurn
38  * @description If the playerBot has a dimond 3, he will first deal out the
  dimond 3 in a round of game
39  * @param {card[]} cards
40  * @returns {card[]} [The dimond 3 card]
41  */
42 export function BotStartingTurn(cards) {
43     var i = 0
44     while (i < cards.length) {
45         if (cards[i].value === 3 && cards[i].suit === "D"){
46             return [cards[i]]
47         }
48         i++
49     }
50 }
51
52 /**
53  * @function BotFreeTurn
54  * @description When all other players pass, and this playerBot will deal out the
  smallest cards combo in the privilage of

```

```

55 * five cards -> pairs -> single card
56 * @param {card[]} cards
57 * @returns {card[]} a list of smallest cards combo it can deal out in the
  privilege of five -> pair -> single
58 */
59 export function BotFreeTurn(cards) {
60     Rules.sortCardsValue(cards)
61
62     var selectedCards = getAllFiveCards(cards)
63
64     if (selectedCards !== null && selectedCards.length !== 0){
65         return selectedCards[0]
66     }
67
68     selectedCards = getAllPairs(cards)
69     if (selectedCards !== null && selectedCards.length !== 0){
70         return selectedCards[0]
71     }
72
73     return [cards[0]]
74 }
75
76 /**
77 * @function BotSelectSingle
78 * @description A function that deals the smallest single card that is valid and
  stronger than the card that the last player dealt
79 * @param {card[]} cards - the cards that the playerBot has
80 * @param {card[]} last - the card(s) that the last player dealt
81 * @returns {card[]} the smallest card(s) that is valid and stronger than the
  card that the last player dealt
82 */
83 export function BotSelectSingle(cards, last) {
84
85     var i = 0
86     while (i < cards.length){
87         if (Rules.isStrongerSingle(last[0], cards[i])){
88             return [cards[i]]
89         }
90         i++
91     }
92
93     return null
94 }
95
96 /**
97 * @function BotSelectPair
98 * @description A function that deals the smallest pair that is valid and
  stronger than the cards that the last player dealt
99 * @param {card[]} cards - the cards that the playerBot has
100 * @param {card[]} last - the cards that the last player dealt
101 * @returns {card[]} the smallest pair that is valid and stronger than the pair
  that the last player dealt
102 */
103 export function BotSelectPair(cards, last) {
104     var pairs = getAllPairs(cards)
105
106     if (pairs){
107         let i = 0
108         while (i < pairs.length){
109             if (Rules.isStrongerPair(last, pairs[i])){

```

```

110         return pairs[i]
111     }
112     i++
113 }
114 }
115
116 return null
117 }
118
119 /**
120  * @function BotSelectFive
121  * @description A function that deals the smallest five-card combo that is valid
122  * and stronger than the cards that the last player dealt
123  * @param {card[]} cards - the cards that the playerBot has
124  * @param {card[]} last - the cards that the last player dealt
125  * @returns {card[]} the smallest five-card combo that is valid and stronger than
126  * the card that the last player dealt
127  */
128 export function BotSelectFive(cards, last) {
129     var combos = getAllFiveCards(cards)
130
131     if (combos) {
132         let i = 0
133         while (i < combos.length) {
134             if (Rules.isStrongerPlay(last, combos[i])) {
135                 return combos[i]
136             }
137             i++
138         }
139     }
140
141     return null
142 }
143
144 /**
145  * @function getAllFiveCards
146  * @description A function that returns all possible valid five-card combinations
147  * @param {card[]} cards - the cards that the playerBot has
148  * @returns {card[]} a list of all possible valid five-card combinations that the
149  * player bot has
150  */
151 function getAllFiveCards(cards) {
152     if (cards.length < 5) return null
153
154     var validCombos = []
155
156     function searchFiveCards(cards, subset, i) {
157         if (i === cards.length) {
158             subset = subset.filter(card => card !== null)
159             subset = subset.slice(0, 5)
160             if (Rules.isValidFiveCardPlay(subset)) {
161                 validCombos.push(subset)
162             }
163             return
164         }
165
166         subset[i] = cards[i]
167         searchFiveCards(cards, subset, i + 1)
168         subset[i] = null

```

```

166         searchFiveCards(cards, subset, i + 1)
167     }
168     searchFiveCards(cards, [], 0)
169
170     return validCombos
171 }
172 }
173
174 /**
175  * @function getAllPairs
176  * @description A function that returns all possible valid pairs
177  * @param {card[]} cards - the cards that the playerBot has
178  * @returns {card[]} a list of all possible valid pairs that the playerBot has
179  */
180 function getAllPairs(cards) {
181     var seenCards = new Map()
182     var pairs = []
183
184     var i = 0
185     while (i < cards.length){
186         if (seenCards.has(cards[i].type)) {
187             var lastSeenCard = seenCards.get(cards[i].type)
188             pairs.push([lastSeenCard, cards[i]])
189         } else {
190             seenCards.set(cards[i].type, cards[i])
191         }
192         i++
193     }
194
195     return pairs
196 }

```