```javascript
/**
 * @file PlayerBot.js
 * @description This file contains functions for the PlayerBot to deal cards 
during the game
 * @author Senni Tan
 * @version Latest edition on April 10, 2021
 */

import * as Rules from './Rules.js'

/**
 * @function BotPlayCards
 * @description A function that takes the input of all cards that the playerBot 
has and
 * an input of the cards last dealed by last player, and returns the selected 
cards for playerBot
 * @param {card[]} cards
 * @param {card[]} last
 * @returns {card[]} selectedCards
 */
export function BotPlayCards(cards, last) {
    Rules.sortCardsValue(cards)
    Rules.sortCardsValue(last)
    var selectedCards

    if (last.length === 1){
        selectedCards = BotSelectSingle(cards, last)
    } else if (last.length === 2){
        selectedCards = BotSelectPair(cards, last)
    } else if (last.length === 5){
        selectedCards = BotSelectFive(cards, last)
    } else {

    }

    return selectedCards
}

/**
 * @function BotStartingTurn
 * @description If the playerBot has a dimond 3, he will first deal out the 
dimond 3 in a round of game
 * @param {card[]} cards
 * @returns {card[]} [The dimond 3 card]
 */
export function BotStartingTurn(cards) {
    var i = 0
    while (i < cards.length) {
        if (cards[i].value === 3 && cards[i].suit === "D"){
            return [cards[i]]
        }
        i++
    }
}

/**
 * @function BotFreeTurn
 * @description When all other players pass, and this playerBot will deal out the 
smallest cards combo in the privilage of
```

```
 * five cards -> pairs -> single card
 * @param {card[]} cards
 * @returns {card[]} a list of smallest cards combo it can deal out in the privilage of five -> pair -> single
 */
export function BotFreeTurn(cards) {
    Rules.sortCardsValue(cards)

    var selectedCards = getAllFiveCards(cards)

    if (selectedCards !== null && selectedCards.length !== 0){
        return selectedCards[0]
    }

    selectedCards = getAllPairs(cards)
    if (selectedCards !== null && selectedCards.length !== 0){
        return selectedCards[0]
    }

    return [cards[0]]
}

/**
 * @function BotSelectSingle
 * @description A function that deals the smallest single card that is valid and stronger than the card that the last player dealed
 * @param {card[]} cards - the cards that the playerBot has
 * @param {card[]} last - the card(s) that the last player dealed
 * @returns {card[]} the smallest card(s) that is valid and stronger than the card that the last player dealed
 */
export function BotSelectSingle(cards, last) {

    var i = 0
    while (i < cards.length){
        if (Rules.isStrongerSingle(last[0], cards[i])){
            return [cards[i]]
        }
        i++
    }

    return null
}

/**
 * @function BotSelectPair
 * @description A function that deals the smallest pair that is valid and stronger than the cards that the last player dealed
 * @param {card[]} cards - the cards that the playerBot has
 * @param {card[]} last - the cards that the last player dealed
 * @returns {card[]} the smallest pair that is valid and stronger than the pair that the last player dealed
 */
export function BotSelectPair(cards, last) {
    var pairs = getAllPairs(cards)

    if (pairs){
        let i = 0
        while (i < pairs.length){
            if (Rules.isStrongerPair(last, pairs[i])){
```

```javascript
                return pairs[i]
            }
            i++
        }
    }

    return null
}

/**
 * @function BotSelectFive
 * @description A function that deals the smallest five-card combo that is valid and stronger than the cards that the last player dealed
 * @param {card[]} cards - the cards that the playerBot has
 * @param {card[]} last - the cards that the last player dealed
 * @returns {card[]} the smallest five-card combo that is valid and stronger than the card that the last player dealed
 */
export function BotSelectFive(cards, last) {
    var combos = getAllFiveCards(cards)

    if (combos){
        let i = 0
        while (i < combos.length){
            if (Rules.isStrongerPlay(last, combos[i])){
                return combos[i]
            }
            i++
        }
    }

    return null
}

/**
 * @function getAllFiveCards
 * @description A function that returns all possible valid five-card combinations
 * @param {card[]} cards - the cards that the playerBot has
 * @returns {card[]} a list of all possible valid five-card combinations that the player bot has
 */
function getAllFiveCards(cards) {
    if (cards.length < 5) return null

    var validCombos = []

    function searchFiveCards(cards, subset, i) {
        if (i === cards.length) {
            subset = subset.filter(card => card !== null)
            subset = subset.slice(0, 5)
            if (Rules.isValidFiveCardPlay(subset)) {
                validCombos.push(subset)
            }
            return
        }

        subset[i] = cards[i]
        searchFiveCards(cards, subset, i + 1)
        subset[i] = null
        searchFiveCards(cards, subset, i + 1)
    }
    searchFiveCards(cards, [], 0)

    return validCombos

}

/**
 * @function getAllPairs
 * @description A function that returns all possible valid pairs
 * @param {card[]} cards - the cards that the playerBot has
 * @returns {card[]} a list of all possible valid pairs that the playerBot has
 */
function getAllPairs(cards) {
    var seenCards = new Map()
    var pairs = []

    var i = 0
    while (i < cards.length){
        if (seenCards.has(cards[i].type)) {
            var lastSeenCard = seenCards.get(cards[i].type)
            pairs.push([lastSeenCard, cards[i]])
        } else {
            seenCards.set(cards[i].type, cards[i])
        }
        i++
    }

    return pairs
}
```