Home

src/bigtwo/src/App.js

This file contains the class APP that renders and generates the game on a web page.

Version: Latest edition on April 10, 2021

Author: Manyi Cheng
Source: App.js, line 1

src/bigtwo/src/components/Card.jsx

This file exports a Card react component.

Author: Manyi Cheng

Source: components/Card.jsx, line 1

src/bigtwo/src/components/Deck.jsx

This file exports a Deck react component.

Version: Latest edition on April 11, 2021

Author: Jiaxin Tang

Source: components/Deck.jsx, line 1

src/bigtwo/src/components/Game.jsx

React extension javascript that exports a Game react component.

Author: Manyi Cheng

Home

Modules

App

Deck

GameplayField

Timer

Classes

App Game

Global

BotFreeTurn

BotPlayCards

BotSelectFive

BotSelectPair

BotSelectSingle

 ${\tt BotStartingTurn}$

Card

Deck

GameplayField

getAllFiveCards

getAllPairs

getSuitValue

importAll

isStrongerFive

isStrongerPair

isStrongerPlay

isStrongerSingle

isValidFiveCardPlay

isValidFlush

isValidFourOfaKind

isValidFullHouse

isValidPair

Source: components/Game.jsx, line 1

isValidSingle
isValidSPlay
isValidStartingPlay
isValidStraight

src/bigtwo/src/components/GameplayField.jsx

This file exports a GameplayField react component.

Version: Latest edition on April 11, 2021

Author: Jiaxin Tang

Source: components/GameplayField.jsx, line 1

Player render

setFirstTurn

setUserCards

shuffle

sortCardsSuit
sortCardsValue

Timer

src/bigtwo/src/components/Player.jsx

React extension javascript that exports a Player react component.

Author: Manyi Cheng

Source: components/Player.jsx, line 1

src/bigtwo/src/components/Timer.js

This file generates a timer for the game.

Version: Latest edition on April 10, 2021

Author: Manyi Cheng

Source: components/Timer.js, line 1

src/bigtwo/src/PlayerBot.js

This file contains functions for the PlayerBot to deal cards during the game

Version: Latest edition on April 10, 2021

Author: Senni Tan

Source: PlayerBot.js, line 1

src/bigtwo/src/Rules.js

This file contains rules of BigTwo game.

Version: Latest edition on April 11, 2021

Author: Jiaxin Tang

Source: Rules.js, line 1

Documentation generated by JSDoc 3.6.6 on Sun Apr 11 2021 23:29:26 GMT-0400 (Eastern Daylight Time)

Class: App

App()

new App()

This is a class that renders and generates the game on a web page.

Source: App.js, line 12

Extends

Component

Home

Modules

App

Deck

GameplayField

Timer

Classes

Арр

Game

Global

 ${\tt BotFreeTurn}$

BotPlayCards

BotSelectFive

BotSelectPair

BotSelectSingle

BotStartingTurn

Card

Deck

GameplayField

getAllFiveCards

getAllPairs

getSuitValue

importAll

isStrongerFive

isStrongerPair

isStrongerPlay

isStrongerSingle

isValidFiveCardPlay

isValidFlush

isValidFourOfaKind

isValidFullHouse

isValidPair

isValidSingle
isValidSPlay
isValidStartingPlay
isValidStraight

newDeck

Player

render

setFirstTurn

setUserCards

shuffle

sortCardsSuit

sortCardsValue

Timer

Documentation generated by JSDoc 3.6.6 on Sun Apr 11 2021 23:29:26 GMT-0400 (Eastern Daylight Time)

Class: Game

Game(props)

A class that extends react Component, represents a big two Game.

Constructor

new Game(props)

This class represents Game component in a big two game.

Parameters:

Name	Туре	Description
props	*	Props from parent component.

Source: components/Game.jsx, line 25

Methods

BotPlayCards()

Controls the logic when its bot's turn to play cards.

Source: components/Game.jsx, line 176

 $computePlayerScore() \rightarrow \{int\}$

Computes player score of the game.

Source: components/Game.jsx, line 371

Returns:

Computed score.

Home

Modules

App

Deck

GameplayField

Timer

Classes

App

Game

Global

BotFreeTurn

BotPlayCards

BotSelectFive

BotSelectPair

BotSelectSingle

BotStartingTurn

Card

Deck

GameplayField

getAllFiveCards

getAllPairs

getSuitValue

importAll

isStrongerFive

isStrongerPair

isStrongerPlay

isStrongerSingle

isValidFiveCardPlay

isValidFlush

isValidFourOfaKind

isValidFullHouse

isValidPair

Type

int

displayPass()

Displays text when players choose to pass the current turn.

Source: components/Game.jsx, line 379

getCardsforTurn()

gets the current players' cards of the turn.

Source: components/Game.jsx, line 198

Returns:

current player cards
handlePlayerDeal(cards)

player action on clicking deal button with selected cards.

Parameters:

Name	Туре	Description
cards	*	Selected cards to be dealt.

Source: components/Game.jsx, line 135

Returns:

true if valid play, false if invalid play.

handlePlayerPass()

Handles player passing for starting turn, last move, free move and normal situations.

Source: components/Game.jsx, line 313

handleTimer()

Handles game over condition when the timer reaches 0.

Source: components/Game.jsx, line 124

isValidSingle

isValidSPlay

isValidStartingPlay

isValidStraight

newDeck

Player

render

setFirstTurn

setUserCards

shuffle

sortCardsSuit

sortCardsValue

Timer

isGameOver()

Checks whether the game is over and sets the game states gameOver and playerScore 1s after validation.

Source: components/Game.jsx, line 353

(async) resetGame()

Resets game states upon user clicking play again button.

Source: components/Game.jsx, line 88

startGame()

Starts the game upon user closing the rules.

Source: components/Game.jsx, line 76

suitSort()

Sorts player's cards in suit order upon player clicking suit button.

Source: components/Game.jsx, line 343

typeSort()

Sorts player's cards in type order upon player clicking type button.

Source: components/Game.jsx, line 334

UNSAFE_componentWillMount()

Execute the code synchronously when the component gets loaded or mounted in the DOM. This method is called during the mounting phase of the React Life-cycle

Deprecated: Will be decrecated be React in the future.

Source: components/Game.jsx, line 69

updateField(cards)

Updates the GamplayField when players deal cards.

Parameters:

Name	Туре	Description
cards	*	Field cards

Source: components/Game.jsx, line 268

updateNextTurn()

Set states turn, and field text for next turn, then on call back triggers next turn's play.

Source: components/Game.jsx, line 291

Returns:

Nothing

updateNextTurnCards(cards)

Updates state cards for next turn based on the cards dealt by the current player.

Parameters:

Name	Туре	Description
cards	*	Cards dealt by the current player.

Source: components/Game.jsx, line 209

Documentation generated by JSDoc 3.6.6 on Sun Apr 11 2021 23:29:26 GMT-0400 (Eastern Daylight Time)

Global

Methods

BotFreeTurn(cards) → {Array.<card>}

When all other players pass, and this playerBot will deal out the smallest cards combo in the privilage of five cards -> pairs -> single card

Parameters:

Name	Туре	Description
cards	Array. <card></card>	

Source: PlayerBot.js, line 52

Returns:

a list of smallest cards combo it can deal out in the privilage of five -> pair -> single

Type

Array.<card>

BotPlayCards(cards, last) → {Array.<card>}

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

Parameters:

Name	Туре	Description
cards	Array. <card></card>	
last	Array. <card></card>	

Source: PlayerBot.js, line 10

Home

Modules

App

Deck

GameplayField

Timer

Classes

App

Game

Global

BotFreeTurn

BotPlayCards

BotSelectFive

BotSelectPair

BotSelectSingle

BotStartingTurn

Card

Deck

GameplayField

getAllFiveCards

getAllPairs

getSuitValue

importAll

isStrongerFive

isStrongerPair

isStrongerPlay

isStrongerSingle

isValidFiveCardPlay

isValidFlush

isValidFourOfaKind

isValidFullHouse

isValidPair

Returns:

selectedCards

Type

Array.<card>

BotSelectFive(cards, last) → {Array.<card>}

A function that deals the smallest five-card combo that is valid and stronger than the cards that the last player dealed

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that the playerBot has
last	Array. <card></card>	the cards that the last player dealed

Source: PlayerBot.js, line 119

Returns:

the smallest five-card combo that is valid and stronger than the card that the last player dealed

Type

Array.<card>

BotSelectPair(cards, last) → {Array.<card>}

A function that deals the smallest pair that is valid and stronger than the cards that the last player dealed

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that the playerBot has
last	Array. <card></card>	the cards that the last player dealed

Source: PlayerBot.js, line 96

Returns:

the smallest pair that is valid and stronger than the pair that the last player dealed

isValidSingle
isValidSPlay
isValidStartingPlay
isValidStraight
newDeck
Player
render
setFirstTurn
setUserCards
shuffle
sortCardsSuit

sortCardsValue

Timer

Array.<card>

A function that deals the smallest single card that is valid and stronger than the card that the last player dealed

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that the playerBot has
last	Array. <card></card>	the card(s) that the last player dealed

Source: PlayerBot.js, line 76

Returns:

the smallest card(s) that is valid and stronger than the card that the last player dealed

Type

Array.<card>

BotStartingTurn(cards) → {Array.<card>}

If the playerBot has a dimond 3, he will first deal out the dimond 3 in a round of game

Parameters:

Name	Туре	Description
cards	Array. <card></card>	

Source: PlayerBot.js, line 36

Returns:

[The dimond 3 card]

Type

Array.<card>

Card(props)

This react arrow function represents a Card component in a BigTwo game.

Parameters:

Name	Туре	Description
props	*	Props from parent component.

Source: components/Card.jsx, line 12

Returns:

React div HTML element displaying the card.

Deck(props)

This react arrow function represents a deck component in a big two game.

Parameters:

Name	Туре	Description
props	*	Props from parent component.

Source: components/Deck.jsx, line 16

Returns:

a div element displaying the deck
GameplayField(props)

This react arrow function arranges the field for the cards that players dealt in a big two game.

Parameters:

Name	Туре	Description
props	*	Props from parent component.

Source: components/GameplayField.jsx, line 18

Returns:

a div element displaying the field getAllFiveCards(cards) → {Array.<card>}

A function that returns all possible valid five-card combinations

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that the playerBot has

Source:

PlayerBot.js, line 142

Returns:

a list of all possible valid five-card combinations that the player bot has

Type

Array.<card>

getAllPairs(cards) → {Array.<card>}

A function that returns all possible valid pairs

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that the playerBot has

Source:

PlayerBot.js, line 174

Returns:

a list of all possible valid pairs that the playerBot has

Type

Array.<card>

$getSuitValue(suit) \rightarrow \{int\}$

A function that gets the integer value of the corresponding suit.

Parameters:

Name	Туре	Description
suit	string	

Source:

Rules.js, line 384

Returns:

- integer value related to suit

Туре

int

importAll(r)

Imports all images from the parameter path r

Parameters:

Name	Туре	Description
r	*	Indicates the required path to the card image folder.

Source: components/Card.jsx, line 15

Returns:

List of json objects containing the images.

isStrongerFive(last, select) → {boolean}

A function that checks if the current five card play is stronger than last five card play

Parameters:

Name	Туре	Description
last	Array. <card></card>	the cards that the last player plays
select	Array. <card></card>	the cards that current player selects

Source: Rules.js, line 288

Returns:

- true if the select play is stronger than last play

Type

boolean

 $isStrongerPair(last, select) \rightarrow \{boolean\}$

A function that checks if the current pair is stronger than last pair

Parameters:

Name	Туре	Description
last	Array. <card></card>	the cards that the last player plays
select	Array. <card></card>	the cards that current player selects

Source: Rules.js, line 269

Returns:

- true if the select play is stronger than last play

Type

boolean

isStrongerPlay(last, select) → {boolean}

A function that checks if the current play is stronger than last play

Parameters:

Name	Туре	Description
last	Array. <card></card>	the cards that the last player plays
select	Array. <card></card>	the cards that current player selects

Source: Rules.js, line 217

Returns:

- true if the select play is stronger than last play

Type

boolean

$isStrongerSingle(last, select) \rightarrow \{boolean\}$

A function that checks if the current single is stronger than last single

Parameters:

Name	Туре	Description
last	Array. <card></card>	the cards that the last player plays
select	Array. <card></card>	the cards that current player selects

Source:

Rules.js, line 237

Returns:

- true if the select play is stronger than last play

Type

boolean

isValidFiveCardPlay(cards) → {boolean}

A function that checks if the current play is valid five card play

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source:

Rules.js, line 113

Returns:

- true if cards is a valid combination of five cards

Type

boolean

isValidFlush(cards) → {boolean}

A function that checks if the current play is valid flush

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source:

Rules.js, line 161

Returns:

- true if cards is a valid flush

Type

boolean

isValidFourOfaKind(cards) → {boolean}

A function that checks if the current play is valid four of a kind

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source: Rules.js, line 198

Returns:

- true if cards is a valid four of a kind

Type

boolean

isValidFullHouse(cards) → {boolean}

A function that checks if the current play is valid fullhouse

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source: Rules.js, line 180

Returns:

- true if cards is a valid fullhouse

Type

boolean

isValidPair(cards) → {boolean}

A function that checks if the current play is valid pair

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source:

Rules.js, line 103

Returns:

- true if cards is a valid pair

Type

boolean

isValidSingle(cards) → {boolean}

A function that checks if the current play is valid single play

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source:

Rules.js, line 93

Returns:

- true if cards contain a single card

Type

boolean

isValidSPlay(cards) → {boolean}

A function that checks if the current play is valid play

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source:

Rules.js, line 78

Returns:

- true if is valid play

Type

boolean

isValidStartingPlay(cards) → {boolean}

A function that checks if the current play is valid starting play

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player has

Source: Rules.js, line 58

Returns:

= true if cards contain Diamond 3

Type

boolean

isValidStraight(cards) → {boolean}

A function that checks if the current play is valid straight

Parameters:

Name	Туре	Description
cards	Array. <card></card>	the cards that current player selects

Source: Rules.js, line 125

Returns:

- true if cards is a valid straight

Type

boolean

newDeck() → {Array.<card>}

A function that generates a deck of 52 cards, and rearranges the order of cards in the deck

Source: Rules.js, line 14

Returns:

deck - with cards in random order

Type

Array.<card>

Player(props)

This react arrow function represents a Player component in a BigTwo game.

Parameters:

Name	Туре	Description
props	*	Props from parent component.

Source: components/Player.jsx, line 15

Returns:

React div HTML element displaying the player component render()

The funtion that generates the game on a web page

Source: App.js, line 19

Returns:

a div container that contains the web game
setFirstTurn(playerCards, opponentLeftCards,
opponentTopCards, opponentRightCards) →
{string}

A function that decides which player plays the first turn.

Parameters:

Name	Туре	Description
playerCards	Array. <card></card>	a list of cards that player has
opponentLeftCards	Array. <card></card>	a list of cards that left Al has
opponentTopCards	Array. <card></card>	a list of cards that top Al has

Name	Туре	Description
opponentRightCards	Array. <card></card>	a list of cards that right Al has

Source: Rules.js, line 355

Returns:

turn - represeting the initial player

Type

string

setUserCards(deck) → {Array.<card>}

A function that places 13 cards in a deck into a list to be assigned to a player.

Parameters:

Name	Туре	Description
deck	Array. <card></card>	a list of 52 cards in a random order

Source: Rules.js, line 341

Returns:

userCards - contains 13 cards for a player

Type

Array.<card>

shuffle(deck) → {Array.<card>}

A function that rearranges the order of cards in the given deck

Parameters:

Name	Туре	Description
deck	Array. <card></card>	a list of cards

Source: Rules.js, line 40

Returns:

deck - with cards in random order

Type

Array.<card>

sortCardsSuit(cards) → {Array.<card>}

A function that sorts the given cards in the suit rank order.

Parameters:

Name	Туре	Description
cards	Array. <card></card>	

Source:

Rules.js, line 408

Returns:

cards - ordered in the suit rank

Type

Array.<card>

sortCardsValue(cards) → {Array.<card>}

A function that sorts the given cards in the number rank order.

Parameters:

Name	Туре	Description
cards	Array. <card></card>	

Source:

Rules.js, line 394

Returns:

cards - ordered in the number rank

Type

Array.<card>

Timer(props)

Parameters:

Name	Туре	Description
props	*	

Source: components/Timer.js, line 10

Returns:

A timmer that counts down from 10 minutes on the upper right corner of the web page during the game

Documentation generated by JSDoc 3.6.6 on Sun Apr 11 2021 23:29:26 GMT-0400 (Eastern Daylight Time)

```
1 /**
 2 * @file App.js
 3 * @description This file contains the class APP that renders and generates the
  game on a web page.
 4 * @author Manyi Cheng
 5 * @version Latest edition on April 10, 2021
 6 */
 7 import React, { Component } from 'react';
 8 import Game from './components/Game.jsx'
 9 import './App.css';
10
11
12 /**
13 * @class App
14 * @description This is a class that renders and generates the game on a web page.
15 * @extends Component
16 */
17 class App extends Component {
18
19
20
    * @function render
21
    * @description The funtion that generates the game on a web page
22
    * @returns a div container that contains the web game
23
    */
24
   render() {
25
     return (
26
       <div className="App">
27
         <header className="App-header">
28
            <Game/>
29
          </header>
30
        </div>
     );
31
   }
32
33 |}
34
35 /**
36 * @exports App
37 */
38 export default App;
```

```
1 /**
  * @file Game.jsx
 3 * @description React extension javascript that exports a Game react component.
 4 * @author Manyi Cheng
 5 */
 6
7 import React, { Component } from 'react';
8 import Player from './Player.jsx';
9 import Deck from './Deck.jsx';
10 import GameplayField from './GameplayField.jsx';
11 import peachIcon from '.../res/peach.png';
12 import luigiIcon from '../res/luigi.png';
13 import boolcon from '../res/boo.png';
14 import Timer from './Timer.js';
15 import * as Rules from '../Rules.js';
16 import * as PlayerBot from '../PlayerBot.js';
17 import startButton from '../res/startbutton.png';
18
19
20 /**
21 * @class A class that extends react Component, represents a big two Game.
   * @description This class represents Game component in a big two game.
23 * @param {*} props Props from parent component.
24 */
25 class Game extends Component {
26
          constructor(props) {
27
                   super(props);
28
                   this.state = {
29
                           rules: true,
30
                           playerScore: 0,
31
                           playerCards: [],
32
                           leftCards: [],
33
                           topCards: [],
34
                           rightCards: [],
35
                           playerField: [],
36
                           leftField: [],
37
                           topField: [],
38
                           rightField: [],
39
                           startingTurn: true,
40
                           turn: null,
41
                           minutes: 10,
42
                           seconds: 0,
43
                           cardsPlayed: [],
44
                           freeMove: false,
45
                           lastMove: [],
46
                           lastMovePlayer: null,
47
                           gameOver: false,
48
                   };
49
                   this.startGame = this.startGame.bind(this);
50
                   this.resetGame = this.resetGame.bind(this);
51
                   this.handlePlayerDeal = this.handlePlayerDeal.bind(this);
52
                   this.handlePlayerPass = this.handlePlayerPass.bind(this);
53
                   this.BotPlayCards = this.BotPlayCards.bind(this);
54
                   this.updateNextTurn = this.updateNextTurn.bind(this);
55
                   this.updateField = this.updateField.bind(this);
56
                   this.updateNextTurnCards = this.updateNextTurnCards.bind(this);
57
                   this.getCardsforTurn = this.getCardsforTurn.bind(this);
58
                   this.typeSort = this.typeSort.bind(this);
```

```
59
                     this.handleTimer = this.handleTimer.bind(this);
 60
                     this.suitSort = this.suitSort.bind(this);
 61
                     this.isGameOver = this.isGameOver.bind(this);
 62
                     this.displayPass = this.displayPass.bind(this);
 63
            }
 64
 65
            /**
             ^{\star} <code>@description</code> Execute the code synchronously when the component gets load
 66
   mounted in the DOM. This method is called during the mounting phase of the React Li
 67
             * @deprecated Will be decrecated be React in the future.
             */
 68
 69
            UNSAFE componentWillMount() {
 70
                    this.resetGame();
 71
            }
 72
 73
            /**
 74
             * @description Starts the game upon user closing the rules.
 75
 76
            startGame() {
 77
                     this.setState({
 78
                            rules: false,
 79
                     });
 80
                     if (this.state.turn !== 'player') {
 81
                            this.BotPlayCards();
 82
                     }
 83
            }
 84
 85
            /**
 86
             * @description Resets game states upon user clicking play again button.
 87
 88
            async resetGame() {
 89
                     let deck = Rules.newDeck();
 90
 91
                     let playerCards = await Rules.setUserCards(deck);
 92
                     let leftCards = await Rules.setUserCards(deck);
 93
                     let topCards = await Rules.setUserCards(deck);
 94
                     let rightCards = await Rules.setUserCards(deck);
 95
 96
                     let turn = Rules.setFirstTurn(playerCards, leftCards, topCards, rig
 97
 98
                     this.setState({
 99
                             rules: true,
100
                             playerScore: 0,
101
                             playerField: [],
102
                             leftField: [],
103
                             topField: [],
104
                             rightField: [],
105
                             playerCards: playerCards,
106
                             leftCards: leftCards,
107
                             topCards: topCards,
108
                             rightCards: rightCards,
109
                             initialMinutes: 10,
110
                             initialSeconds: 0,
111
                             turn: turn,
112
                             startingTurn: true,
113
                             cardsPlayed: [],
114
                             lastMove: [],
115
                             lastMovePlayer: null,
116
                             gameOver: false,
```

```
117
                             playerFieldText: '',
118
                     });
119
            }
120
            / * *
121
122
             * Handles game over condition when the timer reaches 0.
123
124
            handleTimer() {
125
                    this.setState({
126
                             gameOver: true,
127
                     });
128
            }
129
            /**
130
131
             * @description player action on clicking deal button with selected cards.
132
             * @param {*} cards Selected cards to be dealt.
133
             * @returns true if valid play, false if invalid play.
134
135
            handlePlayerDeal(cards) {
136
                    this.setState({ playerFieldText: '' });
137
                    if (this.state.startingTurn) {
138
                             let validPlay = Rules.isValidStartingPlay(cards);
139
                             if (validPlay) {
140
141
                                     this.updateNextTurnCards(cards);
142
                                     this.setState({ startingTurn: false });
143
                                     return true;
144
                             } else {
145
                                     this.setState({
146
                                              playerFieldText: 'Your play must be valid a
    3 of diamonds for starting turn',
147
                                     });
148
                             }
149
                     } else {
150
                             let valid = Rules.isValidPlay(cards);
151
                             let isFreeMove = this.state.lastMovePlayer === 'player';
152
                             let stronger = Rules.isStrongerPlay(this.state.lastMove, ca
153
154
                             if (valid && (isFreeMove || stronger)) {
155
                                     this.updateNextTurnCards(cards);
156
                                     return true;
157
                             } else {
158
                                     if (!valid) {
159
                                              this.setState({
160
                                                      playerFieldText: 'Your play must be
161
                                              });
162
                                     } else if (!stronger && cards.length ===
    this.state.lastMove.length) {
                                              this.setState({ playerFieldText: 'Your play
163
    stronger than the previous play' });
164
                                     } else if (cards.length !== this.state.lastMove) {
165
                                              this.setState({
166
                                                      playerFieldText: 'Your play must cc
    number of cards as the previous play',
167
                                              });
168
                                     }
169
                             }
170
                     }
171
            }
```

```
172
173
            /**
174
             * @description Controls the logic when its bot's turn to play cards.
175
176
            BotPlayCards() {
177
                    let currentCards = this.getCardsforTurn();
178
                    let bestMove;
179
180
                    if (this.state.startingTurn) {
181
                            bestMove = PlayerBot.BotStartingTurn(currentCards);
182
                             this.setState({ startingTurn: false });
183
                    } else {
184
                             if (this.state.lastMovePlayer === this.state.turn) {
185
                                    bestMove = PlayerBot.BotFreeTurn(currentCards);
186
                             } else {
187
                                     bestMove = PlayerBot.BotPlayCards(currentCards,
    this.state.lastMove);
188
                             }
189
                    }
190
191
                    this.updateNextTurnCards(bestMove);
192
            }
193
            /**
194
195
             * @description gets the current players' cards of the turn.
196
             * @returns current player cards
             */
197
198
            getCardsforTurn() {
199
                    if (this.state.turn === 'left') return this.state.leftCards;
200
                    if (this.state.turn === 'top') return this.state.topCards;
201
                    if (this.state.turn === 'right') return this.state.rightCards;
202
                    if (this.state.turn === 'player') return this.state.playerCards;
203
            }
204
205
            /**
206
             * @description Updates state cards for next turn based on the cards dealt
    current player.
207
             * @param {*} cards Cards dealt by the current player.
208
209
            updateNextTurnCards(cards) {
210
                    if (cards) {
211
                             let cardsPlayed = this.state.cardsPlayed;
212
                             let currentPlayerCards = this.getCardsforTurn();
213
214
                             cards.forEach((card) => {
215
                                     currentPlayerCards.splice(currentPlayerCards.indexC
    1);
216
                             });
217
218
                             if (this.state.lastMove) {
219
                                     this.state.lastMove.forEach((card) => {
220
                                             cardsPlayed.push(card);
221
                                     });
222
                             }
223
224
                             if (this.state.turn === 'left') this.setState({ leftCards:
    currentPlayerCards });
225
                             if (this.state.turn === 'top') this.setState({ topCards:
   currentPlayerCards });
```

```
226
                             if (this.state.turn === 'right') this.setState({ rightCards
   currentPlayerCards });
227
                             if (this.state.turn === 'player') this.setState({ playerCar
    currentPlayerCards });
228
229
                             this.updateField(cards);
230
231
                             this.setState(
232
                                     {
233
                                              cardsPlayed: cardsPlayed,
234
                                              lastMove: cards,
235
                                              freeMove: false,
236
                                              lastMovePlayer: this.state.turn,
237
                                     },
238
                                      () => {
239
                                              this.updateNextTurn();
240
                                     }
241
                             );
242
                     } else {
243
                             if (this.state.turn === 'left')
244
                                     this.setState({ leftField: [] }, () => {
245
                                              this.displayPass();
246
                                     });
247
                             if (this.state.turn === 'top')
248
                                     this.setState({ topField: [] }, () => {
249
                                              this.displayPass();
250
                                     });
251
                             if (this.state.turn === 'right')
252
                                     this.setState({ rightField: [] }, () => {
253
                                              this.displayPass();
254
                                     });
255
                             if (this.state.turn === 'player')
256
                                     this.setState({ playerField: [] }, () => {
257
                                              this.displayPass();
258
                                     });
259
260
                             this.updateNextTurn();
261
                     }
262
            }
263
            /**
264
265
             * @description Updates the GamplayField when players deal cards.
266
             * @param {*} cards Field cards
267
268
            updateField(cards) {
269
                    if (this.state.turn === 'left')
270
                             this.setState({ leftField: [] }, () => {
271
                                     this.setState({ leftField: cards });
272
                             });
273
                    if (this.state.turn === 'top')
274
                             this.setState({ topField: [] }, () => {
275
                                     this.setState({ topField: cards });
276
                             });
277
                    if (this.state.turn === 'right')
278
                             this.setState({ rightField: [] }, () => {
279
                                     this.setState({ rightField: cards });
280
                             });
281
                    if (this.state.turn === 'player')
282
                             this.setState({ playerField: [] }, () => {
```

```
283
                                      this.setState({ playerField: cards });
284
                             });
285
            }
286
287
            / * *
             * @description Set states turn, and field text for next turn, then on call
288
    triggers next turn's play.
289
             * @returns Nothing
             * /
290
291
            updateNextTurn() {
292
                    if (this.isGameOver()) return;
293
                    setTimeout(() => {
294
                             if (this.state.turn === 'player') {
295
                                      this.setState({ turn: 'right', playerFieldText: ''
296
                                              this.BotPlayCards();
297
                                      });
298
                             } else if (this.state.turn === 'right') {
299
                                      this.setState({ turn: 'top' }, () => {
300
                                              this.BotPlayCards();
301
                                      });
302
                             } else if (this.state.turn === 'top') {
303
                                     this.setState({ turn: 'left' }, () => {
304
                                              this.BotPlayCards();
305
                                     });
306
                             } else this.setState({ turn: 'player' });
307
                     }, 1200);
308
            }
309
310
             * \thetadescription Handles player passing for starting turn, last move, free \pi
311
   normal situations.
312
             * /
313
            handlePlayerPass() {
314
                    if (this.state.startingTurn) {
315
                             this.setState({
316
                                      freeMove: true,
317
                                     playerFieldText: 'You cannot pass the first turn',
318
319
                     } else if (this.state.lastMovePlayer === 'player') {
320
                             this.setState({
321
                                     freeMove: true,
322
                                     playerFieldText: 'You cannot pass the free move',
323
                             });
324
                     } else {
325
                             this.setState({ playerField: [], playerFieldText: '' });
326
                             this.displayPass();
327
                             this.updateNextTurn();
328
                     }
329
            }
330
            /**
331
332
             * @description Sorts player's cards in type order upon player clicking typ
333
334
            typeSort() {
335
                    let cards = this.state.playerCards;
336
                    Rules.sortCardsValue(cards);
337
338
                    this.setState({ playerCards: cards });
339
            }
```

```
340
            /**
341
             * @description Sorts player's cards in suit order upon player clicking sui
342
343
            suitSort() {
344
                    let cards = this.state.playerCards;
345
                    Rules.sortCardsSuit(cards);
346
347
                    this.setState({ playerCards: cards });
348
            }
349
            /**
350
             * @description Checks whether the game is over and sets the game states ga
351
   playerScore 1s after validation.
352
             */
353
            isGameOver() {
354
                    let currentPlayerCards = this.getCardsforTurn();
355
                    if (currentPlayerCards.length === 0) {
356
                             let score = this.computePlayerScore();
357
                             setTimeout(() => {
358
                                     this.setState({
359
                                             gameOver: true,
360
                                              playerScore: score,
361
                                     });
362
                                     return true;
363
                             }, 1000);
364
                     }
365
            }
366
367
368
             * @description Computes player score of the game.
369
             * @returns {int} Computed score.
370
371
            computePlayerScore() {
372
                    let len = this.state.playerCards.length;
373
                    return Math.ceil((13 - len) * (100 / 13));
374
            }
375
376
            /**
377
             * @description Displays text when players choose to pass the current turn.
378
             * /
379
            displayPass() {
                    let field = this.state.turn;
380
381
                    let node = document.createElement('div');
382
                    node.append(document.createTextNode('Pass'));
383
                    node.setAttribute('class', 'gameplayfield-text');
384
                    document.getElementById(field).append(node);
385
                    setTimeout(() => {
386
                             document.getElementById(field).removeChild(node);
387
                    }, 1000);
388
            }
389
390
            render() {
391
                    if (this.state.rules) {
392
                             return (
393
                                     <div>
394
                                              <div className="game-container">
395
                                                      <div className="window-container">
396
                                                               <div className="window">
397
                                                                       <div className="rul
```

```
398
                                                                              <h4 classNa
   heading">
399
                                                                                      <sr
    className="rules-heading-span">Rules</span>
400
                                                                              </h4>
401
                                                                     </div>
402
                                                                     <div className="rul
   details">
403
                                                                              classNa
   details">
404
                                                                                      <1i
   A > K > Q > J > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 
405
                                                                                      <li
   Spades > hearts > clubs > diamonds
406
                                                                                      <1i
   combinations: single, pairs, triples, five-cards
407
                                                                                      <1i
408
   combination can only be beaten by a better combination with the same
409
   of cards.
410
                                                                                      </1
411
                                                                                      <1i
412
                                                                                      <1i
   consists of five cards of consecutive rank with mixed suits.
413
                                                                                      <1i
   consists of any five cards of the same suit.
414
                                                                                      <1i
415
   House consists of three cards of one rank and two of another rank
416
                                                                                      </1
417
                                                                                      <1i
   made up of all four cards of one rank, plus any fifth card
418
                                                                                      <1i
   Flush consists of five consecutive cards of the same suit.
419
                                                                              420
                                                                     </div>
421
                                                                     <div className="rul
422
                                                                              <img
423
   className="start-button"
424
   src={startButton}
425
                                                                                      onC
    {this.startGame}
426
                                                                                      alt
   button"
427
                                                                              />
428
                                                                     </div>
429
                                                                     <div>3XA3 G06</div>
430
                                                             </div>
431
                                                     </div>
432
                                             </div>
433
                                    </div>
434
                            );
435
                    } else {
436
                            return (
437
                                    <div>
438
                                             <div className="game-container">
439
                                                     {this.state.gameOver && <div
   className="window-container">
440
                                                             <div className="window">
```

```
441
                                                                          <div className="gam
    container">
442
                                                                                   <div>Game C
443
                                                                                   <div>Score
    {this.state.playerScore}</div>
444
                                                                                   <button
445
                                                                                           id=
   button"
446
                                                                                            dis
    {false}
447
    className="playagain-button"
448
                                                                                            onC
    {this.resetGame}
449
450
                                                                                            Pla
451
                                                                                   </button>
452
                                                                          </div>
453
                                                                 </div>
454
                                                        </div>}
455
                                                        <div className="game-opponent">
456
                                                                 <img src={boolcon} alt="cha"</pre>
    className="top-icon" />
457
                                                                 <img src={luigiIcon} alt="c</pre>
    className="opponent-icon" />
458
                                                                 <div className="game-left">
459
                                                                          <Deck
460 ,
                                                                                   class="oppc
    container-left"
461
                                                                                   cardClass="
   side"
462 ,
                                                                                   cards=
    {this.state.leftCards}
463
                                                                          ></Deck>
464
                                                                 </div>
465
                                                                 <div className="game-middle
466
                                                                          <Deck
467,
                                                                                  class="oppc
   container-top"
468
                                                                                   cardClass="
   top"
469
                                                                                   cards=
    {this.state.topCards}
470
                                                                          ></Deck>
471
                                                                          <GameplayField</pre>
472
                                                                                  player=
    {this.state.playerField}
473 |,
                                                                                   right=
    {this.state.rightField}
474
   left={this.state.leftField}
475
    top={this.state.topField}
476 ,
                                                                                   playerField
    {this.state.playerFieldText}
477
                                                                          ></GameplayField>
478
                                                                 </div>
479
                                                                 <div className="game-right"
480
                                                                          <Timer
481 ,
                                                                                   initialMinu
   {this.state.minutes}
```

```
482 ,
                                                                                  initialSecc
    {this.state.seconds}
483 |,
                                                                                  onTimer=
    {this.handleTimer}
484
                                                                         />
485
                                                                         <Deck
486 ,
                                                                                  class="oppc
   container-right"
487 ,
                                                                                  cardClass="
   side"
488
                                                                                  cards=
    {this.state.rightCards}
489 ,
                                                                         ></Deck>
490
                                                                </div>
491
                                                                <img src={peachIcon} alt="c</pre>
    className="opponent-icon" />
492
                                                        </div>
493
                                                        <Player
494
                                                                cards={this.state.playerCar
495 ,
                                                                playerTurn={this.state.turn
    'player'}
496,
                                                                freeMove={this.state.freeMc
497
                                                                playCards={this.handlePlaye
498
                                                                passTurn={this.handlePlayer
499 ,
                                                                turn={this.state.turn}
500 ,
                                                                typeSort={this.typeSort}
501,
                                                                suitSort={this.suitSort}
502 ,
                                                                gameOver={this.state.gameOv
503 .
                                                                playerScore={this.state.pla
504
                                                        ></Player>
505
                                               </div>
506
                                      </div>
507
                              );
508
                     }
509
            }
510 }
511
512 export default Game;
```

```
1 /**
 2 * @file GameplayField.jsx
 3
   * @description This file exports a GameplayField react component.
   * @author Jiaxin Tang
   * @version Latest edition on April 11, 2021
 6
 7
 8 import React from 'react';
 9 import Card from './Card.jsx'
10
11
12
13 /**
14 * @description This react arrow function arranges the field for the cards that
  players dealt in a big two game.
15 * @param {*} props Props from parent component.
   * @return a div element displaying the field
16
17
18 const GameplayField = (props) => {
19
20
       return (
21
           <div className="gameplayfield-container">
22
               <div className="gameplayfield-section section-top" id="top">
                   {props.top.map((card, i) => {return (<Card class="field-card"</pre>
23
   key={i} card={card} user="field" />)})
24
               </div>
25
               <div className="gameplayfield-section">
26
                   <div className="left-field" id="left">
                       {props.left.map((card, i) => {return (< Card class="field-card"
   key={i} card={card} user="field" />)})
2.8
                   </div>
29
                   <div className="right-field" id="right">
30
                       {props.right.map((card, i) => {return (<Card class="field-</pre>
   card" key={i} card={card} user="field" />)})}
31
                   </div>
32
33
               </div>
34
               <div className="gameplayfield-section section-top" id="player">
35
                   <div className="gameplayfield-player">
36
                        {props.player.map((card, i) => {return (<Card key={i}}</pre>
   card={card} class="field-card" user="field" />)})}
37
                   </div>
38
                   <div className="gameplayfield-text">{props.playerFieldText}</div>
39
               </div>
40
41
           )
42
43 }
44
45 | GameplayField.defaultProps = {
46
       props:{
47
           playerFieldText: "",
48
49 }
50
51 /**
52
   * @exports GameplayField
54 export default GameplayField
```

```
1 /**
 2 * @file Card.jsx
 3 * @description This file exports a Card react component.
   * @author Manyi Cheng
 5 */
 6 import React from 'react';
 7 /**
 8 * @description This react arrow function represents a Card component in a BigTwo
  game.
 9 * @param {*} props Props from parent component.
10 * @returns React div HTML element displaying the card.
11
   * /
12 const Card = (props) => {
13
           const path = props.card.imagePath;
           const images = importAll(require.context('../res/Asset', false,
   /\.png$/));
15
          /**
16
            * Imports all images from the parameter path r
17
            * @function importAll
18
            * @param {*} r Indicates the required path to the card image folder.
19
            * @returns List of json objects containing the images.
20
            */
21
           function importAll(r) {
22
                   let images = {};
23
                   r.keys().forEach((item) => {
24
                           images[item.replace('./', '')] = r(item);
25
                   });
26
                   return images;
27
           }
28
29
           if (props.user === 'opponent') {
30
                   return (
31
                            <div>
32
                                    <img className={props.class} alt="opponent-card"</pre>
   src={images['Back.png']} />
                           </div>
33
34
                   );
           } else if (props.user === 'player') {
35
                   const classname = props.selected ? 'selectedcard' : '';
36
37
                   return (
38
                            <div>
39
                                    <img
                                            onClick={() =>
  props.selectCard(props.card) }
41
                                            className={'card ' + classname}
42
                                            alt="player-card"
43
                                            src={images[path]}
44
                                    />
45
                            </div>
46
                   );
47
           } else {
                   return <img className={props.class + ' flip-in-ver-left'}</pre>
   alt="field-card" src={images[path]} />;
49
           }
50 };
51
52 Card.defaultProps = {
53
           props: {
                   user: '',
54
```

```
1 /**
 2 * @file Deck.jsx
 3 \mid \star @description This file exports a Deck react component.
 4 * @author Jiaxin Tang
 5 * @version Latest edition on April 11, 2021
 7
 8 import Card from './Card.jsx'
 9 import React from 'react'
10
11 /**
12 * @description This react arrow function represents a deck component in a big two
  game.
13 * @param {*} props Props from parent component.
14 * @return a div element displaying the deck
16 const Deck = (props) => {
if (props.cards) {
18
         return (
19
              <div className={"opponent-container " + props.class}>
20
                  {props.cards.map((card, i) => < Card class={props.cardClass}}
user="opponent" key={i} card={card}/>
21
                  ) }
22
              </div>
23
         )
24
     }
25 }
26
27 Deck.defaultProps = {
28 props:{
29
     cardClass: "",
30
31 |}
32
33 /**
34 | * @exports Deck
35 */
36 export default Deck
```

```
1 /**
 2 * @file Player.jsx
   * @description React extension javascript that exports a Player react component.
  * @author Manyi Cheng
 5 */
 6 import React, { useState } from 'react';
 7 import marioImg from '../res/mario.png';
 8 import Card from './Card.jsx';
10 /**
11 * @description This react arrow function represents a Player component in a
  BigTwo game.
12 * @param {*} props Props from parent component.
13 * @returns React div HTML element displaying the player component
15 const Player = (props) => {
16
          const [selectedCards, setSelectCard] = useState([]);
17
18
           /**
19
           * @description This react arrow function selects a card upon click
  event.
20
           * @param {*} card The clicked card
21
           * /
22
           const selectCard = (card) => {
23
                   let newSelectedCards = [];
24
                   if (selectedCards.includes(card)) {
25
                           const index = selectedCards.indexOf(card);
26
                           newSelectedCards = [...selectedCards.slice(0, index),
   ...selectedCards.slice(index + 1)];
27
                   } else {
28
                           newSelectedCards = selectedCards.concat([card]);
29
30
                   setSelectCard(newSelectedCards);
31
           };
32
33
           * @description This react arrow function handles player click upon user
  clicking deal button.
35
            * @param {*} e Click event
36
           * /
37
          const handleDeal = (e) => {
38
                   e.preventDefault();
39
                   if (props.playerTurn) {
40
                           if (props.playCards(selectedCards)) {
41
                                   setSelectCard([]);
42
                           }
43
                           document.getElementById('playbtn').disabled = true;
44
                           setTimeout(() => {
45
                                   if (document.getElementById('playbtn'))
  document.getElementById('playbtn').disabled = false;
46
                           }, 1500);
47
                   }
48
           };
49
           /**
50
           * @description This react arrow function handles player click upon user
51
  clicking pass button.
52
           * @param {*} e Click event
53
           * /
```

```
54
            const handlePass = (e) => {
 55
                    e.preventDefault();
 56
                    if (props.playerTurn) {
 57
                             props.passTurn();
 58
                             document.getElementById('passbtn').disabled = true;
 59
                             setTimeout(() => {
 60
                                     if (document.getElementById('passbtn'))
   document.getElementById('passbtn').disabled = false;
 61
                             }, 1500);
 62
                     }
 63
            };
 64
            / * *
 65
 66
             * @description This react arrow function sorts the player deck based on
    type order in increasing order upon user clicking type button.
 67
            const handleTypeSort = () => {
 68
 69
                    props.typeSort();
 70
            };
 71
 72
 73
             * @description This react arrow function sorts the player deck based on
   suit order in increasing order upon user clicking suit button.
 74
 75
            const handleSuitSort = () => {
 76
                    props.suitSort();
 77
            };
 78
 79
            let actionButton = props.playerTurn ? '' : 'disabled-button';
 80
            let freeMoveButton = !props.freeMove ? '' : 'disabled-button';
 81
            return (
 82
                    <div className="player-container">
 83
                <img className = "player-icon" alt = "character" src = {marioImg}/>
 84
                             {props.cards &&
 85
                                     props.cards.map((card, i) => {
 86
                                              let selected =
   selectedCards.includes(card);
 87
                                              return <Card key={i} card={card}</pre>
   user="player" selectCard={selectCard} selected={selected} />;
 88
                                      })}
 89
                             {!props.gameOver && (
 90
                                      <div className="player-action">
 91
                                              <button id="playbtn" className={'player-</pre>
   button ' + actionButton} onClick={handleDeal}>
 92
                                                       Deal
 93
                                              </button>
 94
                                              <button
 95
                                                       id="passbtn"
                                                       className={'player-button' +
   actionButton + ' ' + freeMoveButton}
 97
                                                       onClick={handlePass}
 98
 99
                                                       Pass
100
                                              </button>
101
                                              <button className="player-button"</pre>
   onClick={handleTypeSort}>
102
                                                       Type
103
                                              </button>
104
                                              <button className="player-button"</pre>
   onClick={handleSuitSort}>
```

```
105 | Suit
106 | </button>
107 | </div>
108 | )}
109 | </div>
110 |);
111 |};
112 |
113 | export default Player;
```

```
1 /**
 2 * @file Timer.js
   * @description This file generates a timer for the game.
   * @author Manyi Cheng
 5 * @version Latest edition on April 10, 2021
 7 import React, { useState, useEffect } from 'react';
 8
 9
10 /**
11 * @function Timer
12 | * @param {*} props
   * @returns A timmer that counts down from 10 minutes on the upper right corner of
  the web page during the game
14 | */
15 const Timer = (props) => {
16
           const { initialMinutes = 0, initialSeconds = 0 } = props;
17
           const [minutes, setMinutes] = useState(initialMinutes);
18
           const [seconds, setSeconds] = useState(initialSeconds);
19
20
           useEffect(() => {
21
                   let myInterval = setInterval(() => {
22
                            if (seconds > 0) {
23
                                    setSeconds (seconds - 1);
24
                            }
25
                            if (seconds ===0) {
26
                                    if (minutes === 0) {
27
                                            clearInterval (myInterval);
28
                                    } else {
29
                                             setMinutes(minutes - 1);
30
                                             setSeconds (59);
31
32
33
                   }, 1000);
34
                   return () => {
35
                           clearInterval (myInterval);
36
           }, [minutes, seconds]);
37
38
39
       useEffect(() =>{
40
           if (minutes === 0 && seconds === 0) {
41
               console.log("times up")
42
               props.onTimer()
43
           }
44
       }, [minutes, seconds]);
45
           return (
46
47
                   <div className = "timer-container" >
48
                            {minutes === 0 && seconds === 0 ? null: (
49
                                    <div>
50
                                             { ' '}
51
                                             {minutes}:{seconds < 10 ? `0${seconds}` :</pre>
  seconds }
                                    </div>
52
53
                            ) }
54
                   </div>
55
           );
56|};
57
```

```
58  /**
59  * @exports Timer
60  */
61 export default Timer;
```

```
1 /**
 2 * @file PlayerBot.js
 3 * @description This file contains functions for the PlayerBot to deal cards
  during the game
  * @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 dealed 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)
2.5
      } 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) {</pre>
          if (cards[i].value === 3 && cards[i].suit === "D") {
45
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
   privilage of five -> pair -> single
 58 */
 59 export function BotFreeTurn(cards) {
       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 dealed
 79 * @param {card[]} cards - the cards that the playerBot has
 80 * @param {card[]} last - the card(s) that the last player dealed
 81 * @returns {card[]} the smallest card(s) that is valid and stronger than the
   card that the last player dealed
 82 | */
 83 export function BotSelectSingle(cards, last) {
 84
 85
       var i = 0
 86
       while (i < cards.length) {</pre>
 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 dealed
 99 * @param {card[]} cards - the cards that the playerBot has
100 * @param {card[]} last - the cards that the last player dealed
101 * @returns {card[]} the smallest pair that is valid and stronger than the pair
   that the last player dealed
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) {</pre>
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
    and stronger than the cards that the last player dealed
122 * @param {card[]} cards - the cards that the playerBot has
123 * @param {card[]} last - the cards that the last player dealed
124 * @returns {card[]} the smallest five-card combo that is valid and stronger than
    the card that the last player dealed
125
126 export function BotSelectFive(cards, last) {
127
       var combos = getAllFiveCards(cards)
128
129
      if (combos) {
130
           let i = 0
131
            while (i < combos.length) {</pre>
132
                if (Rules.isStrongerPlay(last, combos[i])){
133
                    return combos[i]
134
                }
135
                i++
136
            }
137
        }
138
139
       return null
140 }
141
142 /**
143 * @function getAllFiveCards
144 * @description A function that returns all possible valid five-card combinations
145 * @param {card[]} cards - the cards that the playerBot has
146 * @returns {card[]} a list of all possible valid five-card combinations that the
    player bot has
147 | */
148 | function getAllFiveCards (cards) {
149
       if (cards.length < 5) return null</pre>
150
151
       var validCombos = []
152
153
        function searchFiveCards(cards, subset, i) {
154
            if (i === cards.length) {
155
                subset = subset.filter(card => card !== null)
156
                subset = subset.slice(0, 5)
157
                if (Rules.isValidFiveCardPlay(subset)) {
158
                    validCombos.push(subset)
159
                }
160
                return
161
            }
162
163
            subset[i] = cards[i]
164
            searchFiveCards(cards, subset, i + 1)
165
            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) {</pre>
186
           if (seenCards.has(cards[i].type)) {
187
               var lastSeenCard = seenCards.get(cards[i].type)
               pairs.push([lastSeenCard, cards[i]])
188
189
            } else {
190
               seenCards.set(cards[i].type, cards[i])
191
            }
            i++
192
193
       }
194
195
       return pairs
196 }
```

```
1 /**
 2 * @file Rules.js
   * @description This file contains rules of BigTwo game.
 4 * @author Jiaxin Tang
 5 * @version Latest edition on April 11, 2021
 6
 7
 8 const suitsPath = ["Diamonds", "Clubs", "Hearts", "Spades"]
 9 const valuesPath = ["", "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10".
  "Jack", "Queen", "King"]
10 const suits = ["D", "C", "H", "S"]
11 const SuiteVal = [1, 2, 3, 4]
12 const type = ["", "A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q",
13
14 |/**
15 * @function newDeck
16 * @description A function that generates a deck of 52 cards, and rearranges the
  order of cards in the deck
17 | * @returns {card[]} deck - with cards in random order
19 export function newDeck() {
20
      let deck = []
21
22
      for (let i = 1; i < 14; i++) {
23
           for (let j = 0; j < 4; j++) {</pre>
24
               let value = (i === 1) ? 14 : (i === 2) ? 15 : i
               let imagePath = "NAP-01 "+ suitsPath[j] + "_"+ valuesPath[i] + ".png"
2.5
26
               let card = {
27
                   type: type[i],
28
                   suit: suits[j],
29
                   suiteVal : SuiteVal[j],
30
                   value: value,
31
                   imagePath: imagePath
32
               }
33
               deck.push (card)
34
           }
35
36
37
     return shuffle (deck)
38 }
39
40 /**
41 * @function shuffle
42 | * @description A function that rearranges the order of cards in the given deck
43 * @param {card[]} deck - a list of cards
44 * @returns {card[]} deck - with cards in random order
45 */
46
47 function shuffle (deck) {
48
      var temp, i, j;
49
      for (i = deck.length - 1; i > 0; i--) {
50
           j = Math.floor(Math.random() * (i + 1));
51
           temp = deck[i];
52
           deck[i] = deck[j];
53
           deck[j] = temp;
54
55
      return deck;
56 }
```

```
57
 58 | / * *
 59 * @function isValidStartingPlay
 60 * @description A function that checks if the current play is valid starting play
 61 * @param {card[]} cards - the cards that current player has
 62 * @returns {boolean} = true if cards contain Diamond 3
 63 */
 64 export function isValidStartingPlay(cards) {
       let containsThreeOfDiamonds
 66
 67
       cards.forEach((card) => {
 68
           if (card.suit === "D" && card.value === 3) containsThreeOfDiamonds = true
 69
 70
 71
       if (containsThreeOfDiamonds) {
 72
           return isValidPlay(cards)
 73
       } else {
 74
           return false
 75
       }
 76 }
 77
 78 /**
 79 * @function isValidSPlay
 80 | * @description A function that checks if the current play is valid play
 81 * @param {card[]} cards - the cards that current player selects
 82 * @returns {boolean} - true if is valid play
 83 */
 84
 85 export function isValidPlay(cards) {
 86
      if (cards == null) return false
 87
       sortCardsValue(cards)
 88
       return isValidSingle(cards) || isValidPair(cards) ||
 89
   isValidFiveCardPlay(cards)
 90 }
 91
 92
 93 /**
 94 * @function isValidSingle
 95 * @description A function that checks if the current play is valid single play
 96 * @param {card[]} cards - the cards that current player selects
 97
    * @returns {boolean} - true if cards contain a single card
 98 */
 99 export function isValidSingle(cards) {
100
      return cards.length === 1
101 |}
102
103 /**
104 * @function isValidPair
105 * @description A function that checks if the current play is valid pair
106 * @param {card[]} cards - the cards that current player selects
107 | * @returns {boolean} - true if cards is a valid pair
108 */
109 export function isValidPair(cards) {
110
      return cards.length === 2 && cards[0].type === cards[1].type
111 |}
112
113 /**
114 * @function isValidFiveCardPlay
```

```
115 * @description A function that checks if the current play is valid five card
   play
116 * @param {card[]} cards - the cards that current player selects
117 * @returns {boolean} - true if cards is a valid combination of five cards
119 export function isValidFiveCardPlay(cards) {
120
     if (cards.length !== 5) return false
121
122
       return isValidStraight(cards) || isValidFlush(cards) ||
   isValidFullHouse(cards) || isValidFourOfaKind(cards)
123 }
124
125 /**
126 * @function isValidStraight
127
    * @description A function that checks if the current play is valid straight
128 * @param {card[]} cards - the cards that current player selects
129 * @returns {boolean} - true if cards is a valid straight
130
131 function isValidStraight(cards) {
132
       if(cards.length !== 5)
133
           return false
134
      //12345
135
       sortCardsValue(cards)
136
      if(cards[0].value === 14){
137
           if(cards[1].value === 15 && cards[2].value === 3 &&
138
                cards[3].value === 4 && cards[4].value === 5)
139
                    return true
140
            else
141
                return false
142
143
       //23456
144
       if(cards[0].value === 15){
            if(cards[1].value === 3 && cards[2].value === 4 &&
145
146
                cards[3].value === 5 && cards[4].value === 6)
147
                    return true
148
            else
149
               return false
150
151
       var flag = true
152
       for(var i = 0; i < 4; i++) {</pre>
153
            if((cards[i].value + 1) !== cards[i+1].value) {
                flag = false
154
155
                return flag
156
            }
157
       }
158
       return flag
159 }
160
161 | / * *
162 * @function isValidFlush
163 * @description A function that checks if the current play is valid flush
164
    * @param {card[]} cards - the cards that current player selects
165 * @returns {boolean} - true if cards is a valid flush
166 */
167 function isValidFlush(cards) {
168
      if(cards.length !== 5)
169
           return false
170
       var flag = true
171
       for(var i = 1; i < 5; i++) {</pre>
```

```
172
           if(cards[i].suiteVal !== cards[0].suiteVal){
173
                flag = false
174
               return flag
175
            }
176
177
       return flag
178 }
179
180 /**
181 * @function isValidFullHouse
182 * @description A function that checks if the current play is valid fullhouse
183
    * @param {card[]} cards - the cards that current player selects
184 * @returns {boolean} - true if cards is a valid fullhouse
185 */
186 function isValidFullHouse(cards) {
187
      if(cards.length !== 5)
188
           return false
189
      sortCardsValue(cards)
190
      if(cards[0].value === cards[1].value && cards[0].value === cards[2].value &&
191
           cards[3].value === cards[4].value)
192
            return true
193
      if(cards[0].value === cards[1].value && cards[2].value === cards[3].value &&
194
           cards[2].value === cards[4].value)
195
           return true
196
       return false}
197
198 | / * *
199 * @function isValidFourOfaKind
200 * @description A function that checks if the current play is valid four of a
201 * @param {card[]} cards - the cards that current player selects
    * @returns {boolean} - true if cards is a valid four of a kind
202
203 */
204 function isValidFourOfaKind(cards) {
205
       if(cards.length !== 5)
206
           return false
207
      sortCardsValue(cards)
208
       if(cards[0].value === cards[1].value && cards[0].value === cards[2].value &&
209
           cards[0].value === cards[3].value)
210
           return true
211
      if(cards[4].value === cards[1].value && cards[4].value === cards[2].value &&
212
           cards[4].value === cards[3].value)
213
           return true
214
       return false
215 }
216
217 /**
218 * @function isStrongerPlay
219 * @description A function that checks if the current play is stronger than last
   play
220 * @param {card[]} last - the cards that the last player plays
221 * @param {card[]} select - the cards that current player selects
222 * @returns {boolean} - true if the select play is stronger than last play
223 */
224 export function isStrongerPlay(last, select) {
225
      var n = select.length
226
       if(n !== last.length)
227
           return false
228
       switch(n) {
```

```
229
           case 1: return isStrongerSingle(last, select);
230
           case 2: return isStrongerPair(last, select);
231
           case 5: return isStrongerFive(last, select);
232
           default:
233
               return false
234
       }
235 }
236
237 /**
238 * @function isStrongerSingle
239 * @description A function that checks if the current single is stronger than
   last single
240 * @param {card[]} last - the cards that the last player plays
241 * @param {card[]} select - the cards that current player selects
242 * @returns {boolean} - true if the select play is stronger than last play
243 */
244 export function isStrongerSingle(last, select) {
       if(select[0] && last[0]){
246
           if(select[0].value > last[0].value)
247
               return true
248
           if(select[0].value === last[0].value && select[0].suiteVal >
   last[0].suiteVal)
249
               return true
250
       }else if(select[0] && !last[0]){
251
           if(select[0].value > last.value)
252
               return true
253
           if(select[0].value === last.value && select[0].suiteVal > last.suiteVal)
254
               return true
       }else if(!select[0] && !last[0]){
255
256
           if(select.value > last.value)
257
               return true
258
           if(select.value === last.value && select.suiteVal > last.suiteVal)
259
               return true
260
       }else if(!select[0] && last[0]){
261
           if(select.value > last[0].value)
262
               return true
263
           if(select.value === last[0].value && select.suiteVal > last[0].suiteVal)
264
               return true
265
       }
266
       return false
267 }
268
269 /**
270 * @function isStrongerPair
271 * @description A function that checks if the current pair is stronger than last
   pair
272 * @param {card[]} last - the cards that the last player plays
273 * @param {card[]} select - the cards that current player selects
274 * @returns {boolean} - true if the select play is stronger than last play
275 */
276 export function isStrongerPair(last, select) {
277
     if(!isValidPair(select))
278
           return false
279
      if(select[0].value > last[0].value)
280
           return true
281
      sortCardsSuit(select)
282
       sortCardsSuit(last)
283
      if(select[0].value === last[0].value && select[1].suiteVal >
   last[1].suiteVal)
```

```
284
           return true
285
       return false
286 }
287
288 /**
289 * @function isStrongerFive
290 * @description A function that checks if the current five card play is stronger
   than last five card play
291 * @param {card[]} last - the cards that the last player plays
292 * @param {card[]} select - the cards that current player selects
293 * @returns {boolean} - true if the select play is stronger than last play
294 */
295 export function isStrongerFive(last, select) {
296
       if(isValidFourOfaKind(select) && isValidFullHouse(last))
297
           return true
298
       if(isValidFourOfaKind(select) && isValidFlush(last))
299
           return true
300
       if(isValidFourOfaKind(select) && isValidStraight(last))
301
            return true
302
       if(isValidFullHouse(select) && isValidFlush(last))
303
           return true
304
       if(isValidFullHouse(select) && isValidStraight(last))
305
            return true
306
       if(isValidFlush(select) && isValidStraight(last))
307
            return true
308
       if(isValidStraight(select) && isValidStraight(last)){
309
           sortCardsValue(select)
310
           sortCardsValue(last)
311
           if(select[4].value > last[4].value)
312
                return true
313
           else
314
                return false
315
       }
316
      if(isValidFlush(select) && isValidFlush(last)) {
317
            sortCardsValue(select)
318
           sortCardsValue(last)
319
           if(select[0].suiteVal > last[0].suiteVal)
320
                return true
           if(select[0].suiteVal === last[0].suiteVal && select[4].value >
321
   last[4].value)
322
               return true
323
           return false
324
325
       if(isValidFullHouse(select) && isValidFullHouse(last)) {
326
           sortCardsValue(select)
327
            sortCardsValue(last)
328
            if (select[3].value > last[3].value)
329
               return true
330
           return false
331
       }
332
       if(isValidFourOfaKind(select) && isValidFourOfaKind(last)) {
333
           sortCardsValue(select)
334
           sortCardsValue(last)
335
           if(select[3].value > last[3].value)
336
                return true
337
           return false
338
       }
339 }
340
```

```
341 /**
342 * @function setUserCards
343 * @description A function that places 13 cards in a deck into a list to be
   assigned to a player.
344 * @param {card[]} deck - a list of 52 cards in a random order
345 * @returns {card[]} userCards - contains 13 cards for a player
346 */
347 export function setUserCards(deck) {
     let userCards = []
349
       for (let i = 0; i < 13; i++) {</pre>
350
          userCards.push(deck.pop())
351
      }
352
       return userCards
353 }
354
355 /**
356 * @function setFirstTurn
357 * @description A function that decides which player plays the first turn.
358 * @param {card[]} playerCards - a list of cards that player has
359 * @param {card[]} opponentLeftCards - a list of cards that left AI has
360 * @param {card[]} opponentTopCards - a list of cards that top AI has
361 * @param {card[]} opponentRightCards - a list of cards that right AI has
362 * @returns {string} turn - represeting the initial player
363 */
364 export function setFirstTurn(playerCards, opponentLeftCards, opponentTopCards,
   opponentRightCards) {
365
       let turn
366
       playerCards.forEach((card) => {
367
           if (card.suit === "D" && card.value === 3) turn = "player"
368
369
370
       opponentLeftCards.forEach((card) => {
371
           if (card.suit === "D" && card.value === 3) turn = "left"
372
       })
373
374
       opponentTopCards.forEach((card) => {
375
           if (card.suit === "D" && card.value === 3) turn = "top"
376
       })
377
378
       opponentRightCards.forEach((card) => {
379
           if (card.suit === "D" && card.value === 3) turn = "right"
380
       })
381
       return turn
382 }
383
384 /**
385 * @function getSuitValue
386 * @description A function that gets the integer value of the corresponding suit.
387 * @param {string} suit
388 * @returns {int} - integer value related to suit
390 export function getSuitValue(suit) {
    return (suit === "D") ? 1 : (suit === "C") ? 2 : (suit === "H") ? 3 : 4
391
392 }
393
394 /**
395 * @function sortCardsValue
396 * @description A function that sorts the given cards in the number rank order.
397 * @param {card[]} cards
```

```
398 * @returns {card[]} cards - ordered in the number rank
399 */
400 export function sortCardsValue(cards) {
401 if (cards == null) return
402
403 | cards.sort((a, b) => {
404
     return a.value - b.value
     })
405
406 }
407
408 /**
409 * @function sortCardsSuit
410 * @description A function that sorts the given cards in the suit rank order.
411 * @param {card[]} cards
412 * @returns {card[]} cards - ordered in the suit rank
413 */
414 export function sortCardsSuit(cards) {
415
     if (cards == null) return
416
417
     cards.sort((a, b) => {
418
       return a.suiteVal - b.suiteVal
419
      })
420 }
```