

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
BotStartingTurn
Card
Deck
GameplayField
getAllFiveCards
getAllPairs
getSuitValue
importAll
isStrongerFive
isStrongerPair
isStrongerPlay
isStrongerSingle
isValidFiveCardPlay
isValidFlush
isValidFourOfaKind
isValidFullHouse
isValidPair

Source: [components/Game.jsx, line 1](#)

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](#)

isValidSingle
isValidSPlay
isValidStartingPlay
isValidStraight
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

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

- isValidSingle
- isValidSPlay
- isValidStartingPlay
- isValidStraight
- newDeck
- Player
- render
- setFirstTurn
- setUserCards
- shuffle
- sortCardsSuit
- sortCardsValue
- Timer

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	Type	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() → {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	Type	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 deprecated by React in the future.

Source: [components/Game.jsx, line 69](#)

updateField(cards)

Updates the GamplayField when players deal cards.

Parameters:

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

Source: [components/Game.jsx, line 209](#)

Global

Methods

BotFreeTurn(cards) → {Array.<card>}

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

Parameters:

Name	Type	Description
cards	Array.<card>	

Source: [PlayerBot.js, line 52](#)

Returns:

a list of smallest cards combo it can deal out in the privilege 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 dealt by last player, and returns the selected cards for playerBot

Parameters:

Name	Type	Description
cards	Array.<card>	
last	Array.<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 dealt

Parameters:

Name	Type	Description
cards	Array.<card>	the cards that the playerBot has
last	Array.<card>	the cards that the last player dealt

Source: [PlayerBot.js, line 119](#)

Returns:

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

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 dealt

Parameters:

Name	Type	Description
cards	Array.<card>	the cards that the playerBot has
last	Array.<card>	the cards that the last player dealt

Source: [PlayerBot.js, line 96](#)

Returns:

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

isValidSingle
isValidSPlay
isValidStartingPlay
isValidStraight
newDeck
Player
render
setFirstTurn
setUserCards
shuffle
sortCardsSuit
sortCardsValue
Timer

Type

Array.<card>

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

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

Parameters:

Name	Type	Description
cards	Array.<card>	the cards that the playerBot has
last	Array.<card>	the card(s) that the last player dealt

Source: [PlayerBot.js, line 76](#)

Returns:

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

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	Type	Description
cards	Array.<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	Type	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	Type	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	Type	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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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) → {int}`

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

Parameters:

Name	Type	Description
suit	string	

Source: [Rules.js, line 384](#)

Returns:

- integer value related to suit

Type

int

`importAll(r)`

Imports all images from the parameter path r

Parameters:

Name	Type	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	Type	Description
last	Array.<card>	the cards that the last player plays
select	Array.<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) → {boolean}`

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

Parameters:

Name	Type	Description
last	Array.<card>	the cards that the last player plays
select	Array.<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	Type	Description
last	Array.<card>	the cards that the last player plays
select	Array.<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) → {boolean}`

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

Parameters:

Name	Type	Description
last	Array.<card>	the cards that the last player plays
select	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<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	Type	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	Type	Description
playerCards	Array.<card>	a list of cards that player has
opponentLeftCards	Array.<card>	a list of cards that left AI has
opponentTopCards	Array.<card>	a list of cards that top AI has

Name	Type	Description
opponentRightCards	Array. <card>	a list of cards that right AI has

Source: [Rules.js, line 355](#)

Returns:

turn - representing 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	Type	Description
deck	Array.<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	Type	Description
deck	Array.<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	Type	Description
cards	Array.<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	Type	Description
cards	Array.<card>	

Source: [Rules.js, line 394](#)

Returns:

cards - ordered in the number rank

Type

Array.<card>

`Timer(props)`

Parameters:

Name	Type	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 /**
2  * @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 booIcon 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.
22  * @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     /**
66     * @description Execute the code synchronously when the component gets load
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
140         if (validPlay) {
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
147 3 of diamonds for starting turn',
148             });
149         }
150     } else {
151         let valid = Rules.isValidPlay(cards);
152         let isFreeMove = this.state.lastMovePlayer === 'player';
153         let stronger = Rules.isStrongerPlay(this.state.lastMove, ca
154
155         if (valid && (isFreeMove || stronger)) {
156             this.updateNextTurnCards(cards);
157             return true;
158         } else {
159             if (!valid) {
160                 this.setState({
161                     playerFieldText: 'Your play must be
162
163                 } else if (!stronger && cards.length ===
164 this.state.lastMove.length) {
165                 this.setState({ playerFieldText: 'Your play
166 stronger than the previous play' });
167             } else if (cards.length !== this.state.lastMove) {
168                 this.setState({
169                     playerFieldText: 'Your play must cc
170 number of cards as the previous play',
171
172                 });
173             }
174         }
175     }
176 }
177

```

```

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,
188 this.state.lastMove);
189         }
190     }
191
192     this.updateNextTurnCards(bestMove);
193 }
194
195 /**
196  * @description gets the current players' cards of the turn.
197  * @returns current player cards
198  */
199 getCardsforTurn() {
200     if (this.state.turn === 'left') return this.state.leftCards;
201     if (this.state.turn === 'top') return this.state.topCards;
202     if (this.state.turn === 'right') return this.state.rightCards;
203     if (this.state.turn === 'player') return this.state.playerCards;
204 }
205
206 /**
207  * @description Updates state cards for next turn based on the cards dealt
208  * @param {*} cards Cards dealt by the current player.
209  */
210 updateNextTurnCards(cards) {
211     if (cards) {
212         let cardsPlayed = this.state.cardsPlayed;
213         let currentPlayerCards = this.getCardsforTurn();
214
215         cards.forEach((card) => {
216             currentPlayerCards.splice(currentPlayerCards.indexC
217 1);
218             });
219
220         if (this.state.lastMove) {
221             this.state.lastMove.forEach((card) => {
222                 cardsPlayed.push(card);
223             });
224         }
225
226         if (this.state.turn === 'left') this.setState({ leftCards:
227 currentPlayerCards });
228         if (this.state.turn === 'top') this.setState({ topCards:
229 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 /**
288  * @description Set states turn, and field text for next turn, then on call
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 /**
311  * @description Handles player passing for starting turn, last move, free n
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  /**
351   * @description Checks whether the game is over and sets the game states ga
352   * playerScore ls after validation.
353   */
354  isGameOver() {
355      let currentPlayerCards = this.getCardsforTurn();
356      if (currentPlayerCards.length === 0) {
357          let score = this.computePlayerScore();
358          setTimeout(() => {
359              this.setState({
360                  gameOver: true,
361                  playerScore: score,
362              });
363              return true;
364          }, 1000);
365      }
366  }
367
368  /**
369   * @description Computes player score of the game.
370   * @returns {int} Computed score.
371   */
372  computePlayerScore() {
373      let len = this.state.playerCards.length;
374      return Math.ceil((13 - len) * (100 / 13));
375  }
376
377  /**
378   * @description Displays text when players choose to pass the current turn.
379   */
380  displayPass() {
381      let field = this.state.turn;
382      let node = document.createElement('div');
383      node.append(document.createTextNode('Pass'));
384      node.setAttribute('class', 'gameplayfield-text');
385      document.getElementById(field).append(node);
386      setTimeout(() => {
387          document.getElementById(field).removeChild(node);
388      }, 1000);
389  }
390
391  render() {
392      if (this.state.rules) {
393          return (
394              <div>
395                  <div className="game-container">
396                      <div className="window-container">
397                          <div className="window">
398                              <div className="rul

```



```

398 heading">
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440

```

```

441 |                                     <div className="game-container">
442 |                                     <div>Game C
443 |                                     <div>Score
444 | {this.state.playerScore}</div>
445 |                                     <button
446 |                                     id=
447 |                                     {false}
448 |                                     className="playagain-button"
449 |                                     {this.resetGame}
450 |                                     >
451 |                                     Pla
452 |                                     </button>
453 |                                     </div>
454 | </div>}
455 | <div className="game-opponent">
456 |     <img src={booIcon} alt="cha
457 |     <img src={luigiIcon} alt="c
458 |     <div className="game-left">
459 |         <Deck
460 |             class="oppc
461 |             cardClass="
462 |             side"
463 |             cards=
464 |             {this.state.leftCards}
465 |             ></Deck>
466 |     </div>
467 |     <div className="game-middle">
468 |         <Deck
469 |             class="oppc
470 |             cardClass="
471 |             top"
472 |             cards=
473 |             {this.state.topCards}
474 |             ></Deck>
475 |         <GameplayField
476 |             player=
477 |             right=
478 |             left={this.state.leftField}
479 |             top={this.state.topField}
480 |             playerFieldc
481 |             {this.state.playerFieldText}
482 |             ></GameplayField>
483 |     </div>
484 |     <div className="game-right">
485 |         <Timer
486 |             initialMinu
487 |             {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 className="opponent-icon" />
    <img src={peachIcon} alt="c
492 </div>
493 <Player
494 ,                                     cards={this.state.playerCar
495 ,                                     playerTurn={this.state.turr
'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.
4  * @author Jiaxin Tang
5  * @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
15  * players dealt in a big two game.
16  * @param {*} props Props from parent component.
17  * @return a div element displaying the field
18  */
19
20 const GameplayField = (props) => {
21
22     return (
23         <div className="gameplayfield-container">
24             <div className="gameplayfield-section section-top" id="top">
25                 {props.top.map((card, i) => {return (<Card class="field-card"
26 key={i} card={card} user="field" />)})}
27             </div>
28             <div className="gameplayfield-section">
29                 <div className="left-field" id="left">
30                     {props.left.map((card, i) => {return (<Card class="field-card"
31 key={i} card={card} user="field" />)})}
32                 </div>
33                 <div className="right-field" id="right">
34                     {props.right.map((card, i) => {return (<Card class="field-
35 card" key={i} card={card} user="field" />)})}
36                 </div>
37             </div>
38             <div className="gameplayfield-section section-top" id="player">
39                 <div className="gameplayfield-player">
40                     {props.player.map((card, i) => {return (<Card key={i}
41 card={card} class="field-card" user="field" />)})}
42                 </div>
43                 <div className="gameplayfield-text">{props.playerFieldText}</div>
44             </div>
45         </div>
46     )
47 }
48
49 GameplayField.defaultProps = {
50     props:{
51         playerFieldText: "",
52     }
53 }
54
55 /**
56  * @exports GameplayField
57  */
58 export default GameplayField

```

```

1 /**
2  * @file Card.jsx
3  * @description This file exports a Card react component.
4  * @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;
14     const images = importAll(require.context('../res/Asset', false,
15 /\.png$/));
16     /**
17      * Imports all images from the parameter path r
18      * @function importAll
19      * @param {*} r Indicates the required path to the card image folder.
20      * @returns List of json objects containing the images.
21      */
22     function importAll(r) {
23         let images = {};
24         r.keys().forEach((item) => {
25             images[item.replace('./', '')] = r(item);
26         });
27         return images;
28     }
29     if (props.user === 'opponent') {
30         return (
31             <div>
32                 <img className={props.class} alt="opponent-card"
src={images['Back.png']} />
33             </div>
34         );
35     } else if (props.user === 'player') {
36         const classname = props.selected ? 'selectedcard' : '';
37         return (
38             <div>
39                 <img
40 ,
41 props.selectCard(props.card) }
42                 className={'card ' + classname}
43                 alt="player-card"
44                 src={images[path]}
45             />
46             </div>
47         );
48     } else {
49         return <img className={props.class + ' flip-in-ver-left'}
alt="field-card" src={images[path]} />;
50     }
51 };
52 Card.defaultProps = {
53     props: {
54         user: '',

```

```
55 |         },  
56 |     };  
57 |  
58 | export default Card;
```

```

1 /**
2  * @file Deck.jsx
3  * @description This file exports a Deck react component.
4  * @author Jiaxin Tang
5  * @version Latest edition on April 11, 2021
6  */
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
13  game.
14  * @param {*} props Props from parent component.
15  * @return a div element displaying the deck
16  */
17 const Deck = (props) => {
18   if(props.cards){
19     return (
20       <div className={"opponent-container " + props.class}>
21         {props.cards.map((card, i) => <Card class={props.cardClass}
22 user="opponent" key={i} card={card}/>
23         )}
24       </div>
25     )
26   }
27 }
28 Deck.defaultProps = {
29   props:{
30     cardClass: "",
31   }
32 }
33 /**
34  * @exports Deck
35  */
36 export default Deck

```

```

1 /**
2  * @file Player.jsx
3  * @description React extension javascript that exports a Player react component.
4  * @author Manyi Cheng
5  */
6 import React, { useState } from 'react';
7 import marioImg from '../res/mario.png';
8 import Card from './Card.jsx';
9
10 /**
11  * @description This react arrow function represents a Player component in a
12  * BigTwo game.
13  * @param {*} props Props from parent component.
14  * @returns React div HTML element displaying the player component
15  */
16 const Player = (props) => {
17     const [selectedCards, setSelectCard] = useState([]);
18
19     /**
20      * @description This react arrow function selects a card upon click
21      * event.
22      * @param {*} card The clicked card
23      */
24     const selectCard = (card) => {
25         let newSelectedCards = [];
26         if (selectedCards.includes(card)) {
27             const index = selectedCards.indexOf(card);
28             newSelectedCards = [...selectedCards.slice(0, index),
29 ...selectedCards.slice(index + 1)];
30         } else {
31             newSelectedCards = selectedCards.concat([card]);
32         }
33         setSelectCard(newSelectedCards);
34     };
35
36     /**
37      * @description This react arrow function handles player click upon user
38      * clicking deal button.
39      * @param {*} e Click event
40      */
41     const handleDeal = (e) => {
42         e.preventDefault();
43         if (props.playerTurn) {
44             if (props.playCards(selectedCards)) {
45                 setSelectCard([]);
46             }
47             document.getElementById('playbtn').disabled = true;
48             setTimeout(() => {
49                 if (document.getElementById('playbtn'))
50 document.getElementById('playbtn').disabled = false;
51             }, 1500);
52         }
53     };
54
55     /**
56      * @description This react arrow function handles player click upon user
57      * clicking pass button.
58      * @param {*} e Click event
59      */

```



```

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      */
68     const handleTypeSort = () => {
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}
user="player" selectCard={selectCard} selected={selected} />;
88                     })}
89                 {!props.gameOver && (
90                     <div className="player-action">
91                         <button id="playbtn" className={'player-
button ' + actionButton} onClick={handleDeal}>
92                             Deal
93                         </button>
94                         <button
95                             id="passbtn"
96                             className={'player-button ' +
actionButton + ' ' + freeMoveButton}
97                             onClick={handlePass}>
98                             >
99                             Pass
100                         </button>
101                         <button className="player-button"
onClick={handleTypeSort}>
102                             Type
103                         </button>
104                         <button className="player-button"
onClick={handleSuitSort}>

```

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

```

1 /**
2  * @file Timer.js
3  * @description This file generates a timer for the game.
4  * @author Manyi Cheng
5  * @version Latest edition on April 10, 2021
6  */
7 import React, { useState, useEffect } from 'react';
8
9
10 /**
11  * @function Timer
12  * @param {*} props
13  * @returns A timer that counts down from 10 minutes on the upper right corner of
14  * the web page during the game
15  */
16 const Timer = (props) => {
17     const { initialMinutes = 0, initialSeconds = 0 } = props;
18     const [minutes, setMinutes] = useState(initialMinutes);
19     const [seconds, setSeconds] = useState(initialSeconds);
20
21     useEffect(() => {
22         let myInterval = setInterval(() => {
23             if (seconds > 0) {
24                 setSeconds(seconds - 1);
25             }
26             if (seconds === 0) {
27                 if (minutes === 0) {
28                     clearInterval(myInterval);
29                 } else {
30                     setMinutes(minutes - 1);
31                     setSeconds(59);
32                 }
33             }
34             }, 1000);
35         return () => {
36             clearInterval(myInterval);
37         };
38     }, [minutes, seconds]);
39
40     useEffect(() => {
41         if (minutes === 0 && seconds === 0) {
42             console.log("times up")
43             props.onTimer()
44         }
45     }, [minutes, seconds]);
46
47     return (
48         <div className = "timer-container" >
49             {minutes === 0 && seconds === 0 ? null: (
50                 <div>
51                     {' '}
52                     {minutes}:{seconds < 10 ? `0${seconds}` :
53                     seconds}
54                 </div>
55             )}
56         </div>
57     );
58 };
59

```

```
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
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 }

```



```

1 /**
2  * @file Rules.js
3  * @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",
10 "Jack", "Queen", "King"]
11 const suits = ["D", "C", "H", "S"]
12 const SuiteVal = [1, 2, 3, 4]
13 const type = ["", "A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q",
14 "K"]
15
16 /**
17  * @function newDeck
18  * @description A function that generates a deck of 52 cards, and rearranges the
19  order of cards in the deck
20  * @returns {card[]} deck - with cards in random order
21  */
22 export function newDeck() {
23     let deck = []
24
25     for (let i = 1; i < 14; i++) {
26         for (let j = 0; j < 4; j++) {
27             let value = (i === 1) ? 14 : (i === 2) ? 15 : i
28             let imagePath = "NAP-01_" + suitsPath[j] + "_" + valuesPath[i] + ".png"
29             let card = {
30                 type: type[i],
31                 suit: suits[j],
32                 suiteVal : SuiteVal[j],
33                 value: value,
34                 imagePath: imagePath
35             }
36             deck.push(card)
37         }
38     }
39
40     return shuffle(deck)
41 }
42
43 /**
44  * @function shuffle
45  * @description A function that rearranges the order of cards in the given deck
46  * @param {card[]} deck - a list of cards
47  * @returns {card[]} deck - with cards in random order
48  */
49
50 function shuffle(deck) {
51     var temp, i, j;
52     for (i = deck.length - 1; i > 0; i--) {
53         j = Math.floor(Math.random() * (i + 1));
54         temp = deck[i];
55         deck[i] = deck[j];
56         deck[j] = temp;
57     }
58     return deck;
59 }

```

```

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) {
65     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
89     return isValidSingle(cards) || isValidPair(cards) ||
90     isValidFiveCardPlay(cards)
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
118 */
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){
145         if(cards[1].value === 3 && cards[2].value === 4 &&
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++){
153         if((cards[i].value + 1) !== cards[i+1].value){
154             flag = false
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++){

```

```

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  * kind
202  * @param {card[]} cards - the cards that current player selects
203  * @returns {boolean} - true if cards is a valid four of a kind
204  */
205 function isValidFourOfaKind(cards) {
206     if(cards.length !== 5)
207         return false
208     sortCardsValue(cards)
209     if(cards[0].value === cards[1].value && cards[0].value === cards[2].value &&
210         cards[0].value === cards[3].value)
211         return true
212     if(cards[4].value === cards[1].value && cards[4].value === cards[2].value &&
213         cards[4].value === cards[3].value)
214         return true
215     return false
216 }
217
218 /**
219  * @function isStrongerPlay
220  * @description A function that checks if the current play is stronger than last
221  * play
222  * @param {card[]} last - the cards that the last player plays
223  * @param {card[]} select - the cards that current player selects
224  * @returns {boolean} - true if the select play is stronger than last play
225  */
226 export function isStrongerPlay(last, select) {
227     var n = select.length
228     if(n !== last.length)
229         return false
230     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){
245     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
255     }else if(!select[0] && !last[0]){
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
321         if(select[0].suiteVal === last[0].suiteVal && select[4].value >
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) {
348     let userCards = []
349     for (let i = 0; i < 13; i++) {
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
389  */
390 export function getSuitValue(suit) {
391     return (suit === "D") ? 1 : (suit === "C") ? 2 : (suit === "H") ? 3 : 4
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 | }

```