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

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

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

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

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.

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

`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	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 privilage 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 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	Type	Description
`cards`	Array.<card>	
`last`	Array.<card>	

Source: PlayerBot.js, line 10

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	Type	Description
`cards`	Array.<card>	the cards that the playerBot has
`last`	Array.<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	Type	Description
`cards`	Array.<card>	the cards that the playerBot has
`last`	Array.<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

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 dealed

Parameters:

Name	Type	Description
`cards`	Array.<card>	the cards that the playerBot has
`last`	Array.<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	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 - 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	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)

```
/**
 * @file App.js
 * @description This file contains the class APP that renders and generates the
game on a web page.
 * @author Manyi Cheng
 * @version Latest edition on April 10, 2021
 */
import React, { Component } from 'react';
import Game from './components/Game.jsx'
import './App.css';


/**
 * @class App
 * @description This is a class that renders and generates the game on a web page.
 * @extends Component
 */
class App extends Component {

  /**
   * @function render
   * @description The funtion that generates the game on a web page
   * @returns a div container that contains the web game
   */
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <Game/>
        </header>
      </div>
    );
  }
}

/**
 * @exports App
 */
export default App;
```

```
/**
 * @file Game.jsx
 * @description React extension javascript that exports a Game react component.
 * @author Manyi Cheng
 */

import React, { Component } from 'react';
import Player from './Player.jsx';
import Deck from './Deck.jsx';
import GameplayField from './GameplayField.jsx';
import peachIcon from '../res/peach.png';
import luigiIcon from '../res/luigi.png';
import booIcon from '../res/boo.png';
import Timer from './Timer.js';
import * as Rules from '../Rules.js';
import * as PlayerBot from '../PlayerBot.js';
import startButton from '../res/startbutton.png';


/**
 * @class A class that extends react Component, represents a big two Game.
 * @description This class represents Game component in a big two game.
 * @param {*} props Props from parent component.
 */
class Game extends Component {
        constructor(props) {
                super(props);
                this.state = {
                        rules: true,
                        playerScore: 0,
                        playerCards: [],
                        leftCards: [],
                        topCards: [],
                        rightCards: [],
                        playerField: [],
                        leftField: [],
                        topField: [],
                        rightField: [],
                        startingTurn: true,
                        turn: null,
                        minutes: 10,
                        seconds: 0,
                        cardsPlayed: [],
                        freeMove: false,
                        lastMove: [],
                        lastMovePlayer: null,
                        gameOver: false,
                };
                this.startGame = this.startGame.bind(this);
                this.resetGame = this.resetGame.bind(this);
                this.handlePlayerDeal = this.handlePlayerDeal.bind(this);
                this.handlePlayerPass = this.handlePlayerPass.bind(this);
                this.BotPlayCards = this.BotPlayCards.bind(this);
                this.updateNextTurn = this.updateNextTurn.bind(this);
                this.updateField = this.updateField.bind(this);
                this.updateNextTurnCards = this.updateNextTurnCards.bind(this);
                this.getCardsforTurn = this.getCardsforTurn.bind(this);
                this.typeSort = this.typeSort.bind(this);
                this.handleTimer = this.handleTimer.bind(this);
                this.suitSort = this.suitSort.bind(this);
                this.isGameOver = this.isGameOver.bind(this);
                this.displayPass = this.displayPass.bind(this);
        }

        /**
         * @description Execute the code synchronously when the component gets loac
mounted in the DOM. This method is called during the mounting phase of the React Li
         * @deprecated Will be decrecated be React in the future.
         */
        UNSAFE_componentWillMount() {
                this.resetGame();
        }

        /**
         * @description Starts the game upon user closing the rules.
         */
        startGame() {
                this.setState({
                        rules: false,
                });
                if (this.state.turn !== 'player') {
                        this.BotPlayCards();
                }
        }

        /**
         * @description Resets game states upon user clicking play again button.
         */
        async resetGame() {
                let deck = Rules.newDeck();

                let playerCards = await Rules.setUserCards(deck);
                let leftCards = await Rules.setUserCards(deck);
                let topCards = await Rules.setUserCards(deck);
                let rightCards = await Rules.setUserCards(deck);

                let turn = Rules.setFirstTurn(playerCards, leftCards, topCards, ric

                this.setState({
                        rules: true,
                        playerScore: 0,
                        playerField: [],
                        leftField: [],
                        topField: [],
                        rightField: [],
                        playerCards: playerCards,
                        leftCards: leftCards,
                        topCards: topCards,
                        rightCards: rightCards,
                        initialMinutes: 10,
                        initialSeconds: 0,
                        turn: turn,
                        startingTurn: true,
                        cardsPlayed: [],
                        lastMove: [],
                        lastMovePlayer: null,
                        gameOver: false,
                        playerFieldText: '',
                });
        }

        /**
         * Handles game over condition when the timer reaches 0.
         */
        handleTimer() {
                this.setState({
                        gameOver: true,
                });
        }

        /**
         * @description player action on clicking deal button with selected cards.
         * @param {*} cards Selected cards to be dealt.
         * @returns true if valid play, false if invalid play.
         */
        handlePlayerDeal(cards) {
                this.setState({ playerFieldText: '' });
                if (this.state.startingTurn) {
                        let validPlay = Rules.isValidStartingPlay(cards);

                        if (validPlay) {
                                this.updateNextTurnCards(cards);
                                this.setState({ startingTurn: false });
                                return true;
                        } else {
                                this.setState({
                                        playerFieldText: 'Your play must be valid a
3 of diamonds for starting turn',
                                });
                        }
                } else {
                        let valid = Rules.isValidPlay(cards);
                        let isFreeMove = this.state.lastMovePlayer === 'player';
                        let stronger = Rules.isStrongerPlay(this.state.lastMove, ca

                        if (valid && (isFreeMove || stronger)) {
                                this.updateNextTurnCards(cards);
                                return true;
                        } else {
                                if (!valid) {
                                        this.setState({
                                                playerFieldText: 'Your play must be
                                        });
                                } else if (!stronger && cards.length ===
this.state.lastMove.length) {
                                        this.setState({ playerFieldText: 'Your play
stronger than the previous play' });
                                } else if (cards.length !== this.state.lastMove) {
                                        this.setState({
                                                playerFieldText: 'Your play must cc
number of cards as the previous play',
                                        });
                                }
                        }
                }
        }
```

```

	/**
	 * @description Controls the logic when its bot's turn to play cards.
	 */
	BotPlayCards() {
		let currentCards = this.getCardsforTurn();
		let bestMove;

		if (this.state.startingTurn) {
			bestMove = PlayerBot.BotStartingTurn(currentCards);
			this.setState({ startingTurn: false });
		} else {
			if (this.state.lastMovePlayer === this.state.turn) {
				bestMove = PlayerBot.BotFreeTurn(currentCards);
			} else {
				bestMove = PlayerBot.BotPlayCards(currentCards, this.state.lastMove);
			}
		}

		this.updateNextTurnCards(bestMove);
	}

	/**
	 * @description gets the current players' cards of the turn.
	 * @returns current player cards
	 */
	getCardsforTurn() {
		if (this.state.turn === 'left') return this.state.leftCards;
		if (this.state.turn === 'top') return this.state.topCards;
		if (this.state.turn === 'right') return this.state.rightCards;
		if (this.state.turn === 'player') return this.state.playerCards;
	}

	/**
	 * @description Updates state cards for next turn based on the cards dealt current player.
	 * @param {*} cards Cards dealt by the current player.
	 */
	updateNextTurnCards(cards) {
		if (cards) {
			let cardsPlayed = this.state.cardsPlayed;
			let currentPlayerCards = this.getCardsforTurn();

			cards.forEach((card) => {
				currentPlayerCards.splice(currentPlayerCards.indexC
1);
			});

			if (this.state.lastMove) {
				this.state.lastMove.forEach((card) => {
					cardsPlayed.push(card);
				});
			}

			if (this.state.turn === 'left') this.setState({ leftCards: currentPlayerCards });
			if (this.state.turn === 'top') this.setState({ topCards: currentPlayerCards });
```

```
            if (this.state.turn === 'right') this.setState({ rightCards
currentPlayerCards });
            if (this.state.turn === 'player') this.setState({ playerCar
currentPlayerCards });

            this.updateField(cards);

            this.setState(
                {
                    cardsPlayed: cardsPlayed,
                    lastMove: cards,
                    freeMove: false,
                    lastMovePlayer: this.state.turn,
                },
                () => {
                    this.updateNextTurn();
                }
            );
        } else {
            if (this.state.turn === 'left')
                this.setState({ leftField: [] }, () => {
                    this.displayPass();
                });
            if (this.state.turn === 'top')
                this.setState({ topField: [] }, () => {
                    this.displayPass();
                });
            if (this.state.turn === 'right')
                this.setState({ rightField: [] }, () => {
                    this.displayPass();
                });
            if (this.state.turn === 'player')
                this.setState({ playerField: [] }, () => {
                    this.displayPass();
                });

            this.updateNextTurn();
        }
    }

    /**
     * @description Updates the GamplayField when players deal cards.
     * @param {*} cards Field cards
     */
    updateField(cards) {
        if (this.state.turn === 'left')
            this.setState({ leftField: [] }, () => {
                this.setState({ leftField: cards });
            });
        if (this.state.turn === 'top')
            this.setState({ topField: [] }, () => {
                this.setState({ topField: cards });
            });
        if (this.state.turn === 'right')
            this.setState({ rightField: [] }, () => {
                this.setState({ rightField: cards });
            });
        if (this.state.turn === 'player')
            this.setState({ playerField: [] }, () => {
```

```
                                this.setState({ playerField: cards });
                        });
        }

        /**
         * @description Set states turn, and field text for next turn, then on call
triggers next turn's play.
         * @returns Nothing
         */
        updateNextTurn() {
                if (this.isGameOver()) return;
                setTimeout(() => {
                        if (this.state.turn === 'player') {
                                this.setState({ turn: 'right', playerFieldText: ''
                                        this.BotPlayCards();
                                });
                        } else if (this.state.turn === 'right') {
                                this.setState({ turn: 'top' }, () => {
                                        this.BotPlayCards();
                                });
                        } else if (this.state.turn === 'top') {
                                this.setState({ turn: 'left' }, () => {
                                        this.BotPlayCards();
                                });
                        } else this.setState({ turn: 'player' });
                }, 1200);
        }

        /**
         * @description Handles player passing for starting turn, last move, free m
normal situations.
         */
        handlePlayerPass() {
                if (this.state.startingTurn) {
                        this.setState({
                                freeMove: true,
                                playerFieldText: 'You cannot pass the first turn',
                        });
                } else if (this.state.lastMovePlayer === 'player') {
                        this.setState({
                                freeMove: true,
                                playerFieldText: 'You cannot pass the free move',
                        });
                } else {
                        this.setState({ playerField: [], playerFieldText: '' });
                        this.displayPass();
                        this.updateNextTurn();
                }
        }

        /**
         * @description Sorts player's cards in type order upon player clicking typ
         */
        typeSort() {
                let cards = this.state.playerCards;
                Rules.sortCardsValue(cards);

                this.setState({ playerCards: cards });
        }
```

```
	/**
	 * @description Sorts player's cards in suit order upon player clicking sui
	 */
	suitSort() {
		let cards = this.state.playerCards;
		Rules.sortCardsSuit(cards);

		this.setState({ playerCards: cards });
	}

	/**
	 * @description Checks whether the game is over and sets the game states ga
playerScore 1s after validation.
	 */
	isGameOver() {
		let currentPlayerCards = this.getCardsforTurn();
		if (currentPlayerCards.length === 0) {
			let score = this.computePlayerScore();
			setTimeout(() => {
				this.setState({
					gameOver: true,
					playerScore: score,
				});
				return true;
			}, 1000);
		}
	}

	/**
	 * @description Computes player score of the game.
	 * @returns {int} Computed score.
	 */
	computePlayerScore() {
		let len = this.state.playerCards.length;
		return Math.ceil((13 - len) * (100 / 13));
	}

	/**
	 * @description Displays text when players choose to pass the current turn.
	 */
	displayPass() {
		let field = this.state.turn;
		let node = document.createElement('div');
		node.append(document.createTextNode('Pass'));
		node.setAttribute('class', 'gameplayfield-text');
		document.getElementById(field).append(node);
		setTimeout(() => {
			document.getElementById(field).removeChild(node);
		}, 1000);
	}

	render() {
		if (this.state.rules) {
			return (
				<div>
					<div className="game-container">
						<div className="window-container">
							<div className="window">
								<div className="rul
```

```
                                                                                     <h4 classNa
heading">
                                                                                             <sp
className="rules-heading-span">Rules</span>
                                                                                     </h4>
                                                                             </div>
                                                                             <div className="rul
details">
                                                                                     <ul classNa
details">
                                                                                             <li
A > K > Q > J > 10 > 9 > 8 > 7 > 6 > 5 > 4 > 3 </li>
                                                                                             <li
Spades > hearts > clubs > diamonds</li>
                                                                                             <li
combinations: single, pairs, triples, five-cards</li>
                                                                                             <li

combination can only be beaten by a better combination with the same

of cards.
                                                                                             </l
                                                                                             <li
                                                                                             <li
consists of five cards of consecutive rank with mixed suits.</li>
                                                                                             <li
consists of any five cards of the same suit.</li>
                                                                                             <li

House consists of three cards of one rank and two of another rank
                                                                                             </l
                                                                                             <li
made up of all four cards of one rank, plus any fifth card</li>
                                                                                             <li
Flush consists of five consecutive cards of the same suit.</li>
                                                                                     </ul>
                                                                             </div>
                                                                             <div className="rul
                                                                                     <img
,
className="start-button"
,
src={startButton}
,                                                                                            onC
{this.startGame}
,                                                                                            alt
button"
,                                                                                    />
                                                                             </div>
                                                                             <div>3XA3 G06</div>
                                                                     </div>
                                                             </div>
                                                     </div>
                                             </div>
                                     );
                             } else {
                                     return (
                                             <div>
                                                     <div className="game-container">
                                                             {this.state.gameOver && <div
className="window-container">
                                                                     <div className="window">
```

```
                                                                <div className="gam
container">
                                                                        <div>Game C
                                                                        <div>Score
{this.state.playerScore}</div>
                                                                        <button
,                                                                               id=
button"
,                                                                               dis
{false}
,
className="playagain-button"
,                                                                               onC
{this.resetGame}
,                                                                       >
                                                                                Pla
                                                                        </button>
                                                                </div>
                                                        </div>
                                                </div>}
                                                <div className="game-opponent">
                                                        <img src={booIcon} alt="cha
className="top-icon" />
                                                        <img src={luigiIcon} alt="c
className="opponent-icon" />
                                                        <div className="game-left">
                                                                <Deck
,                                                                       class="oppo
container-left"
,                                                                       cardClass="
side"
,                                                                       cards=
{this.state.leftCards}
,                                                               ></Deck>
                                                        </div>
                                                        <div className="game-middle
                                                                <Deck
,                                                                       class="oppo
container-top"
,                                                                       cardClass="
top"
,                                                                       cards=
{this.state.topCards}
,                                                               ></Deck>
                                                                <GameplayField
,                                                                       player=
{this.state.playerField}
,                                                                       right=
{this.state.rightField}
,
left={this.state.leftField}
,
top={this.state.topField}
,                                                                       playerFiel
{this.state.playerFieldText}
,                                                               ></GameplayField>
                                                        </div>
                                                        <div className="game-right"
                                                                <Timer
,                                                                       initialMinu
{this.state.minutes}
```

```
,                                                                       initialSecc
{this.state.seconds}
,                                                                       onTimer=
{this.handleTimer}
,                                                               />
                                                                <Deck
,                                                                       class="oppc
container-right"
,                                                                       cardClass="
side"
,                                                                       cards=
{this.state.rightCards}
,                                                               ></Deck>
                                                        </div>
                                                        <img src={peachIcon} alt="c
className="opponent-icon" />
                                                </div>
                                                <Player
,                                                       cards={this.state.playerCar
,                                                       playerTurn={this.state.turr
'player'}
,                                                       freeMove={this.state.freeMc
,                                                       playCards={this.handlePlaye
,                                                       passTurn={this.handlePlayer
,                                                       turn={this.state.turn}
,                                                       typeSort={this.typeSort}
,                                                       suitSort={this.suitSort}
,                                                       gameOver={this.state.gameOv
,                                                       playerScore={this.state.pla
,                                               ></Player>
                                        </div>
                                </div>
                        );
                }
        }
}

export default Game;
```

```jsx
/**
 * @file GameplayField.jsx
 * @description This file exports a GameplayField react component.
 * @author Jiaxin Tang
 * @version Latest edition on April 11, 2021
 */

import React from 'react';
import Card from './Card.jsx'



/**
 * @description This react arrow function arranges the field for the cards that players dealt in a big two game.
 * @param {*} props Props from parent component.
 * @return a div element displaying the field
 */
const GameplayField = (props) => {

    return (
        <div className="gameplayfield-container">
            <div className="gameplayfield-section section-top" id="top">
                {props.top.map((card, i) => {return (<Card class="field-card" key={i} card={card} user="field" />)})}
            </div>
            <div className="gameplayfield-section">
                <div className="left-field" id="left">
                    {props.left.map((card, i) => {return (<Card class="field-card" key={i} card={card} user="field" />)})}
                </div>
                <div className="right-field" id="right">
                    {props.right.map((card, i) => {return (<Card class="field-card" key={i} card={card} user="field" />)})}
                </div>

            </div>
            <div className="gameplayfield-section section-top" id="player">
                <div className="gameplayfield-player">
                    {props.player.map((card, i) => {return (<Card key={i} card={card} class="field-card" user="field" />)})}
                </div>
                <div className="gameplayfield-text">{props.playerFieldText}</div>
                </div>
            </div>
        )

}

GameplayField.defaultProps = {
    props:{
        playerFieldText: "",
    }
}

/**
 * @exports GameplayField
 */
export default GameplayField
```

```
/**
 * @file Card.jsx
 * @description This file exports a Card react component.
 * @author Manyi Cheng
 */
import React from 'react';
/**
 * @description This react arrow function represents a Card component in a BigTwo game.
 * @param {*} props Props from parent component.
 * @returns React div HTML element displaying the card.
 */
const Card = (props) => {
        const path = props.card.imagePath;
        const images = importAll(require.context('../res/Asset', false, /\.png$/));
        /**
         * Imports all images from the parameter path r
         * @function importAll
         * @param {*} r Indicates the required path to the card image folder.
         * @returns List of json objects containing the images.
         */
        function importAll(r) {
                let images = {};
                r.keys().forEach((item) => {
                        images[item.replace('./', '')] = r(item);
                });
                return images;
        }

        if (props.user === 'opponent') {
                return (
                        <div>
                                <img className={props.class} alt="opponent-card" src={images['Back.png']} />
                        </div>
                );
        } else if (props.user === 'player') {
                const classname = props.selected ? 'selectedcard' : '';
                return (
                        <div>
                                <img
,                                       onClick={() => props.selectCard(props.card)}
                                        className={'card ' + classname}
                                        alt="player-card"
                                        src={images[path]}
                                />
                        </div>
                );
        } else {
                return <img className={props.class + ' flip-in-ver-left'} alt="field-card" src={images[path]} />;
        }
};

Card.defaultProps = {
        props: {
                user: '',
        },
};

export default Card;
```

```
/**
 * @file Deck.jsx
 * @description This file exports a Deck react component.
 * @author Jiaxin Tang
 * @version Latest edition on April 11, 2021
 */

import Card from './Card.jsx'
import React from 'react'

/**
 * @description This react arrow function represents a deck component in a big two game.
 * @param {*} props Props from parent component.
 * @return a div element displaying the deck
 */
const Deck = (props) => {
    if(props.cards){
        return (
            <div className={"opponent-container " + props.class}>
                {props.cards.map((card, i) => <Card class={props.cardClass} user="opponent" key={i} card={card}/>
                )}
            </div>
        )
    }
}

Deck.defaultProps = {
    props:{
        cardClass: "",
    }
}

/**
 * @exports Deck
 */
export default Deck
```

```jsx
/**
 * @file Player.jsx
 * @description React extension javascript that exports a Player react component.
 * @author Manyi Cheng
 */
import React, { useState } from 'react';
import marioImg from '../res/mario.png';
import Card from './Card.jsx';

/**
 * @description This react arrow function represents a Player component in a BigTwo game.
 * @param {*} props Props from parent component.
 * @returns React div HTML element displaying the player component
 */
const Player = (props) => {
        const [selectedCards, setSelectCard] = useState([]);

        /**
         * @description This react arrow function selects a card upon click event.
         * @param {*} card The clicked card
         */
        const selectCard = (card) => {
                let newSelectedCards = [];
                if (selectedCards.includes(card)) {
                        const index = selectedCards.indexOf(card);
                        newSelectedCards = [...selectedCards.slice(0, index), ...selectedCards.slice(index + 1)];
                } else {
                        newSelectedCards = selectedCards.concat([card]);
                }
                setSelectCard(newSelectedCards);
        };

        /**
         * @description This react arrow function handles player click upon user clicking deal button.
         * @param {*} e Click event
         */
        const handleDeal = (e) => {
                e.preventDefault();
                if (props.playerTurn) {
                        if (props.playCards(selectedCards)) {
                                setSelectCard([]);
                        }
                        document.getElementById('playbtn').disabled = true;
                        setTimeout(() => {
                                if (document.getElementById('playbtn')) document.getElementById('playbtn').disabled = false;
                        }, 1500);
                }
        };

        /**
         * @description This react arrow function handles player click upon user clicking pass button.
         * @param {*} e Click event
         */
        const handlePass = (e) => {
                e.preventDefault();
                if (props.playerTurn) {
                        props.passTurn();
                        document.getElementById('passbtn').disabled = true;
                        setTimeout(() => {
                                if (document.getElementById('passbtn')) document.getElementById('passbtn').disabled = false;
                        }, 1500);
                }
        };

        /**
         * @description This react arrow function sorts the player deck based on type order in increasing order upon user clicking type button.
         */
        const handleTypeSort = () => {
                props.typeSort();
        };

        /**
         * @description This react arrow function sorts the player deck based on suit order in increasing order upon user clicking suit button.
         */
        const handleSuitSort = () => {
                props.suitSort();
        };

        let actionButton = props.playerTurn ? '' : 'disabled-button';
        let freeMoveButton = !props.freeMove ? '' : 'disabled-button';
        return (
                <div className="player-container">
            <img className = "player-icon" alt = "character" src = {marioImg}/>
                        {props.cards &&
                                props.cards.map((card, i) => {
                                        let selected = selectedCards.includes(card);
                                        return <Card key={i} card={card} user="player" selectCard={selectCard} selected={selected} />;
                                })}
                        {!props.gameOver && (
                                <div className="player-action">
                                        <button id="playbtn" className={'player-button ' + actionButton} onClick={handleDeal}>
                                                Deal
                                        </button>
                                        <button
,                                               id="passbtn"
,                                               className={'player-button ' + actionButton + ' ' + freeMoveButton}
,                                               onClick={handlePass}
,                                       >
                                                Pass
                                        </button>
                                        <button className="player-button" onClick={handleTypeSort}>
                                                Type
                                        </button>
                                        <button className="player-button" onClick={handleSuitSort}>
```

```
                                                Suit
                                        </button>
                                </div>
                        )}
                </div>
        );
};

export default Player;
```

```javascript
/**
 * @file Timer.js
 * @description This file generates a timer for the game.
 * @author Manyi Cheng
 * @version Latest edition on April 10, 2021
 */
import React, { useState, useEffect } from 'react';


/**
 * @function Timer
 * @param {*} props
 * @returns A timmer that counts down from 10 minutes on the upper right corner of the web page during the game
 */
const Timer = (props) => {
        const { initialMinutes = 0, initialSeconds = 0 } = props;
        const [minutes, setMinutes] = useState(initialMinutes);
        const [seconds, setSeconds] = useState(initialSeconds);

        useEffect(() => {
                let myInterval = setInterval(() => {
                        if (seconds > 0) {
                                setSeconds(seconds - 1);
                        }
                        if (seconds === 0) {
                                if (minutes === 0) {
                                        clearInterval(myInterval);
                                } else {
                                        setMinutes(minutes - 1);
                                        setSeconds(59);
                                }
                        }
                }, 1000);
                return () => {
                        clearInterval(myInterval);
                };
        }, [minutes, seconds]);

    useEffect(() =>{
        if(minutes === 0 && seconds === 0){
            console.log("times up")
            props.onTimer()
        }
    }, [minutes, seconds]);

        return (
                <div className = "timer-container" >
                        {minutes === 0 && seconds === 0 ? null: (
                                <div>
                                        {' '}
                                        {minutes}:{seconds < 10 ? `0${seconds}` : seconds}
                                </div>
                        )}
                </div>
        );
};

```

```
/**
 * @exports Timer
 */
export default Timer;
```

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

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

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

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

    }

    return selectedCards
}

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

/**
 * @function BotFreeTurn
 * @description When all other players pass, and this playerBot will deal out the smallest cards combo in the privilage of
 * five cards -> pairs -> single card
 * @param {card[]} cards
 * @returns {card[]} a list of smallest cards combo it can deal out in the privilage of five -> pair -> single
 */
export function BotFreeTurn(cards) {
    Rules.sortCardsValue(cards)

    var selectedCards = getAllFiveCards(cards)

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

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

    return [cards[0]]
}

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

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

    return null
}

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

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

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

    return null
}

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

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

    return null
}

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

    var validCombos = []

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

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

    return validCombos

}

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

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

    return pairs
}
```

```javascript
/**
 * @file Rules.js
 * @description This file contains rules of BigTwo game.
 * @author Jiaxin Tang
 * @version Latest edition on April 11, 2021
 */

const suitsPath = ["Diamonds", "Clubs", "Hearts", "Spades"]
const valuesPath = ["", "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King"]
const suits = ["D", "C", "H", "S"]
const SuiteVal = [1, 2, 3, 4]
const type = ["", "A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]

/**
 * @function newDeck
 * @description A function that generates a deck of 52 cards, and rearranges the order of cards in the deck
 * @returns {card[]} deck - with cards in random order
 */
export function newDeck() {
    let deck = []

    for (let i = 1; i < 14; i++) {
        for (let j = 0; j < 4; j++) {
            let value = (i === 1) ? 14 : (i === 2) ? 15 : i
            let imagePath = "NAP-01_"+ suitsPath[j] + "_"+ valuesPath[i] + ".png"
            let card = {
                type: type[i],
                suit: suits[j],
                suiteVal : SuiteVal[j],
                value: value,
                imagePath: imagePath
            }
            deck.push(card)
        }
    }

    return shuffle(deck)
}

/**
 * @function shuffle
 * @description A function that rearranges the order of cards in the given deck
 * @param {card[]} deck - a list of cards
 * @returns {card[]} deck - with cards in random order
 */

function shuffle(deck) {
    var temp, i, j;
    for (i = deck.length - 1; i > 0; i--) {
        j = Math.floor(Math.random() * (i + 1));
        temp = deck[i];
        deck[i] = deck[j];
        deck[j] = temp;
    }
    return deck;
}
```

```javascript

/**
 * @function isValidStartingPlay
 * @description A function that checks if the current play is valid starting play
 * @param {card[]} cards - the cards that current player has
 * @returns {boolean} = true if cards contain Diamond 3
 */
export function isValidStartingPlay(cards) {
    let containsThreeOfDiamonds

    cards.forEach((card) => {
        if (card.suit === "D" && card.value === 3) containsThreeOfDiamonds = true
    })

    if (containsThreeOfDiamonds) {
        return isValidPlay(cards)
    } else {
        return false
    }
}

/**
 * @function isValidSPlay
 * @description A function that checks if the current play is valid play
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if is valid play
 */

export function isValidPlay(cards) {
    if (cards == null) return false
    sortCardsValue(cards)

    return isValidSingle(cards) || isValidPair(cards) ||
isValidFiveCardPlay(cards)
}


/**
 * @function isValidSingle
 * @description A function that checks if the current play is valid single play
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards contain a single card
 */
export function isValidSingle(cards) {
    return cards.length === 1
}

/**
 * @function isValidPair
 * @description A function that checks if the current play is valid pair
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards is a valid pair
 */
export function isValidPair(cards) {
    return cards.length === 2 && cards[0].type === cards[1].type
}

/**
 * @function isValidFiveCardPlay
 * @description A function that checks if the current play is valid five card
play
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards is a valid combination of five cards
 */
export function isValidFiveCardPlay(cards) {
    if (cards.length !== 5) return false

    return isValidStraight(cards) || isValidFlush(cards) ||
isValidFullHouse(cards) || isValidFourOfaKind(cards)
}

/**
 * @function isValidStraight
 * @description A function that checks if the current play is valid straight
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards is a valid straight
 */
function isValidStraight(cards) {
    if(cards.length !== 5)
        return false
    //12345
    sortCardsValue(cards)
    if(cards[0].value === 14){
        if(cards[1].value === 15 && cards[2].value === 3 &&
            cards[3].value === 4 && cards[4].value === 5 )
                return true
        else
            return false
    }
    //23456
    if(cards[0].value === 15){
        if(cards[1].value === 3 && cards[2].value === 4 &&
            cards[3].value === 5 && cards[4].value === 6 )
                return true
        else
            return false
    }
    var flag = true
    for(var i = 0; i < 4; i++){
        if((cards[i].value + 1) !== cards[i+1].value){
            flag = false
            return flag
        }
    }
    return flag
}

/**
 * @function isValidFlush
 * @description A function that checks if the current play is valid flush
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards is a valid flush
 */
function isValidFlush(cards) {
    if(cards.length !== 5)
        return false
    var flag = true
    for(var i = 1; i < 5; i++){
        if(cards[i].suiteVal !== cards[0].suiteVal){
            flag = false
            return flag
        }
    }
    return flag
}

/**
 * @function isValidFullHouse
 * @description A function that checks if the current play is valid fullhouse
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards is a valid fullhouse
 */
function isValidFullHouse(cards) {
    if(cards.length !== 5)
        return false
    sortCardsValue(cards)
    if(cards[0].value === cards[1].value && cards[0].value === cards[2].value &&
        cards[3].value === cards[4].value)
        return true
    if(cards[0].value === cards[1].value && cards[2].value === cards[3].value &&
        cards[2].value === cards[4].value)
        return true
    return false}

/**
 * @function isValidFourOfaKind
 * @description A function that checks if the current play is valid four of a
kind
 * @param {card[]} cards - the cards that current player selects
 * @returns {boolean} - true if cards is a valid four of a kind
 */
function isValidFourOfaKind(cards) {
     if(cards.length !== 5)
        return false
    sortCardsValue(cards)
    if(cards[0].value === cards[1].value && cards[0].value === cards[2].value &&
        cards[0].value === cards[3].value)
        return true
    if(cards[4].value === cards[1].value && cards[4].value === cards[2].value &&
        cards[4].value === cards[3].value)
        return true
    return false
}

/**
 * @function isStrongerPlay
 * @description A function that checks if the current play is stronger than last
play
 * @param {card[]} last - the cards that the last player plays
 * @param {card[]} select - the cards that current player selects
 * @returns {boolean} - true if the select play is stronger than last play
 */
export function isStrongerPlay(last, select) {
    var n = select.length
    if(n !== last.length)
        return false
    switch(n){
```

```javascript
        case 1: return isStrongerSingle(last, select);
        case 2: return isStrongerPair(last, select);
        case 5: return isStrongerFive(last, select);
        default:
            return false
    }
}

/**
 * @function isStrongerSingle
 * @description A function that checks if the current single is stronger than last single
 * @param {card[]} last - the cards that the last player plays
 * @param {card[]} select - the cards that current player selects
 * @returns {boolean} - true if the select play is stronger than last play
 */
export function isStrongerSingle(last, select){
    if(select[0] && last[0]){
        if(select[0].value > last[0].value)
            return true
        if(select[0].value === last[0].value && select[0].suiteVal > last[0].suiteVal)
            return true
    }else if(select[0] && !last[0]){
        if(select[0].value > last.value)
            return true
        if(select[0].value === last.value && select[0].suiteVal > last.suiteVal)
            return true
    }else if(!select[0] && !last[0]){
        if(select.value > last.value)
            return true
        if(select.value === last.value && select.suiteVal > last.suiteVal)
            return true
    }else if(!select[0] && last[0]){
        if(select.value > last[0].value)
            return true
        if(select.value === last[0].value && select.suiteVal > last[0].suiteVal)
            return true
    }
    return false
}

/**
 * @function isStrongerPair
 * @description A function that checks if the current pair is stronger than last pair
 * @param {card[]} last - the cards that the last player plays
 * @param {card[]} select - the cards that current player selects
 * @returns {boolean} - true if the select play is stronger than last play
 */
export function isStrongerPair(last, select){
    if(!isValidPair(select))
        return false
    if(select[0].value > last[0].value)
        return true
    sortCardsSuit(select)
    sortCardsSuit(last)
    if(select[0].value === last[0].value && select[1].suiteVal > last[1].suiteVal)
```

```
        return true
    return false
}

/**
 * @function isStrongerFive
 * @description A function that checks if the current five card play is stronger than last five card play
 * @param {card[]} last - the cards that the last player plays
 * @param {card[]} select - the cards that current player selects
 * @returns {boolean} - true if the select play is stronger than last play
 */
export function isStrongerFive(last, select){
    if(isValidFourOfaKind(select) && isValidFullHouse(last))
        return true
    if(isValidFourOfaKind(select) && isValidFlush(last))
        return true
    if(isValidFourOfaKind(select) && isValidStraight(last))
        return true
    if(isValidFullHouse(select) && isValidFlush(last))
        return true
    if(isValidFullHouse(select) && isValidStraight(last))
        return true
    if(isValidFlush(select) && isValidStraight(last))
        return true
    if(isValidStraight(select) && isValidStraight(last)){
        sortCardsValue(select)
        sortCardsValue(last)
        if(select[4].value > last[4].value)
            return true
        else
            return false
    }
    if(isValidFlush(select) && isValidFlush(last)){
        sortCardsValue(select)
        sortCardsValue(last)
        if(select[0].suiteVal > last[0].suiteVal)
            return true
        if(select[0].suiteVal === last[0].suiteVal && select[4].value > last[4].value)
            return true
        return false
    }
    if(isValidFullHouse(select) && isValidFullHouse(last)){
        sortCardsValue(select)
        sortCardsValue(last)
        if(select[3].value > last[3].value)
            return true
        return false
    }
    if(isValidFourOfaKind(select) && isValidFourOfaKind(last)){
        sortCardsValue(select)
        sortCardsValue(last)
        if(select[3].value > last[3].value)
            return true
        return false
    }
}

```

```
/**
 * @function setUserCards
 * @description A function that places 13 cards in a deck into a list to be
assigned to a player.
 * @param {card[]} deck - a list of 52 cards in a random order
 * @returns {card[]} userCards - contains 13 cards for a player
 */
export function setUserCards(deck) {
    let userCards = []
    for (let i = 0; i < 13; i++) {
        userCards.push(deck.pop())
    }
    return userCards
}

/**
 * @function setFirstTurn
 * @description A function that decides which player plays the first turn.
 * @param {card[]} playerCards - a list of cards that player has
 * @param {card[]} opponentLeftCards - a list of cards that left AI has
 * @param {card[]} opponentTopCards - a list of cards that top AI has
 * @param {card[]} opponentRightCards - a list of cards that right AI has
 * @returns {string} turn - represeting the initial player
 */
export function setFirstTurn(playerCards, opponentLeftCards, opponentTopCards,
opponentRightCards) {
    let turn
    playerCards.forEach((card) => {
        if (card.suit === "D" && card.value === 3) turn = "player"
    })

    opponentLeftCards.forEach((card) => {
        if (card.suit === "D" && card.value === 3) turn = "left"
    })

    opponentTopCards.forEach((card) => {
        if (card.suit === "D" && card.value === 3) turn = "top"
    })

    opponentRightCards.forEach((card) => {
        if (card.suit === "D" && card.value === 3) turn = "right"
    })
    return turn
}

/**
 * @function getSuitValue
 * @description A function that gets the integer value of the corresponding suit.
 * @param {string} suit
 * @returns {int} - integer value related to suit
 */
export function getSuitValue(suit) {
    return (suit === "D") ? 1 : (suit === "C") ? 2 : (suit === "H") ? 3 : 4
}

/**
 * @function sortCardsValue
 * @description A function that sorts the given cards in the number rank order.
 * @param {card[]} cards
 * @returns {card[]} cards - ordered in the number rank
 */
export function sortCardsValue(cards) {
    if (cards == null) return

    cards.sort((a, b) => {
        return a.value - b.value
    })
}

/**
 * @function sortCardsSuit
 * @description A function that sorts the given cards in the suit rank order.
 * @param {card[]} cards
 * @returns {card[]} cards - ordered in the suit rank
 */
export function sortCardsSuit(cards) {
    if (cards == null) return

    cards.sort((a, b) => {
        return a.suiteVal - b.suiteVal
    })
}
```