React Chess Board v0.4.7

Yet another Javascript chessboard.

Intended to be used as a pluggable component in projects constructed using React.js and involving chess games.

Install

Through npm

```
cd your-react-project-folder
npm install --save next-chess-board
```

or through yarn

```
cd your-react-project-folder
yarn add next-chess-board
```

You are ready to use ChessBoard component in your react.js project, for instance in your `index.js`:

```
import ChessBoard from 'next-chess-board'

export default () => (
  <div>
    <Chessboard size={400} moveValidator={true} />
  </div>
)
```

Motivation

In spite of the fact that there are many fine alternatives around, it’s somewhat difficult to find one that offers a good mix of functionality and aesthetics. This work began with this idea in mind, and the result is a web chessboard that features:

1. Seven chess figures set to choose from .

2. Four differnet board backgrounds.

3. Four different working modes: **setup**, **analysis**, **play** and **view**.

4. A smart promotion panel that detects the promotion square and sits directly on it, sparing the player a long movement with the mouse to choose the promoting figure. It also detects color of the figures and background of the the square, “dressing” itself with those options, providing a clear visual clue of the action that’s undergoing.

5. Importing/exporting of FEN positions and PGN games.

6. Exporting the board to image (as of now, PNG format), using the function `drawDiagram`

7. Loads of other minor but useful features.

API

Main recomendation here is the old, wise *Look at the source, Luke*

The client pages, users of the component, may be found in the `/pages` directory of the source. There almost every feature this component provides is used, and the code is pretty self explanatory. Looking at

them (and reusing code!) is strongly recommended.

Anyway the formal description follows.

Static properties and methods

```
    static Modes = {
      MODE_SETUP: 'MODE_SETUP',
      MODE_ANALYSIS: 'MODE_ANALYSIS',
      MODE_VIEW: 'MODE_VIEW',
      MODE_PLAY: 'MODE_PLAY'
    }
```

Define the 4 modes that ChessBoard can adopt.

`Chessboard.chessSets` Exposes an array of image sets in data format, which is used internally by the board to draw its images. This means that the user of the component doesn't have to care about figure images location, since they are embedded within the component itself. Exposing them publicly through this property allows their usage outside of the chessboard, for example:

```
  <img src={ChessBoard.chessSets.alt1.B} title="White Bis
hop from alt1 set" />
```

```
static Messages = {
      CHECK_MATE: {en: 'Checkmate', es: 'Jaque Mate'},
```

```
      CHECK: {en: 'Check', es: 'Jaque'},
      STALE_MATE: {en: 'Stalemate draw', es: 'Tablas por
mate ahogado'},
      INSUFFICIENT_MATERIAL: {en: 'Draw for insufficient
material', es: 'Tablas por material insuficiente'},
      WRONG_MOVE: {en: 'Wrong move', es: 'Movimiento inco
rrecto'},
      ERROR_LOAD_FEN: {en: "Could not load position", es:
 "No se pudo cargar la posición"},
      ERROR_PREV_POS: {en: "Attempt to move from a non la
st position", es: "Intento de mover desde una posición qu
e no es la última"},
      ERROR_MOVE_ARGS: {en: "Move called with wrong numbe
r of arguments", es: "Función 'move' invocada con número
incorrecto de argumentos"},
      ERROR_MOVE_TURN: {en: "Attempt to move the wrong co
lor", es: "Intento de mover el color equivocadd"},
      ERROR_CANT_PROCESS_SAN: {en: "Can't process standar
d algebraic notation (SAN) move without a move validator"
, es: "No se puede procesar movimiento en notación algebr
aica estandard (SAN) sin un validador de movimientos"},
      ERROR_WRONG_MOVE: {en: 'Wrong move', es: 'Movida er
rónea'}
    }
```

These are the messages that ChessBoard internally uses to provide feedback to its client page. As it shows, it’s an array of objects, each

one of them providing a string message in English and Spanish. It’s up to the client page to choose the proper one according to its settings. Example of this can be seen in the message handliers implemented in `index.js` which provides for the home page at the demo site. It can be found under the `/pages` directory of the source. It’s strongly recommended to have a thorough look at it, as it shows almost all the features depicted here.

```
    static Figurines = {
      p: {codePoint: '0x265f',  html: '&#9823;'},
      n: {codePoint: '0x265e',  html: '&#9822;'},
      b: {codePoint: '0x265d',  html: '&#9821;'},
      r: {codePoint: '0x265c',  html: '&#9820;'},
      q: {codePoint: '0x265b',  html: '&#9819;'},
      k: {codePoint: '0x265a',  html: '&#9818;'},
      P: {codePoint: '0x2659',  html: '&#9817;'},
      N: {codePoint: '0x2658',  html: '&#9816;'},
      B: {codePoint: '0x2657',  html: '&#9815;'},
      R: {codePoint: '0x2656',  html: '&#9814;'},
      Q: {codePoint: '0x2655',  html: '&#9813;'},
      K: {codePoint: '0x2654',  html: '&#9812;'}
    }
```

Codes for figurines in `utf8` and `html`. Reserved for future use.

```
    static partPosition = (pos) => ChessBoard.partition([
...pos]).map(r => r.join('')).join('/')
    static compressPosition = (pos) => ChessBoard.partPos
ition(pos).replace(/0+/g, (m => m.length.toString()))
    static expandPosition = (pos) => pos.replace(/\//g, '
').replace(/[1-8]/g, (d) => ChessBoard.range(0, parseInt(
d)).map(i => '0').join(''))

    static sqBgLabels = ['Blue', 'Brown', 'Acqua', 'Maroo
n']
    static lightSqBgs = ['#add8e6', '#f0d9b5', '#dfdfdf',
 '#FFF2D7']
    static darkSqBgs = ['#6495ed', '#b58863', '#56b6e2',
"#B2535B"]
    static selectedSqBg = '#bfd'

    static emptyPosition = ChessBoard.range(0, 64).map(i
=>'0').join('')
    static defaultPosition = 'rnbqkbnrpppppppp0000000000
0000000000000000000000PPPPPPPPRNBQKBNR'
    static sicilianPosition = 'rnbqkbnrpp0ppppp0000000000
p000000000P00000000000PPPP0PPPRNBQKBNR'

    static emptyFen = '8/8/8/8/8/8/8/8 w - - 0 1'
    static defaultFen = 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPP
PP/RNBQKBNR w KQkq - 0 1'
    static sicilianFen = 'rnbqkbnr/pp1ppppp/8/2p5/4P3/8/P
```

```
PPP1PPP/RNBQKBNR w KQkq - 0 1'

    static pgnTagLineRE = /^\s*\[\s*(.+?)\s+"(.+?)"\s*\]\
s*$/
    static sanRE = /(?:(^0-0-0|^O-O-O)|(^0-0|^O-O)|(?:^([
a-h])(?:([1-8])|(?:x([a-h][1-8])))(?:=?([NBRQ]))?)|(?:^([
NBRQK])([a-h])?([1-8])?(x)?([a-h][1-8])))(?:(\+)|(#)|(\+\
+))?$/

    static defaultSettings = {
      size: 400,
      flipped: false,
      chessSet: 'default',
      currentPosition: 0,
      positions: [ChessBoard.defaultFen],
      lightSqsBg: ChessBoard.lightSqBgs[0],
      darkSqsBg: ChessBoard.darkSqBgs[0],
      selectedSqBg: ChessBoard.selectedSqBg,
      movements:  [],
      isCrowning: false,
      showNotation: true,
      whitePlayer: 'White Player',
      blackPlayer: 'Black Player',
      lang: 'en'
    }
```

All of the above should be pretty self explanatory

Instance methods

To be invoked to perform a particular action on a selected instance, for example, to flip the board:

```
import {Component} from 'react'
import ChessBoard from 'next-chess-board'

export default class BoardPage extend Component {
   render () {
     return (
        <div>
            <ChessBoard ref="board1" size={400} moveValid
ator={true} />
            <button onClick={(evt) => this.refs.board1.fl
ip()}>
              Flip it!
            </button>
        </div>
    )
  }
}
```

List of available instance methods

- setup() *Puts the board in setup mode, making the setup panel visible and hiding the notation panel*

- analyse() *Puts the board in analyse mode, in which the user cand od basically anything that involves a legal chess move, including undoing*

- view() *Puts the board in view mode, in which the user can browse the game, but isn't allowed to alter the game by any mean*

- play() *Puts the board in game mode, which allows him to move when it's her/his turn*

- flip() *Flips/Unflips the board. Shortcut: this function* ***can be invoked by double-clicking the board*** *in any mode*

- setSize(newSize) *Sets board size to* ***newsize***

- drawDiagram(context, ctxSize = this.state.size) *Copies board position to a canvas where it may be treated (copied, saved) as an image. Second argument corresponds to canvas size. If it is omitted defaults to board size, which implies that canvas may remain partially unused if it is larger than board, or that board image won't fit in entirely in the opposite case. Recommended setting is passing in canvas size*

- doScroll() *Auxiliary function to scroll notation panel to the bottom. Imperative (smells)*

- useSet(set) *Defines shape for figures being shown. Available values for **set** are: 'default', 'alt1', 'modern', 'fantasy', 'spatial', 'eyes' and 'veronika'. Feel free to roll your own*

- useSquares(n) *Defines color for light and dark squares on the board. There are currently 4 schemas available (0, 1, 2, 3, 4) a.k.a. ('Blue', 'Brown', 'Acqua', 'Maroon'). Take on account, however, that the function must be invoked with a numerical argument. For instance,* `board.useSquares(1)` *gives you these board colors*

- goto(n) *Sets the board position to **n** and consequently redraws

it to reflect the new position, where **n** should be a number between 0 and `positions.length - 1`. But regardless, ChessBoard won't allow going off boundaries.

- previous()
- next()
- last()
- first() *4 convenience methods to move to previous, next, last and first position respectively. They call* `goto()` *under the hood.*
- empty() *As the name implies, it just empties the chessboard*
- reset() *Resets the board, cleaning the positions and movements arrays, and sets it to* `ChessBoard.defaultFen`
- loadFen(fen) *Reads a string in Forsyth Edwards notation and attemps to make it the current first position*
- loadPgn(pgn) *Reads a string in Portable Game Notation and attempts to load its data and all the moves in the current game. Requires* `moveValidator={true}` *in the properties for this to work*
- takeback() *Undoes last move. Not to be used in play mode.*

- move(...args) *the essence of the whole thing, performing moves, where **...args** is an array of arguments with a number that can be:*
  a. 1 = move in Standard Algebraic Notation. moveValidator must be true for this to work
  b. 3 = sqFrom, sqTo, figure. This is the way ChessBoard uses to perform moves in response to user actions
  c. 4 = sqFrom, sqTo, figure, promotion. Same as previous, with promotion special case. sqFrom and sqTo are numerical, while figure and promotion are string representation of the figure being moved or promoted to respectively

- isFlipped() *Just lets the client know if the board is flipped (just in case the client is not human and/or can't see :-)*

- setHeader(k, v) *Sets game header **k** to value **v**. For example:* `game.setHeader('Place', 'Rosario')`

- setDate(date)

- setDate(date) *Shortcut for* `board.setHeader('Date', date)`

- setPlayer(color, name) *Shortcut for* `board.setHeader('White|Black', name)`

- getPgnText() *Retrieves current game in PGN format*

Events

The chessboard uses its own event system in order to communicate the components/pages that host it when something important has happened on the board, be it related to the game itself or otherwise.

List of events that ChessBoard publishes

- CHECK_MATE
- CHECK
- DRAW
- STALE_MATE
- INSUFFICIENT_MATERIAL
- MOVE
- ERROR
- CHANGE
- FLIP

List of Messages associated to ChessBoard events

- CHECK_MATE
- CHECK
- STALE_MATE
- INSUFFICIENT_MATERIAL
- WRONG_MOVE
- ERROR_LOAD_FEN
- ERROR_PREV_POS
- ERROR_MOVE_ARGS

- ERROR_MOVE_TURN
- ERROR_CANT_PROCESS_SAN
- ERROR_WRONG_MOVE

Each one of the previous messages is an object which - as of now - contains two keys: **en** and **es**, each one of them containing the corresponding message in English and Spanish respectively. It would be nice to add italian, french, german, russian and portuguese to begin with, and to continue with other national languages. That's a task that will be undertaken in the near future, as long as we get some colabs who are up to the task.

Usage of ChessBoard event system

The clients (usually pages that host the board) shall subcribe to the events they're interested in, feeding ChessBoard with the desired event and a callback function that will be invoked when the event is fired/emitted by ChessBoard. In response ChessBoard will return a function that will cancel the subscription upon invocation. A typical scenario is subscribing to the event/s in `componentDidMount`, and unsubsribing in `componentWillUnmount`, like so:

```
componentDidMount () {
  this.unsubsCheckMate = board1.on(ChessBoard.Events.CHEC
K_MATE,
                                   (msg, move, result) =
> {
                                     console.log(`${move
```

```
}   ${msg[this.state.lang]} ${result}`})
                                        })
  }
}
componentWillUnmount () {
  this.unsubsCheckMate() // Cleaning up before leaving
}
```

Looking at the way this is done in `/pages/index.js` of the demo project is strongly encouraged.

> *Todo:*
>
> - Embelishing

Technology

This project was bootstrapped with Create Next App.

Find the most recent version of this guide at here. And check out Next.js repo for the most up-to-date info.

Copyright