|  |  |
| --- | --- |
| *voenmeh* | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** |
| БГТУ.СМК-Ф-4.2-К5-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Факультет |  | О |  | Естественнонаучный |
|  |  | шифр |  | наименование |
| Кафедра |  | О7 |  | Информационные системы и программная инженерия |
|  |  | шифр |  | Наименование |
| Дисциплина |  | Представление знаний в ИС | | |

Лабораторная Работа №3.1

на тему

|  |
| --- |
|  |
| Реализация минимаксного алгоритма |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выполнил студент группы И594  Рахимбердиев Т.А. | | | | |
| Фамилия И.О. | | | | |
| **РУКОВОДИТЕЛЬ** | | | | |
| Рохлин Н.С. | |  |  | |
| Фамилия И.О. Подпись | | | | |
| Оценка |  | | |  |
| «\_\_\_\_» |  | | | 2022 г. |

САНКТ-ПЕТЕРБУРГ

2022 г.

**Цель работы:** Разработать программу, реализующую минимаксный алгоритм.

**Краткое описание работы**

Для разработки программы крестики-нулики, с одной стороны пользователь с иной ai (adaptive intelligence) применен алгоритм минимаксный реализованный на Фреймворке React с использованием функциональных компонентов и хуков. Для стилистической части интерфейса использован css.

При изменении состоянии игры данные выводятся в консоль.

**Минимаксный алгоритм**

Минимакс — это своего рода алгоритм [обратного отслеживания](https://www.geeksforgeeks.org/tag/backtracking/) , который используется в принятии решений и теории игр, чтобы найти оптимальный ход для игрока, предполагая, что ваш противник также играет оптимально. Он широко используется в пошаговых играх для двух игроков, таких как крестики-нолики, нарды, манкала, шахматы и т.д.  
В минимаксе двух игроков называют максимизатором и минимизатором. Максимизатор пытается получить как можно более высокий балл, а **минимизатор** пытается сделать обратное и получить как можно более низкий балл. Каждое состояние доски имеет значение, связанное с ним. В данном состоянии, если максимизатор имеет преимущество, счет доски будет стремиться к некоторому положительному значению. Если минимизатор имеет преимущество в этом состоянии доски, то оно будет иметь тенденцию к некоторому отрицательному значению. Значения доски рассчитываются с помощью некоторых эвристик, которые уникальны для каждого типа игры.

**Пример.**   
Рассмотрим игру, в которой есть 4 конечных состояния, а пути достижения конечного состояния — от корня до 4 листьев идеального бинарного дерева, как показано ниже. Предположим, что вы максимизирующий игрок и у вас есть первый шанс сделать ход, т. е. вы находитесь в корне, а ваш противник на следующем уровне. **Какой ход вы бы сделали как максимизирующий игрок, учитывая, что ваш противник также играет оптимально?**

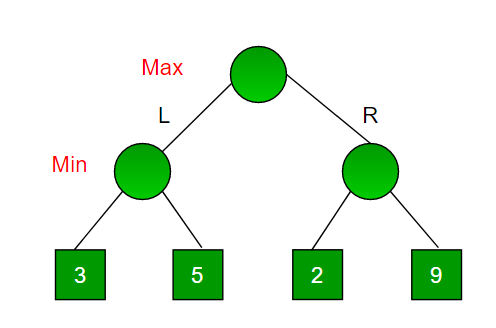
****

Рисунок 1 – Дерево

Поскольку это алгоритм, основанный на возврате, он пробует все возможные ходы, затем возвращается и принимает решение.

* Максимайзер идет ВЛЕВО: теперь очередь минимизаторов. У минимизатора теперь есть выбор между 3 и 5. Будучи минимизатором, он обязательно выберет наименьшее из обоих, то есть 3
* Максимайзер идет ПРАВИЛЬНО: теперь очередь минимизаторов. Теперь у минимизатора есть выбор между 2 и 9. Он выберет 2, так как это наименьшее из двух значений.

Будучи максимизатором, вы бы выбрали большее значение, равное 3. Следовательно, оптимальным ходом для максимизатора является движение ВЛЕВО, а оптимальное значение равно 3.

Теперь дерево игры выглядит следующим образом:

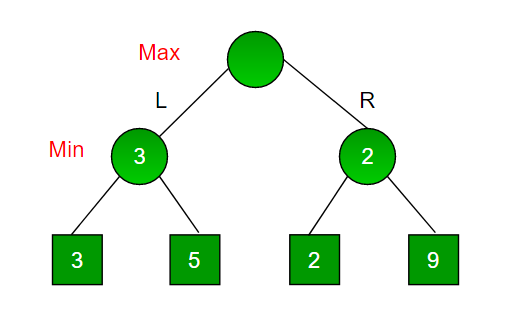


Рисунок 2 – следующее состояние дерева

Приведенное выше дерево показывает два возможных результата, когда максимизатор совершает ходы влево и вправо.

**Примечание.** Несмотря на то, что в правом поддереве есть значение 9, минимизатор никогда не выберет его. Мы всегда должны исходить из того, что наш оппонент играет оптимально.

<script>

 function minimax(depth, nodeIndex, isMax,

            scores, h)

{

    // Terminating condition. i.e leaf node is reached

    if (depth == h)

        return scores[nodeIndex];

    // If current move is maximizer, find the maximum attainable

    // value

    if (isMax)

    return Math.max(minimax(depth+1, nodeIndex\*2, false, scores, h),

            minimax(depth+1, nodeIndex\*2 + 1, false, scores, h));

    // Else (If current move is Minimizer), find the minimum

    // attainable value

    else

        return Math.min(minimax(depth+1, nodeIndex\*2, true, scores, h),

            minimax(depth+1, nodeIndex\*2 + 1, true, scores, h));

}

// A utility function to find Log n in base 2

 function log2(n)

{

return (n==1)? 0 : 1 + log2(n/2);

}

// Driver Code

    // The number of elements in scores must be

    // a power of 2.

    let scores = [3, 5, 2, 9, 12, 5, 23, 23];

    let n = scores.length;

    let h = log2(n);

    let res = minimax(0, 0, true, scores, h);

    document.write( "The optimal value is : "  +res);

</script>

**Отчет**

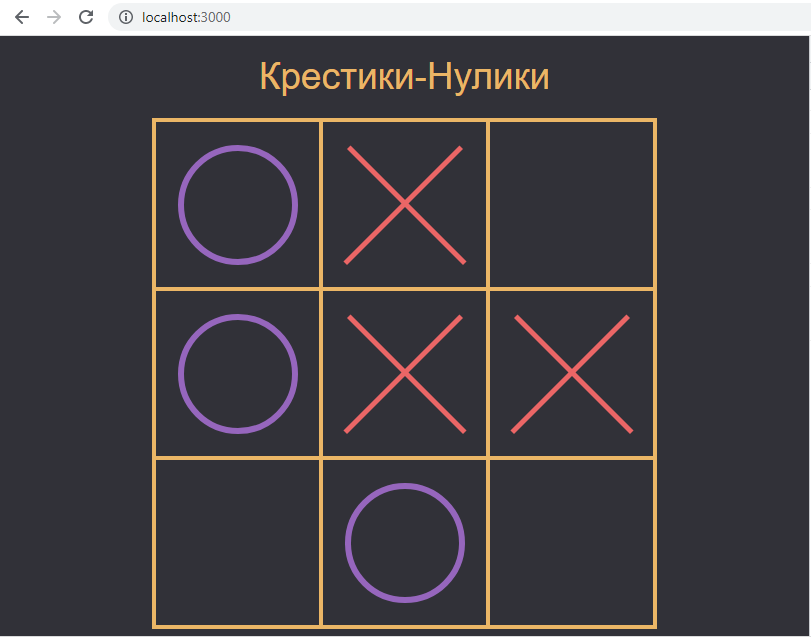
****

Рисунок 1 – Крестики-нулики

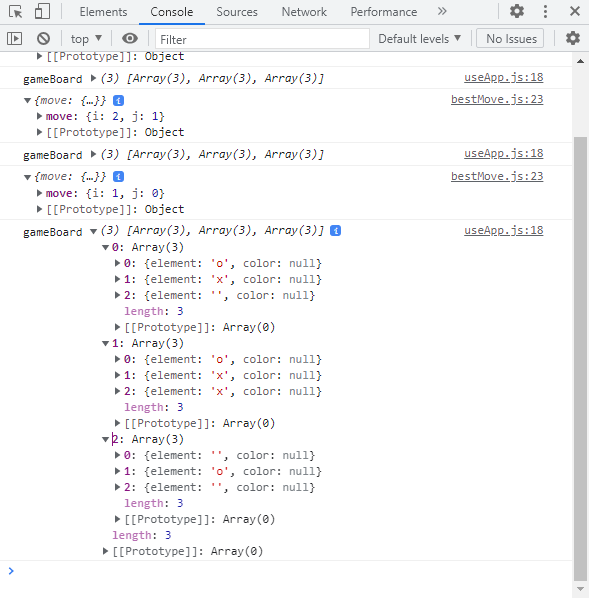
****

Рисунок 2 – промежуточные состояния

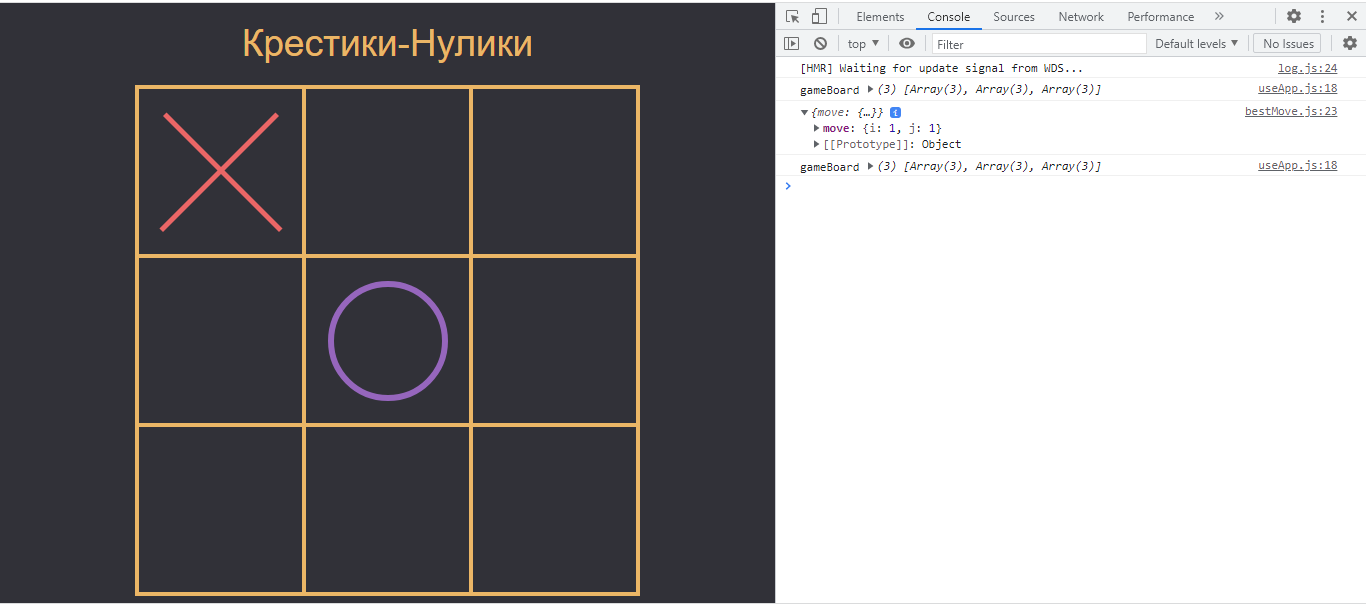
****

Рисунок 3 – выбор первого хода игрока (слева направо)

Игра в “Крестики нулики” состоит из трех рядов, каждая с тремя колонами.

Сторонами являются игрок и “ai” то есть адаптивный интеллект, который использует для выигрыша минимаксный алгоритм. Первым ходит игрок. После проигрыша или ничьи, всплывает кнопка к которому прикреплена функция сыграть заново, что очищает состояние и начинается игра заново.

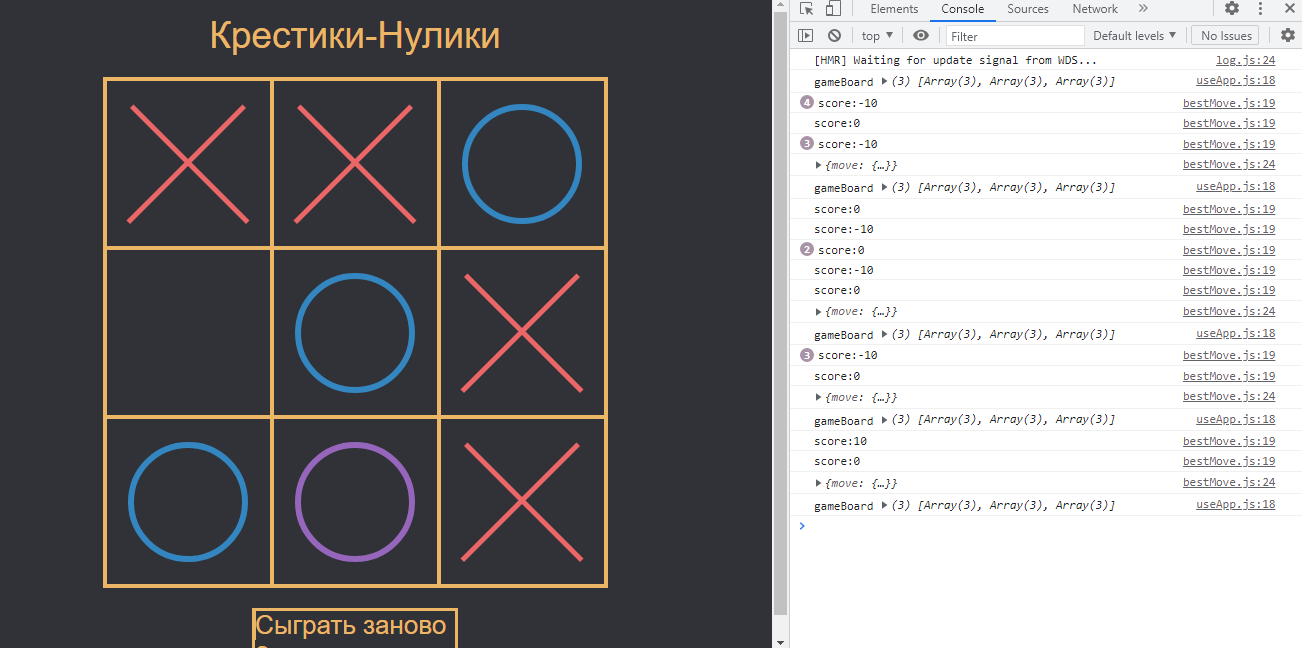
****

Рисунок 4 – выбор хода (справа налево)

**Приложение А**

**index.html**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1" />

    <meta name="theme-color" content="#000000" />

    <meta

      name="description"

      content="Web site created using create-react-app"

    />

    <title>React App</title>

  </head>

  <body>

    <noscript>You need to enable JavaScript to run this app.</noscript>

    <div id="root"></div>

  </body>

</html>

**bestMove.js**

import { minimax } from "./minmax";

export function bestMove(board, ai, player, setGameBoard){

    let bestScore = -Infinity;

    const updatedBoard = [...board];

    let move;

    for(let i = 0; i < 3; i++){

        for(let j = 0; j < 3; j++){

            if(updatedBoard[i][j].element === ''){

                updatedBoard[i][j].element = ai;

                let score = minimax(updatedBoard, 0, false, ai, player);

                updatedBoard[i][j].element = '';

                if(score > bestScore){

                    bestScore = score;

                    move = { i, j}

                }

            }

        }

    }

    console.log({move});

    updatedBoard[move.i][move.j].element = ai;

    setGameBoard(updatedBoard);}

**checkFreeCells.js**

export function checkFreeCells(board){

    let isFreeCellExists = false

    for (let i = 0; i < 3; i++){

        for (let j = 0; j < 3; j++){

            if(board[i][j].element === ''){

                isFreeCellExists = true;

            }

        }

    }

    return isFreeCellExists;

}

**getRandomFreeCellIndexes.js**

export function getRandomFreeCellIndexes(gameMatrix) {

  const freeCellsIndexesArray = getFreeCellsIndexesArray(gameMatrix);

  if (freeCellsIndexesArray.length !== 0) {

    const randomIndex = getRandomInt(0, freeCellsIndexesArray.length - 1);

    return freeCellsIndexesArray[randomIndex];

  }

  return [];

}

function getFreeCellsIndexesArray(gameMatrix) {

  return gameMatrix.reduce((acc, row, rowIndex) => {

    const filteredRow = [];

    row.forEach((cell, cellIndex) => {

      if (cell.element === '') {

        filteredRow.push([rowIndex, cellIndex]);

      }

    });

    return [...acc, ...filteredRow];

  }, []);

}

function getRandomInt(min, max) {

  min = Math.ceil(min);

  max = Math.floor(max);

  return Math.floor(Math.random() \* (max - min)) + min;

}

**minmax.js**

import { checkVictory } from '../helpers/checkVictory';

const scoresDesk = {

    o : 10,

    x: -10,

    tie: 0

}

export function minimax(board, depth, isMaximizing, ai, player){

    // check winner

    const winner = checkVictory(board);

    if(winner){

        return scoresDesk[winner.element];

    }

    if(isMaximizing) {

        let bestScore = -Infinity;

        for(let i = 0; i < 3; i++){

            for(let j = 0; j < 3; j++){

               if(board[i][j].element === ''){

                   board[i][j].element = ai;

                   let score = minimax(board, depth + 1, false, ai, player);

                   board[i][j].element = '';

                   bestScore = Math.max(score, bestScore);

               }

            }

        }

        return bestScore;

    } else {

        let bestScore = Infinity;

        for(let i = 0; i < 3; i++){

            for(let j = 0; j < 3; j++){

                if(board[i][j].element === ''){

                    board[i][j].element = player;

                    let score = minimax(board, depth + 1, true,  ai, player);

                    board[i][j].element = '';

                    bestScore = Math.min(score, bestScore);

                }

            }

        }

        return bestScore

    }

}

**Circle.js**

import React from 'react';

export function Circle(props) {

  const { color } = props;

  return (

    <svg width="120" height="120" viewBox="0 0 120 120" fill="none" xmlns="http://www.w3.org/2000/svg">

      <circle cx="60" cy="60" r="60" fill="transparent" />

      <circle cx="60" cy="60" r="57" stroke={color || '#AB72D9'} strokeOpacity="0.83" strokeWidth="6" />

    </svg>

  );

}

**Cross.js**

import React from 'react';

export function Cross(props) {

  const { color } = props;

  return (

    <svg width="124" height="120" viewBox="0 0 124 120" fill="none" xmlns="http://www.w3.org/2000/svg">

      <line

        x1="5.76777"

        y1="2.23223"

        x2="121.768"

        y2="118.232"

        stroke={color || '#EB6666'}

        strokeWidth="5"

      />

      <line

        y1="-2.5"

        x2="164.049"

        y2="-2.5"

        transform="matrix(-0.707107 0.707107 0.707107 0.707107 120 4)"

        stroke={color || '#EB6666'}

        strokeWidth="5"

      />

    </svg>

  );

}

**TicTacToe.js**

import React from 'react';

import PropTypes from 'prop-types';

import { TicToeElement } from '../TicToeElement/TicToeElement';

import styles from './TicTacToe.module.css';

TicTacToe.propTypes = {

  gameArray: PropTypes.array,

};

export function TicTacToe(props) {

  const {

    gameBoard,

    handleCellClick,

    playerSigns,

    isGameFinished

  } = props;

  return (

    <div className={styles.tic\_container}>

      {gameBoard.map(((row, rowIndex) => (

        <div

          className={styles.tic\_row}

          key={rowIndex}

        >

          {row.map((cell, cellIndex) => (

            <TicToeElement

              handleCellClick={handleCellClick}

              isGameFinished={isGameFinished}

              rowIndex={rowIndex}

              cellIndex={cellIndex}

              sign={cell.element}

              playerSigns={playerSigns}

              color={cell.color}

              key={`${rowIndex}\_${cellIndex}`}

            />

          ))}

        </div>

      )))}

    </div>

  );

}

**TicToeElement.js**

import React from 'react';

import PropTypes from 'prop-types';

import styles from './TicToe.module.css';

import { Cross } from '../Cross/Cross';

import { Circle } from '../Circle/Circle';

export const TicToeElement = (props) => {

  const {

    sign,

    color,

    rowIndex,

    cellIndex,

    handleCellClick,

    playerSigns,

    isGameFinished

  } = props;

  const getTicTacToeSign = (sign) => {

    const ticToeSigns = {

      x: <Cross color={color} />,

      o: <Circle color={color} />,

    };

    return ticToeSigns[sign] || null;

  };

  return (

    <div className={styles.tic\_element\_container} onClick={

      !isGameFinished

          ? () => handleCellClick(rowIndex, cellIndex, playerSigns.player)

          : null

      }>

      {getTicTacToeSign(sign)}

    </div>

  );

};

TicToeElement.propTypes = {

  sign: PropTypes.string,

};

export default TicToeElement;

**checkVictory.js**

export function checkVictory(board) {

   // check every row

   for (let i = 0; i < board.length; i++) {

      const row = board[i];

      if (compareThree(row[0].element, row[1].element, row[2].element)) {

         return {

            element: row[0].element,

            combination: [

               {

                  row: i,

                  cell: 0

               },

               {

                  row: i,

                  cell: 1

               },

               {

                  row: i,

                  cell: 2

               }

            ]

         }

      }

   }

   // check every column

   for (let i = 0; i < board.length; i++) {

      if (compareThree(board[0][i].element, board[1][i].element, board[2][i].element)) {

         return {

            element: board[0][i].element,

            combination: [

               {

                  row: 0,

                  cell: i

               },

               {

                  row: 1,

                  cell: i

               },

               {

                  row: 2,

                  cell: i

               }

            ]

         }

      }

   }

   // check left diagonal

   if (compareThree(board[0][0].element, board[1][1].element, board[2][2].element)) {

      return {

         element: board[0][0].element,

         combination: [

            {

               row: 0,

               cell: 0

            },

            {

               row: 1,

               cell: 1

            },

            {

               row: 2,

               cell: 2

            }

         ]

      }

   }

   // check right diagonal

   if (compareThree(board[0][2].element, board[1][1].element, board[2][0].element)) {

      return {

         element: board[0][2].element,

         combination: [

            {

               row: 0,

               cell: 2

            },

            {

               row: 1,

               cell: 1

            },

            {

               row: 2,

               cell: 0

            }

         ]

      }

   }

   //check tie

   for (let i = 0; i < 3; i++) {

      for (let j = 0; j < 3; j++) {

         if (board[i][j].element === '') {

            return null

         }

      }

   }

   return {

      element: 'tie',

      combination: null

   }

}

function compareThree(firstCell, secondCell, thirdCell) {

   if (firstCell === '') {

      return false

   }

   return firstCell === secondCell && secondCell === thirdCell;

}

**App.js**

import React from 'react';

import './App.css';

import { TicTacToe } from './components/TicTacToe/TicTacToe';

import { useApp } from './useApp';

function App() {

   const {

      gameBoard,

      handleCellClick,

      playerSigns,

      setGameBoard,

      colorCombination,

      isGameFinished,

      resetGame

   } = useApp();

   return (

      <div className="App">

         <div className="game\_title">Крестики-Нулики</div>

         <TicTacToe

            isGameFinished={isGameFinished}

            gameBoard={gameBoard}

            handleCellClick={handleCellClick}

            playerSigns={playerSigns}

            setGameBoard={setGameBoard}

            colorCombination={colorCombination}

         />

         {isGameFinished

            ? <div

               className="button"

               onClick={resetGame}

            >

               Сыграть заново ?

            </div>

            : null

         }

      </div>

   );

}

export default App;

**useApp.js**

import { useState } from 'react';

import { checkFreeCells } from "./ai/checkFreeCells";

import { checkVictory } from "./helpers/checkVictory";

import { bestMove } from "./ai/bestMove";

const initialArray = [

   [{ element: '', color: null }, { element: '', color: null }, { element: '', color: null }],

   [{ element: '', color: null }, { element: '', color: null }, { element: '', color: null }],

   [{ element: '', color: null }, { element: '', color: null }, { element: '', color: null }],

];

export function useApp() {

   const [gameBoard, setGameBoard] = useState(initialArray);

   const [playerSigns] = useState({ ai: 'o', player: 'x' })

   const [isGameFinished, setIsGameFinished] = useState(false);

   console.log('gameBoard', gameBoard)

   const setCellValue = (rowIndex, cellIndex, value) => {

      const draftCell = gameBoard[rowIndex][cellIndex];

      if (draftCell.element === '') {

         draftCell.element = value;

         const draftState = [...gameBoard];

         draftState[rowIndex][cellIndex] = {

            element: value,

            color: null,

         };

         setGameBoard(draftState);

      }

   };

   const handleCellClick = (rowIndex, cellIndex, value) => {

      if (gameBoard[rowIndex][cellIndex].element === '') {

         setCellValue(rowIndex, cellIndex, value);

         const isFreeCellsExists = checkFreeCells(gameBoard);

         if (isFreeCellsExists) {

            bestMove(gameBoard, playerSigns.ai, playerSigns.player, setGameBoard);

         }

         const winner = checkVictory(gameBoard);

         if (winner) {

            setIsGameFinished(true)

            if (winner.element !== 'tie') {

               colorCombination(winner.combination);

            }

         }

      }

   };

   const colorCombination = (indexes) => {

      const updatedBoard = [...gameBoard];

      indexes.forEach((index) => {

         updatedBoard[index.row][index.cell].color = '#3498db';

      })

      setGameBoard(updatedBoard);

   }

   const resetGame = () => {

      setIsGameFinished(false);

      setGameBoard([

         [{ element: '', color: null }, { element: '', color: null }, { element: '', color: null }],

         [{ element: '', color: null }, { element: '', color: null }, { element: '', color: null }],

         [{ element: '', color: null }, { element: '', color: null }, { element: '', color: null }]

      ])

   }

   return {

      gameBoard,

      handleCellClick,

      playerSigns,

      isGameFinished,

      resetGame

   };

}

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));

**Список литературы**

1. Краткое описание алгоритма А\*
2. Краткое описание алгоритма «муравьиной колонии».
3. Учебное пособие: Толмачев С.Г. «Алгоритмы поиска в системах искусственного интеллекта». БГТУ 2012. см. глава 4 – Поиск на графе (параграф 7.4)