LAMPIRAN

```
import { useMemo } from 'react';
import { HTML5Backend } from 'react-dnd-html5-backend';
import { DndProvider } from 'react-dnd';
import { Grid, GridItem } from '@chakra-ui/react';
import { TouchBackend } from 'react-dnd-touch-backend';
import { useDraughtsBoard } from '../DraughtsBoardContext';
import { useDraughtsSettings } from '../../settings/DraughtsSettingsContext';
import { DraughtsCell } from './DraughtsCell';
import { DraughtsGameOverModal } from './DraughtsGameOverModal';
import { Players } from '@draughts/core';

function isTouchDevice() {
  return (
    'ontouchstart' in window ||
    navigator.maxTouchPoints > 0 ||
    navigator.msMaxTouchPoints > 0
  );
}

export function DraughtsBoard() {
  const { board } = useDraughtsBoard();
  const { userPlayer } = useDraughtsSettings();

  const backend = useMemo(() => {
    if (typeof window === 'undefined') return HTML5Backend;
    return isTouchDevice() ? TouchBackend : HTML5Backend;
  }, []);

  const rows = useMemo(() => {
    const entries = board.position.map((row, rowIndex) => ({
      row,
      rowIndex,
    }));
    if (userPlayer === Players.WHITE) {
      return entries;
    }
    return entries.reverse();
  }, [userPlayer, board.position]);

  return (
    <DndProvider backend={backend}>
      <DraughtsGameOverModal />
      <Grid
        templateRows="repeat(8, 1fr)"
        templateColumns="repeat(8, 1fr)"
        w="100%"
        h="100%"
        style={{ aspectRatio: 1 }}
      >
        {rows.map(({ row, rowIndex }) =>
          row.map((piece, colIndex) => (
            // eslint-disable-next-line react/no-array-index-key
            <GridItem key={`${rowIndex}:${colIndex}:${piece}`}>
              <DraughtsCell
                piece={piece}
                rowIndex={rowIndex}
                colIndex={colIndex}
              />
            </GridItem>
          ))
        )}
      </Grid>
    </DndProvider>
  );
}

import PropTypes from 'prop-types';
import { useDrop } from 'react-dnd';
```

```
import { Box, Square } from
'@chakra-ui/react';
import { useDraughtsBoard } from
'../DraughtsBoardContext';
import { DraughtsPiece } from
'./DraughtsPiece';
import { compareCells, Pieces }
from '@draughts/core';

DraughtsCell.propTypes = {
  colIndex:
PropTypes.number.isRequired,
  piece:
PropTypes.oneOf(Object.values(Pi
eces)).isRequired,
  rowIndex:
PropTypes.number.isRequired,
};

export                   function
DraughtsCell(props) {
  const { board, doMove, lastMove
} = useDraughtsBoard();
  const isDark = (props.rowIndex
+ props.colIndex) % 2 === 0;
  const currentCell = { col:
props.colIndex,              row:
props.rowIndex };
  const       isLastMove       =
compareCells(currentCell,
lastMove);

  const   validMovePredicate   =
(start) => (move) => {
    const       startMove       =
move.path.at(0);
    const        endMove        =
move.path.at(-1);
    return
compareCells(startMove, start) &&
compareCells(endMove,
currentCell);
  };

  const [{ isOver, canDrop },
dropReference] = useDrop(
    () => ({
      accept: 'piece',
      canDrop:     (start)     =>
board.moves.some(validMovePredic
ate(start)),
      collect: (monitor) => ({
        canDrop:
!!monitor.canDrop(),
        isOver:
!!monitor.isOver(),
      }),
      drop: (start) => {

doMove(board.moves.find(validMov
ePredicate(start)));
      },
    }),
    [board.moves, doMove]
  );

  let bg = 'gray.300';
  if (canDrop) {
    bg = 'lightblue';
  } else if ((canDrop && isOver)
|| isLastMove) {
    bg = 'lightgreen';
  } else if (isDark) {
    bg = 'gray.500';
  }

  return (
    <Square
      ref={dropReference}
      pos="relative"
      h="100%"
      p="0.2em"
      bg={bg}
      userSelect="none"
    >
      <Box
        pos="absolute"
        top="0.1rem"
        right="0.1rem"
        fontSize="0.4rem"
        opacity={0.4}
        userSelect="none"
      >
        {props.rowIndex},
{props.colIndex}
      </Box>
      {props.piece            !==
Pieces.NONE && (
        <DraughtsPiece
          piece={props.piece}

rowIndex={props.rowIndex}

colIndex={props.colIndex}
        />
      )}
    </Square>
  );
```

```
}

const { Icon } = require('@chakra-ui/react');

export function DraughtsCrown(props) {
  return (
    <Icon viewBox="0 0 230 200" {...props}>
      <path
        d="m9,51 72,21 37-64 37,64 72-21-33,99H42 m75-74a15,29 0 1,0 2,0m71,86-11,33H57l-11-33"
        fill="gold"
      />
    </Icon>
  );
}

import { useEffect } from 'react';
import {
  Modal,
  Button,
  Text,
  ModalOverlay,
  ModalContent,
  ModalHeader,
  ModalFooter,
  ModalBody,
  ModalCloseButton,
  useDisclosure,
  Divider,
} from '@chakra-ui/react';
import { useDraughtsBoard } from '../DraughtsBoardContext';
import { useDraughtsWinner } from '../hooks/use-draughts-winner';
import { useDraughtsGame } from '../../game/DraughtsGameContext';
import { useDraughtsSettings } from '../../settings/DraughtsSettingsContext';
import { formatPlayer, Players, GameStates } from '@draughts/core';

function capitalizeFirstLetter(string) {
  return string.at(0).toUpperCase() + string.slice(1);
}

function formatGameOverMessage(winner) {
  if (winner === Players.NONE) return 'The game was a draw.';
  // const winnerFormatted = capitalizeFirstLetter(formatPlayer(winner));
  // return `${winnerFormatted} won the game.`;
}

export function DraughtsGameOverModal() {
  const { isOpen, onOpen, onClose } = useDisclosure();
  const { restartGame } = useDraughtsGame();
  const { board } = useDraughtsBoard();
  const { settingsModal } = useDraughtsSettings();
  const { userWon, winner } = useDraughtsWinner();

  useEffect(() => {
    if (board.state === GameStates.PLAYING) return;
    onOpen();
    return onClose;
  }, [board.state, onOpen, onClose]);

  return (
    <Modal isOpen={isOpen} onClose={onClose}>
      <ModalOverlay />
      <ModalContent>
        <ModalHeader>
          {userWon ? 'Wooo! Anda Menang. 🎉' : 'Semoga lain kali lebih beruntung... 💪'}
        </ModalHeader>
        <ModalCloseButton />
        <ModalBody>

<Text>{formatGameOverMessage(winner)}</Text>
          <Divider marginY={5} />
          <Text>
```

```
            Mengapa          bukan
kesulitan komputer yang berbeda,
Main Lagi Bosku, Anda Masih Kuat,
Bertenaga?
          </Text>
        </ModalBody>
        <ModalFooter>
          <Button
            mr={3}
            colorScheme="blue"
            onClick={() => {
              onClose();

settingsModal.onOpen();
            }}
          >
            Setting Game
          </Button>
          <Button
            onClick={() => {
              restartGame();
              onClose();
            }}
          >
            Main Lagi
          </Button>
        </ModalFooter>
      </ModalContent>
    </Modal>
  );
}

import { useDrag } from 'react-
dnd';
import { Center } from '@chakra-
ui/react';
import   PropTypes   from   'prop-
types';
import { useDraughtsBoard } from
'../DraughtsBoardContext';
import   {   useDraughtsSettings   }
from
'../../settings/DraughtsSettings
Context';
import   {   DraughtsCrown   }   from
'./DraughtsCrown';
import {
  compareCells,
  pieceIsPlayer,
  pieceIsQueen,
  Pieces,
  Players,
} from '@draughts/core';

export                    function
DraughtsPiece(props) {
  const     {     board     }     =
useDraughtsBoard();
  const    {    userPlayer    }    =
useDraughtsSettings();

  const         isWhite         =
pieceIsPlayer(props.piece,
Players.WHITE);
  const activePlayer =
    pieceIsPlayer(props.piece,
userPlayer) &&
    pieceIsPlayer(props.piece,
board.playerToMove);

  const   currentCell   =   {   col:
props.colIndex,              row:
props.rowIndex };

  const canDrag =
    activePlayer &&
    board.moves.some((move)    =>
compareCells(move.path.at(0),
currentCell));

  const [, dragRef] = useDrag(
    () => ({
      canDrag: () => canDrag,
      isDragging:   (monitor)   =>
compareCells(monitor.getItem(),
currentCell),
      item: currentCell,
      type: 'piece',
    }),
    [board, canDrag, currentCell]
  );

  return (
    <Center
      ref={dragRef}
      zIndex="1"
      w="100%"
      h="100%"
      opacity={canDrag ? 1 : 0.8}
      borderWidth="0.2em"
      borderStyle="solid"
      borderColor={isWhite      ?
'gray.400' : 'gray.600'}
      borderRadius="50%"
      bgColor={isWhite          ?
'yellow.50' : 'gray.900'}
    >
```

```
      {pieceIsQueen(props.piece)
&& (
        <DraughtsCrown    w="70%"
h="70%"              opacity="0.6"
userSelect="none" />
      )}
    </Center>
  );
}

DraughtsPiece.propTypes = {
  colIndex: PropTypes.number,
  piece:
PropTypes.oneOf(Object.values(Pi
eces)).isRequired,
  rowIndex: PropTypes.number,
};

import      {      createContext,
useCallback, useContext, useState
} from 'react';
import   PropTypes   from   'prop-
types';
import { Board, Pieces, Players }
from '@draughts/core';

export const DraughtsBoardContext
= createContext();

export  const  useDraughtsBoard  =
()                               =>
useContext(DraughtsBoardContext)
;

export                    function
DraughtsBoardProvider(props) {
  const [lastMove, setLastMove] =
useState(null);
  const   [board,   setBoard]   =
useState(
    new     Board(props.position,
props.playerToMove)
  );

  const         doMove         =
useCallback((move) => {
    setBoard((board) => {
      return board.doMove(move);
    });
    setLastMove(move.path.at(-
1));
  }, []);

  const       resetBoard       =
useCallback(() => {
    setBoard(new
Board(props.position,
props.playerToMove));
    setLastMove(null);
  },            [props.position,
props.playerToMove]);

  return (

<DraughtsBoardContext.Provider
      value={{
        board,
        doMove,
        lastMove,
        resetBoard,
      }}
    >
      {props.children}

</DraughtsBoardContext.Provider>
  );
}

export                        const
DraughtsBoardProviderProps = {
  playerToMove:
PropTypes.oneOf(Object.values(Pl
ayers)),
  position: PropTypes.arrayOf(

PropTypes.arrayOf(PropTypes.oneO
f(Object.values(Pieces)))
  ).isRequired,
};

DraughtsBoardProvider.propTypes =
{
  children:
PropTypes.node.isRequired,
  ...DraughtsBoardProviderProps,
};

import { quiescenceSearch } from
'./quiescence-search';
import   {   GameStates   }   from
'@draughts/core';

const getShuffledArray = (arr) =>
{
  const newArr = [...arr];
  for (let i = newArr.length - 1;
i > 0; i--) {
```

```
    const rand = Math.floor(Math.random() * (i + 1));
    [newArr[i], newArr[rand]] = [newArr[rand], newArr[i]];
  }
  return newArr;
};

export function alphaBetaMove(board, depth) {
  let recordE = Number.NEGATIVE_INFINITY;
  let recordMove = null;

  for (const move of getShuffledArray(board.moves)) {
    const nextBoard = board.doMove(move);
    const e = -alphaBetaSearch(
      nextBoard,
      depth - 1,
      Number.NEGATIVE_INFINITY,
      Number.POSITIVE_INFINITY
    );
    if (e >= recordE) {
      recordE = e;
      recordMove = move;
    }
  }

  return recordMove;
}

export function alphaBetaSearch(board, depth, alpha, beta) {
  if (depth === 0 || board.state !== GameStates.PLAYING)
    return quiescenceSearch(board, alpha, beta);

  for (const move of board.moves) {
    const nextBoard = board.doMove(move);
    const e = -alphaBetaSearch(nextBoard, depth - 1, -beta, -alpha);
    if (e >= beta) return beta;
    alpha = Math.max(e, alpha);
  }
  return alpha;
}

import {
  eachCell,
  pieceIsPlayer,
  pieceIsQueen,
  BOARD_SIZE,
  Players,
  GameStates,
} from '@draughts/core';

const winnerMap = {
  [GameStates.WHITE_WON]: Players.WHITE,
  [GameStates.BLACK_WON]: Players.BLACK,
};

export function evaluateBoard(board) {
  if (board.state === GameStates.DRAW) {
    return 0;
  }
  if (board.state !== GameStates.PLAYING) {
    const winner = winnerMap[board.state];
    return board.playerToMove === winner
      ? Number.POSITIVE_INFINITY
      : Number.NEGATIVE_INFINITY;
  }
  if (isEndgame(board)) {
    return evaluateEndgamePosition(board);
  }
  return evaluateRegularPosition(board);
}

function isEndgame(board) {
  for (const { piece } of eachCell(board.position)) {
    if (!pieceIsPlayer(piece, Players.NONE) && !pieceIsQueen(piece)) {
      return false;
    }
  }
  return true;
}
```

```
function
evaluateEndgamePosition(board) {
  let playerPieces = 0;
  let opponentPieces = 0;
  let distances = 0;
  for (const { cell, piece } of
eachCell(board.position)) {
    if      (pieceIsPlayer(piece,
board.playerToMove)) {
      playerPieces += 1;
      distances                +=
calculateDistances(board, cell);
    }           else           if
(!pieceIsPlayer(piece,
Players.NONE)) {
      opponentPieces += 1;
    }
  }
  if       (playerPieces       >=
opponentPieces) {
    return -distances;
  }
  return distances;
}

function
evaluateRegularPosition(board) {
  let e = 0;
  for (const { cell, piece } of
eachCell(board.position)) {
    const    pieceEvaluation    =
evaluatePiece(cell, piece);
    if      (pieceIsPlayer(piece,
board.playerToMove)) {
      e += pieceEvaluation;
    }           else           if
(!pieceIsPlayer(piece,
Players.NONE)) {
      e -= pieceEvaluation;
    }
  }
  return e;
}

function      evaluatePiece(cell,
piece) {
  let e = 20;
  if (pieceIsQueen(piece)) {
    e += 40;
  } else if (pieceIsPlayer(piece,
Players.WHITE)) {
    e   +=   BOARD_SIZE   -   1   -
cell.row;
  } else if (pieceIsPlayer(piece,
Players.BLACK)) {
    e += cell.row;
  }
  return e;
}

function
calculateDistances(board,
baseCell) {
  let distances = 0;
  for (const { cell, piece } of
eachCell(board.position)) {
    if (
      !pieceIsPlayer(piece,
board.playerToMove) &&
      !pieceIsPlayer(piece,
Players.NONE)
    ) {
      distances                +=
euclideanDistance(baseCell,
cell);
    }
  }
  return distances;
}

function euclideanDistance(a, b)
{
  const        rowSquare        =
Math.pow(a.row - b.row, 2);
  const        colSquare        =
Math.pow(a.col - b.col, 2);
  const      distanceFloat      =
Math.sqrt(rowSquare + colSquare);
  return
Math.ceil(distanceFloat);
}

export  {  alphaBetaMove  }  from
'./alpha-beta-search';

import {
  eachCell,
  pieceIsPlayer,
  pieceIsQueen,
  BOARD_SIZE,
  Players,
  GameStates,
} from '@draughts/core';

const winnerMap = {
  [GameStates.WHITE_WON]:
Players.WHITE,
```

```
  [GameStates.BLACK_WON]:
Players.BLACK,
};

export                    function
negascout_search(board) {
  if       (board.state       ===
GameStates.DRAW) {
    return 0;
  }
  if       (board.state       !==
GameStates.PLAYING) {
    const         winner         =
winnerMap[board.state];
    return board.playerToMove ===
winner
      ? Number.POSITIVE_INFINITY
      : Number.NEGATIVE_INFINITY;
  }
  if (isEndgame(board)) {
    return
evaluateEndgamePosition(board);
  }
  return
evaluateRegularPosition(board);
}

function        eval_board(Board,
pieceType, restrictions) {
    let score = 0;
    const         min_r         =
restrictions[0];
    const         min_c         =
restrictions[1];
    const         max_r         =
restrictions[2];
    const         max_c         =
restrictions[3];
    for (let row = min_r; row <
max_r + 1; row++) {
        for (let column = min_c;
column < max_c + 1; column++) {
            if
(Board[row][column] == pieceType)
{
                let block = 0;
                let piece = 1;
                // left
                if (column === 0
|| Board[row][column - 1] !== 0)
{
                    block++;
                }
                // pieceNum
                for    (column++;
column      <      Columns      &&
Board[row][column] === pieceType;
column++) {
                    piece++;
                }
                // right
                if   (column   ===
Columns || Board[row][column] !==
0) {
                    block++;
                }
                score          +=
evaluateblock(block, piece);
            }
        }
    }

    for  (let  column  =  min_c;
column < max_c + 1; column++) {
        for (let row = min_r; row
< max_r + 1; row++) {
            if
(Board[row][column] == pieceType)
{
                let block = 0;
                let piece = 1;
                // left
                if (row === 0 ||
Board[row - 1][column] !== 0) {
                    block++;
                }
                // pieceNum
                for (row++; row <
Rows  &&  Board[row][column]  ===
pieceType; row++) {
                    piece++;
                }
                // right
                if (row === Rows
|| Board[row][column] !== 0) {
                    block++;
                }
                score          +=
evaluateblock(block, piece);
            }
        }
    }

    for (let n = min_r; n < (max_c
- min_c + max_r); n += 1) {
        let r = n;
        let c = min_c;
```

```
        while (r >= min_r && c <=
max_c) {
            if (r <= max_r) {
                if   (Board[r][c]
=== pieceType) {
                    let  block  =
0;
                    let  piece  =
1;
                    // left
                    if  (c  ===  0
||  r  ===  Rows  -  1  ||  Board[r  +
1][c - 1] !== 0) {
                        block++;
                    }
                    // pieceNum
                    r--;
                    c++;
                    for (; r >= 0
&& Board[r][c] === pieceType; r--
) {
                        piece++;
                        c++
                    }
                    // right
                    if (r < 0 ||
c === Columns || Board[r][c] !==
0) {
                        block++;
                    }
                    score      +=
evaluateblock(block, piece);
                }
            }
            r -= 1;
            c += 1;
        }
    }

    for (let n = min_r - (max_c -
min_c); n <= max_r; n++) {
        let r = n;
        let c = min_c;
        while (r <= max_r && c <=
max_c) {
            if (r >= min_r && r
<= max_r) {
                if   (Board[r][c]
=== pieceType) {
                    let  block  =
0;
                    let  piece  =
1;
                    // left
                    if  (c  ===  0
||  r  ===  0  ||  Board[r  -  1][c  -  1]
!== 0) {
                        block++;
                    }
                    // pieceNum
                    r++;
                    c++;
                    for  (;  r  <
Rows && Board[r][c] == pieceType;
r++) {
                        piece++;
                        c++;
                    }
                    // right
                    if   (r   ===
Rows   ||   c   ===   Columns   ||
Board[r][c] !== 0) {
                        block++;
                    }
                    score      +=
evaluateblock(block, piece);
                }
            }
            r += 1;
            c += 1;
        }

    }
    return score;
}

function    evaluateblock(blocks,
pieces) {
    if (blocks === 0) {
        switch (pieces) {
            case 1:
                return LiveOne;
            case 2:
                return LiveTwo;
            case 3:
                return LiveThree;
            case 4:
                return LiveFour;
            default:
                return Five;
        }
    } else if (blocks === 1) {
        switch (pieces) {
            case 1:
                return DeadOne;
            case 2:
                return DeadTwo;
            case 3:
```

```
                return DeadThree;
            case 4:
                return DeadFour;
            default:
                return Five;
        }
    } else {
        if (pieces >= 5) {
            return Five;
        } else {
            return 0
        }
    }
}

function check_directions(arr) {
    for (let i = 0; i < arr.length
- 4; i++) {
        if (arr[i] !== 0) {
            if (arr[i] === arr[i
+ 1] && arr[i] === arr[i + 2] &&
arr[i] === arr[i + 3] && arr[i]
=== arr[i + 4]) {
                return true
            }
        }

    }
}

function get_directions(Board, x,
y) {
    const      Directions      =
[[],[],[],[]];
    for (let i = -4; i < 5; i++)
{
        if (x + i >= 0 && x + i
<= Rows - 1) {

Directions[0].push(Board[x      +
i][y])
            if (y + i >= 0 && y +
i <= Columns - 1) {

Directions[2].push(Board[x + i][y
+ i])
            }
        }
        if (y + i >= 0 && y + i
<= Columns - 1) {

Directions[1].push(Board[x][y   +
i])
            if (x - i >= 0 && x -
i <= Rows - 1) {

Directions[3].push(Board[x - i][y
+ i])
            }
        }

    }
    return Directions
}

function checkwin(Board, x, y) {
    const      Directions      =
get_directions(Board, x, y)
    for (let i = 0; i < 4; i++) {
        if
(check_directions(Directions[i])
) {
            return true
        }
    }
}

function remoteCell(Board, r, c)
{
    for (let i = r - 2; i <= r +
2; i++) {
        if (i < 0 || i >= Rows)
continue;
        for (let j = c - 2; j <=
c + 2; j++) {
            if (j < 0 || j >=
Columns) continue;
            if (Board[i][j] !==
0) return false;
        }
    }
    return true;
}

function  Get_restrictions(Board)
{
    let min_r = Infinity;
    let min_c = Infinity;
    let max_r = -Infinity;
    let max_c = -Infinity;
    for (let i = 0; i < Rows; i++)
{
        for (let j = 0; j <
Columns; j++) {
            if (Board[i][j] !==
0) {
```

```
                min_r           =
Math.min(min_r, i)
                min_c           =
Math.min(min_c, j)
                max_r           =
Math.max(max_r, i)
                max_c           =
Math.max(max_c, j)
            }
        }
    }
    if (min_r - 2 < 0) {
        min_r = 2;
    }
    if (min_c - 2 < 0) {
        min_c = 2;
    }
    if (max_r + 2 >= Rows) {
        max_r = Rows - 3;
    }
    if (max_c + 2 >= Columns) {
        max_c = Columns - 3;
    }
    return [min_r, min_c, max_r,
max_c]
}

function
Change_restrictions(restrictions
, i, j) {
    let min_r = restrictions[0];
    let min_c = restrictions[1];
    let max_r = restrictions[2];
    let max_c = restrictions[3];
    if (i < min_r) {
        min_r = i
    } else if (i > max_r) {
        max_r = i
    }
    if (j < min_c) {
        min_c = j
    } else if (j > max_c) {
        max_c = j
    }
    if (min_r - 2 < 0) {
        min_r = 2;
    }
    if (min_c - 2 < 0) {
        min_c = 2;
    }
    if (max_r + 2 >= Rows) {
        max_r = Rows - 3;
    }
    if (max_c + 2 >= Columns) {
        max_c = Columns - 3;
    }
    return [min_r, min_c, max_r,
max_c]
}

function compare(a, b) {
    if (a.score < b.score)
        return 1;
    if (a.score > b.score)
        return -1;
    return 0;
}

function
BoardGenerator(restrictions,
Board, player) {
    const availSpots_score = [];
//c is j  r is i;
    const        min_r         =
restrictions[0];
    const        min_c         =
restrictions[1];
    const        max_r         =
restrictions[2];
    const        max_c         =
restrictions[3];;
    for (let i = min_r - 2; i <=
max_r + 2; i++) {
        for (let j = min_c - 2; j
<= max_c + 2; j++) {
            if (Board[i][j] === 0
&& !remoteCell(Board, i, j)) {
                const move = {}
                move.i = i;
                move.j = j;
                move.score      =
evaluate_move(Board,    i,    j,
player)
                if    (move.score
=== WIN_DETECTED) {
                    return [move]
                }

availSpots_score.push(move)
            }
        }
    }

availSpots_score.sort(compare);
    //                    return
availSpots_score.slice(0,20)
    return availSpots_score;
}
```

```
function
evaluate_direction(direction_arr
, player) {
    let score = 0;
    for (let i = 0;(i + 4) <
direction_arr.length; i++) {
        let you = 0;
        let enemy = 0;
        for (let j = 0; j <= 4;
j++) {
            if (direction_arr[i +
j] === player) {
                you++
            }       else       if
(direction_arr[i   +   j]   ===   -
player) {
                enemy++
            }
        }
        score                   +=
evalff(get_seq(you, enemy));
        if ((score >= 800000)) {
            return WIN_DETECTED;
        }
    }
    return score
}

function get_seq(y, e) {
    if (y + e === 0) {
        return 0;
    }
    if (y !== 0 && e === 0) {
        return y
    }
    if (y === 0 && e !== 0) {
        return -e
    }
    if (y !== 0 && e !== 0) {
        return 17
    }
}

function  evaluate_move(Board,  x,
y, player) {
    let score = 0;
    const      Directions      =
get_directions(Board, x, y);
    let temp_score;
    for (let i = 0; i < 4; i++) {
        temp_score              =
evaluate_direction(Directions[i]
, player);
        if    (temp_score    ===
WIN_DETECTED) {
            return WIN_DETECTED
        } else {
            score += temp_score
        }
    }
    return score;
}


function    evaluate_state(Board,
player, hash, restrictions) {
    const        black_score    =
eval_board(Board,             -1,
restrictions);
    const        white_score    =
eval_board(Board,              1,
restrictions);
    let score = 0;
    if (player == -1) {
        score  =  (black_score  -
white_score);
    } else {
        score  =  (white_score  -
black_score);
    }
    StateCache.set(hash,score);
    StateCachePuts++;
    return score;
}



function random32() {
    let o = new Uint32Array(1);

self.crypto.getRandomValues(o);
    return o[0];
}

function Table_init() {
    for (let i = 0; i < Rows; i++)
{
        Table[i] = [];
        for  (let  j  =  0;  j  <
Columns; j++) {
            Table[i][j] = []
            Table[i][j][0]      =
random32(); //1
            Table[i][j][1]      =
random32(); //2
        }
```

```
    }

}

function hash(board) {
    let h = 0;
    let p;
    for (let i = 0; i < Rows; i++)
{
        for (let j = 0; j <
Columns; j++) {
            let Board_value =
board[i][j];
            if (Board_value !==
0) {
                if (Board_value
=== -1) {
                    p = 0
                } else {
                    p = 1
                }
                h = h ^
Table[i][j][p];
            }
        }
    }
    return h;
}

function update_hash(hash,
player, row, col) {
    if (player === -1) {
        player = 0
    } else {
        player = 1
    }
    hash = hash ^
Table[row][col][player];
    return hash
}


function negascout(newBoard,
player, depth, alpha, beta, hash,
restrictions, last_i, last_j) {
    const alphaOrig = alpha;
    const CacheNode
=Cache.get(hash)
    if ((CacheNode !== undefined)
&& (CacheNode.depth >= depth)) {
        CacheHits++;
        const score =
CacheNode.score;
        if (CacheNode.Flag === 0)
{
            CacheCutoffs++;
            return score
        }
        if (CacheNode.Flag === -
1) {
            alpha =
Math.max(alpha, score);
        } else if (CacheNode.Flag
=== 1) {
            beta = Math.min(beta,
score);
        }
        if (alpha >= beta) {
            CacheCutoffs++
            return score
        }
    }
    fc++

    if (checkwin(newBoard,
last_i, last_j)) {
        return -2000000 +
(MaximumDepth - depth)
    }
    if (depth === 0) {
        const
StateCacheNode=StateCache.get(ha
sh);
        if (StateCacheNode !==
undefined) {
            StateCacheHits++
            return StateCacheNode
        }
        return
evaluate_state(newBoard, player,
hash, restrictions)
    }

    const availSpots =
BoardGenerator(restrictions,
newBoard, player);
    if (availSpots.length === 0)
{
        return 0;
    }

    let b = beta;
    let bestscore = -Infinity;
    const bestMove={};
    for (let y = 0; y <
availSpots.length; y++) {
        let i = availSpots[y].i;
```

```
        let j = availSpots[y].j;
        const      newHash      =
update_hash(hash, player, i, j)
        newBoard[i][j] = player;
        const restrictions_temp =
Change_restrictions(restrictions
, i, j)
        let     score     =     -
negascout(newBoard,      -player,
depth  -  1,  -b,  -alpha,  newHash,
restrictions_temp, i, j)
        if (score > alpha && score
< beta && y > 0) {
            score       =       -
negascout(newBoard,      -player,
depth - 1, -beta, -score, newHash,
restrictions_temp, i, j)
        }
        if (score > bestscore) {
            bestscore = score
            if     (depth     ===
MaximumDepth) {
                bestMove.i=i
                bestMove.j=j

bestMove.score=score;
            }
        }
        newBoard[i][j] = 0;
        alpha  =  Math.max(alpha,
score)
        if (alpha >= beta) {
            break;
        }
        b = alpha + 1;
    }
    CachePuts++
    const             obj={score:
bestscore,depth:depth};
    if (bestscore <= alphaOrig) {
        obj.Flag = 1
    } else if (bestscore >= b) {
        obj.Flag = -1
    } else {
        obj.Flag = 0
    }
    Cache.set(hash,obj);
    if (depth == MaximumDepth) {
        return bestMove
    } else {
        return bestscore
    }
}
```

```
function
iterative_negascout(player,
Board, depth) {
    let bestmove;
    let i = 2;
    while (i !== depth + 2) {
        MaximumDepth = i;
        bestmove                =
negascout(Board,          player,
MaximumDepth,          -Infinity,
Infinity,            hash(Board),
Get_restrictions(Board), 0, 0)
        //
Set_last_best(bestmove)

console.log(MaximumDepth)
        console.log(bestmove)
        let         t11         =
performance.now();
        console.log((t11 - t00) /
1000)
        if    (bestmove.score    >
1999900) {
            break;
        }
        i += 2;
    }
    return bestmove
}

import  {  evaluateBoard  }  from
'./evaluate-board';

export                    function
quiescenceSearch(board,    alpha,
beta) {
  const          baseE          =
evaluateBoard(board);

  if (baseE >= beta) return beta;
  alpha = Math.max(baseE, alpha);

  for (const move of board.moves)
{
    if  (move.captures.length  <=
0) continue;

    const       nextBoard       =
board.doMove(move);
    const       e       =       -
quiescenceSearch(nextBoard,     -
beta, -alpha);

    if (e >= beta) return beta;
```

```
    alpha = Math.max(e, alpha);
  }

  return alpha;
}

import { ChakraProvider } from
'@chakra-ui/react';
import Head from 'next/head';
import Script from 'next/script';

//       eslint-disable-next-line
react/prop-types
export      default      function
App(props) {
  return (
    <>
      <Script
        strategy="lazyOnload"

src="https://www.googletagmanage
r.com/gtag/js?id=G-N7EVGCFBVC"
      />
      <Script
        id="google-analytics"
        strategy="lazyOnload"

dangerouslySetInnerHTML={{
          __html: `
            window.dataLayer    =
window.dataLayer || []; function

gtag(){dataLayer.push(arguments)
;} gtag('js', new Date());
            gtag('config',    'G-
N7EVGCFBVC', { page_path:

window.location.pathname, });
          `,
        }}
      />
      <Head>
        <link
          rel="apple-touch-icon"
          sizes="180x180"
          href="/apple-touch-
icon.png"
        />
        <link
          rel="icon"
          type="image/png"
          sizes="32x32"
          href="/favicon-
32x32.png"
        />
        <link
          rel="icon"
          type="image/png"
          sizes="16x16"
          href="/favicon-
16x16.png"
        />
        <link      rel="manifest"
href="/site.webmanifest" />
        <link     rel="mask-icon"
href="/safari-pinned-tab.svg"
color="#d6c420" />
        <meta
name="msapplication-TileColor"
content="#ffc40d" />
        <meta  name="theme-color"
content="#ffffff" />
        <meta  property="og:type"
content="website" />
        <meta
          property="og:title"
          content="Free    Online
Draughts Game - play against the
computer"
        />
        <meta   property="og:url"
content="https://draughts.org/"
/>
        <meta
property="og:site_name"
content="Draughts" />
        <meta

property="og:description"
          content="Play  draughts
free online against the computer.
Read  about  Rules  and  Strategies
for Draughts"
        />
        <meta       name="robots"
content="index, follow" />
        <meta
          name="description"
          content="Play  draughts
free online against the computer.
Read  about  Rules  and  Strategies
for Draughts"
        />
        <meta
          name="keywords"

content="draughts,checkers,free
draughts,free      checkers,board
```

```
games,board gaming online,play draughts online,play checkers online,play free games online,free games,online games,online checkers,online draughts"
        />
      </Head>
      <ChakraProvider>
        <props.Component {...props.pageProps} />
      </ChakraProvider>
    </>
  );
}

import NextDocument, { Html, Head, Main, NextScript } from 'next/document';

export default class Document extends NextDocument {
  render() {
    return (
      <Html lang="en">
        <Head />
        <body>
          <Main />
          <NextScript />
        </body>
      </Html>
    );
  }
}

import Head from 'next/head';
import { Container, Text, VStack } from '@chakra-ui/react';
import Image from 'next/image';
import { MainLayout } from '../components/layout/MainLayout';
import aquerqueImage from '../public/history/aquerque.jpeg';
import draughts1700sImage from '../public/history/draughts-1700s.webp';

export default function History() {
  return (
    <MainLayout>
      <Head>
        <title>Givan Dam - Main Dam</title>
      </Head>
      <VStack p={[3, 0]} spacing={6}>
        <Text>
          Penggalian arkeologi di Irak menemukan bentuk paling awal yang diketahui
          permainan Draf. Penanggalan karbon digunakan untuk menentukan umur
          permainan kuno, dan tampaknya berasal dari sekitar 3000 SM. Itu
          papan dan jumlah potongan yang digunakan berbeda dari
          Papan draf dan potongan digunakan saat ini.
        </Text>
        <Text>
          Sekitar 1400 SM, orang Mesir kuno menggunakan papan berukuran 5 x 5 untuk memainkan a
          permainan yang disebut Aquerque. Permainan ini sangat populer selama ini dan
          itu dimainkan di seluruh peradaban barat selama ribuan tahun.
        </Text>
        <Container maxW="md">
          <Image src={aquerqueImage} alt="ancient 5x5 egyption chessboard" />
        </Container>
        <Text>
          Sekitar tahun 1100 A.D., permainan Aquerque berubah ketika seorang Prancis memainkannya
          itu di papan catur menggunakan 12 buah untuk setiap pemain. Nama dari
          permainan juga berubah. Itu dikenal sebagai "Fierges."
        </Text>
        <Text>
          Evolusi Draf berikutnya terjadi ketika peraturan berubah lagi,
```

```
          membuatnya wajib untuk
melompati Draf untuk maju ke
seluruh papan.
          Versi yang lebih baru
ini lebih menantang daripada yang
lama. Versi lama
          dianggap lebih lambat
dan kurang menantang, dan menjadi
sosial
          permainan yang
dimainkan oleh para wanita pada
zaman itu dan disebut “La Jeu
Pleasant De
          Dames,” (permainan
wanita yang menyenangkan). Yang
baru, lebih agresif
          bentuk permainan
tersebut dikenal dengan nama “Jeu
Force” (Play Force).
        </Text>
        <Text>
          Draf akhirnya diekspor
dari Prancis ke Inggris, Spanyol,
dan
          Amerika. Di Spanyol
sekitar pertengahan tahun 1500-
an, buku mulai ada
          ditulis tentang Draf.
Pada tahun 1756, William Payne,
seorang matematikawan di
          Inggris, menulis
bukunya sendiri tentang Draf.
        </Text>
        <Container maxW="md">
          <Image

src={draughts1700sImage}
            alt="draughts being
played in the 1700s"
          />
        </Container>
        <Text>
          Pada tahun 1847,
kejuaraan dunia Draf pertama
berlangsung. Seiring waktu
          berlalu, menjadi jelas
bahwa game tersebut menghadirkan
bukaan yang memberi
          keuntungan satu pihak
atas pihak lain. Ada dua batasan
gerakan
          dibuat di mana
permainan dimulai dengan gaya
acak. Dua ini
          pembatasan bergerak
sebagian besar digunakan oleh
pemain ahli. Di zaman modern
          Turnamen draf, tiga
batasan gerakan digunakan.
        </Text>
        <Text>
          Seiring kemajuan
teknologi, tidak lama kemudian
programmer komputer
          mulai mengembangkan
game Draft yang sangat mendasar
yang dapat dimainkan
          komputer. Alan Turing
membuat game Draf yang belum
sempurna di atas kertas
          karena komputer pada
saat itu belum cukup berkembang
untuk menjalankannya
          Program draf. Pada
tahun 1952, Arthur L. Samuel
membuat Draf pertama
          program yang sebenarnya
bisa dimainkan di komputer. Dari
titik itu
          maju, game Draf
komputer telah meningkat dalam
kecepatan dan fungsi.
        </Text>
        <Text>
          Program Draf saat ini
membutuhkan perencanaan yang
kurang strategis dan lebih banyak
lagi
          kemampuan pencarian
data komputer. Program Draf
menggunakan database
          pencarian yang
menampilkan semua kemungkinan
kombinasi ketika ada beberapa
bagian
          tertinggal di papan.
        </Text>
      </VStack>
    </MainLayout>
  );
}

import Head from 'next/head';
import { Divider, Heading,
HStack, VStack } from '@chakra-
ui/react';
```

```
import { DraughtsMenuView } from
'../components/draughts/settings
/views/DraughtsMenu';
import { DraughtsProvider } from
'../components/draughts/Draughts
Context';
import { DraughtsBoard } from
'../components/draughts/board/vi
ews/DraughtsBoard';
import { DraughtsGameInfoView }
from
'../components/draughts/Draughts
GameInfo';
import { ComputerDifficulty }
from
'../components/draughts/settings
/constants/computer-difficulty';
import { MainLayout } from
'../components/layout/MainLayout
';
import { DraughtsRulesContent }
from
'../components/content/DraughtsR
ulesContent';
import { INITIAL_POSITION,
Players } from '@draughts/core';

export default function Home() {
  return (
    <MainLayout>
      <Head>
        <title>Givan Checkers -
Main Checkers Secara
Gratis</title>
      </Head>
      <DraughtsProvider
        settings={{
          computerDifficulty:
ComputerDifficulty.MEDIUM,
          userPlayer:
Players.WHITE,
        }}
        board={{ playerToMove:
Players.WHITE, position:
INITIAL_POSITION }}
      >
        <VStack spacing={4}>
          <DraughtsBoard />
          <HStack>
            <DraughtsMenuView />

<DraughtsGameInfoView />
          </HStack>
          <Divider />
          <Heading
size="sm">Peraturan
Main</Heading>
          <DraughtsRulesContent
/>
        </VStack>
      </DraughtsProvider>
    </MainLayout>
  );
}

/* eslint-disable
unicorn/filename-case */
import { AspectRatio } from
'@chakra-ui/react';
import Head from 'next/head';
import { DraughtsRulesContent }
from
'../components/content/DraughtsR
ulesContent';
import { MainLayout } from
'../components/layout/MainLayout
';

export default function History()
{
  return (
    <MainLayout>
      <Head>
        <title>Givan Checkers -
Peraturan &amp;amp; Cara
Bermain</title>
      </Head>
      <DraughtsRulesContent />
      {/*<AspectRatio maxW="md"
radio={16 / 9}>
        <iframe

src="https://www.youtube.com/emb
ed/PgNN6CdkYXs"
          title="YouTube video
player"
          frameBorder="0"
          allow="accelerometer;
autoplay; clipboard-write;
encrypted-media; gyroscope;
picture-in-picture"
          allowFullScreen
        ></iframe>
      </AspectRatio>*/}
    </MainLayout>
  );
}
```

```
import Head from 'next/head';
import { AspectRatio, Text, VStack } from '@chakra-ui/react';
import { MainLayout } from '../components/layout/MainLayout';

export default function History() {
  return (
    <MainLayout>
      <Head>
        <title>Givan Checkers - Strategi &amp; Pembukaan</title>
      </Head>
      <VStack p={[3, 0]} spacing={6}>
        <Text>
          Draf, juga dikenal sebagai checkers, adalah permainan papan strategi yang memiliki
          telah ada selama ribuan tahun. Ada banyak varian, tapi
          versi paling umum dimainkan di papan kotak-kotak 8x8. Keduanya
          permainan pemain terdiri dari dua belas buah (pria, catur, draft)per
          samping. Potongan dimulai dari tiga baris pertama pada warna hitam/gelap
          kotak saja. Pria hanya bisa maju secara diagonal &quot;melompat&quot; lebih
          bagian itu dan mendarat di ruang kosong yang berdekatan. Setelah semua pria memiliki
          ditangkap, permainan dimenangkan. Permainan juga bisa dimenangkan melalui
          menghalangi kemampuan lawan untuk bergerak. Ada banyak taktik yang berguna
          untuk meningkatkan peluang keberhasilan Anda.
        </Text>
        <Text>
          Menobatan, atau menjadikan raja, sangat meningkatkan kekuatan dan portabilitas
          laki-laki Anda. Jika Anda bisa mendapatkan bagian ke garis dasar pemain lain, itu
          bisa &quot;dimahkotai&quot;. Sepotong lain ditempatkan di atas
          membedakannya dari draf biasa. Raja sekarang dapat dipindahkan keduanya
          maju dan mundur, secara efektif menggandakan jangkauannya.
        </Text>
        <Text>
          Karena pria yang menangkap membutuhkan kotak kosong untuk dilompati, memang begitu
          bijaksana untuk memindahkan Anda potongan secara massal. Cobalah untuk tidak meninggalkan potongan individu
          terpencil. Pindahkan lebih sedikit potongan dalam formasi ketat.
        </Text>
        <Text>
          Cobalah untuk meninggalkan orang-orang garis dasar Anda di stasiun selama mungkin.
          Kotak bebas apa pun berpotensi untuk penobatan oposisi. Mereka akan
          tidak dapat membuat raja jika mereka tidak bisa mendarat di sana.
        </Text>
        <Text>
          Draf, seperti permainan papan lainnya, bekerja berdasarkan prinsip umum
          bertukar potongan setiap kali ada yang di depan. Keuntungan material dari memiliki
          hanya satu orang tambahan menjadi lebih signifikan secara proporsional semakin sedikit
          potongan tetap. Peluang mahkota akan meningkat pesat. Satu
```

```
          peringatan untuk
prinsip umum ini, adalah
mengabaikan keunggulan posisi
          untuk keuntungan materi
buta. Seorang raja bisa sangat
mengubah jalannya permainan
          dengan cepat.
        </Text>
        <Text>
          Mengorbankan draf bisa
tampak sembrono atau ceroboh.
Tapi ini
          strategi dapat
digunakan untuk menarik
keuntungan posisional. Untuk
menghapus a
          potongan dasar dalam
persiapan untuk penobatan
misalnya, akan menjadi a
          penggunaan taktik
pengorbanan yang baik.
        </Text>
        <Text>
          Aturan draf menyatakan
bahwa jika lawan menawarkan
bagian untuk
          menangkap, itu harus
diambil. &amp;quot;gerakan
paksa&amp;quot; dapat
          dipekerjakan untuk
keuntungan besar. Jika draf lawan
menghalangi Anda
          cara membuat raja, Anda
dapat memajukan bagian lain ke
sisi lain
          dari pemblokir yang
menyinggung. Ini akan memaksa
lawan Anda untuk menangkap
          memungkinkan jalur yang
jelas ke garis belakang untuk
penobatan.
        </Text>
        <Text>
          Memblokir digunakan
untuk menggagalkan gerakan
oposisi. Itu membutuhkan yang
baik
          banyak pemikiran ke
depan oleh pemain lawan. Mencoba
untuk yang kedua
          menebak rencana atau
rangkaian gerakan membutuhkan
pengetahuan yang baik tentang
          strategi. Sementara
pemblokiran bersifat defensif
dalam tujuannya (untuk mencegah
          pemain lain dari maju)
itu dapat menghasilkan posisi
menang. Aku jatuh
          potongan lawan diblokir
dan dia tidak bisa bergerak,
sesuai
          dengan aturan
permainan, dia kalah dalam
permainan.
        </Text>
        {/*<Text>Here&apos;s a
great video on draughts
strategies:</Text>*/}
        {/*<AspectRatio ratio={16
/ 9}>
          <iframe

src="https://www.youtube.com/emb
ed/Lfo3yfrbUs0"
            title="YouTube video
player"
            frameBorder="0"

allow="accelerometer; autoplay;
clipboard-write; encrypted-media;
gyroscope; picture-in-picture"
            allowFullScreen
          />
        </AspectRatio>*/}
      </VStack>
    </MainLayout>
  );
}
```