KALAHA

Kalaha Game Overview

Kalaha is a traditional two-player board game that we have implemented in code. In this game, each player has a total of 6 pits, initially containing 4 balls each. Additionally, each player has one larger pit called a "kalaha," where they aim to collect as many balls as possible.

Game Rules

- **Players:** There are two opponents in Kalaha.
- **Pits:** Each player has 6 pits, initially filled with 4 balls each.
- **Kalaha:** Each player also has one kalaha, which serves as a larger pit for collecting balls.
- **Gameplay:** Players take turns picking up the balls from one of their pits and distributing them counterclockwise, including their own kalaha, but not their opponent's kalaha.
- **Objective:** The game continues until one player's pits are empty. The winner is determined by who has the most balls in their kalaha.

Winning Strategy

- **Strategic Moves:** Players strategically aim to accumulate as many balls as possible in their own kalaha while preventing their opponent from doing the same.
- **Capture:** If the last ball of a player's turn lands in their own empty pit, that player captures that ball and any balls in the opposite pit, placing them all in their kalaha.
- **Repeat:** Players continue taking turns until one player's pits are empty, at which point the player with the most balls in their kalaha wins the game.

Which datastructures

- `ARRAY`

The game-array and UI-array

GAME-ARRAY	INDEX POS	UI-ARRAY
Player 1 - pit	0	Player 1 - Kalaha
Player 1 - pit	1	Player 1 - pit

GAME-ARRAY	INDEX POS	UI-ARRAY
Player 1 - pit	2	Player 1 - pit
Player 1 - pit	3	Player 1 - pit
Player 1 - pit	4	Player 1 - pit
Player 1 - pit	5	Player 1 - pit
Player 1 - kalaha	6	Player 1 - pit
Player 2 - pit	7	Player 2 - kalaha
Player 2 - pit	8	Player 2 - pit
Player 2 - pit	9	Player 2 - pit
Player 2 - pit	10	Player 2 - pit
Player 2 - pit	11	Player 2 - pit
Player 2 - pit	12	Player 2 - pit
Player 2 - kalaha	13	Player 2 - pit

Creating the `game-array`

```
export const createPits = (): Pit[] => {
    let pits: Pit[] = [];

    // Creates player 1 pitstack with 4 balls in each
    for (let i = 0; i < 6; i++) {
      pits.push(createPit({debugName: "player 1 pit: " + i, index: i }));
    };

    // Creates player 1 kalaha
    pits.push(createPit({ isKalaha: true, debugName: "player 1", index: 6 }));


    // Creates player 2 pitstack with 4 balls in each
    for (let i = 0; i < 6; i++) {
      pits.push(createPit({debugName: "player 2 pit: " + i, index: i + 7 }));
    }

    // Creates player 2 kalaha
    pits.push(createPit({ isKalaha: true, debugName: "Player 2", index: 13 }));

    return pits;
  };
```

Creating the `UI-Array`

```
const player1 = pits.slice(6, 7);
  const player1Pits = pits.slice(0, 6).toReversed();

  const player2 = pits.slice(13);
  const player2Pits = pits.slice(7, 13);

  const uiArray =[...player1, ...player1Pits, ...player2, ...player2Pits];
```

We are reversing the player1Pits because that is how the board is handled when going counter clockwise.

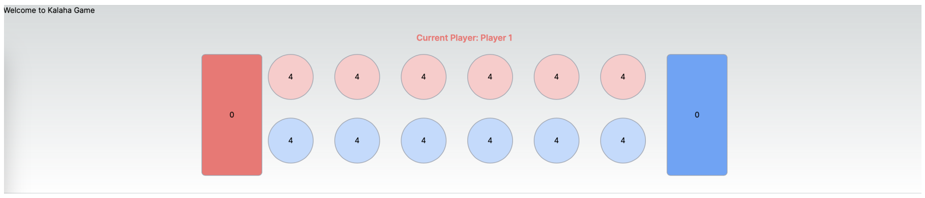

Figure 1: kalaha_gamestart.png

The algorithm we have used

Each time client clicks a `pit` we execute `makeMove()`, which checks if the client draw is valid: not click opponents `pit` or any `kalaha pit`. Then the amount of balls will be forwarded in the next pits, and when all the balls have been addded the algorithm will check, if the last pit inserted into contains more than 1 ball. If more than one ball is placed in the last inserted pit and it's not a `kalaha pit` then a `recursive` call is made to `makeMove()` else the current player will be toggled.

```
if (endPit.seeds > 1 && !endPit.isKalaha) {
        makeMove(game, endPit, false);
    } else {
        game.currentPlayer = currentPlayer === 1 ? 2 : 1;
    }
```

AlphaBeta Algorithm implementation

We added an AI player using the AlphaBeta Algorithm.

Strategy

When it's the AI's turn to make a move, the first step involves cloning the current game state. This clone acts as a sandbox, allowing the AI to explore and simulate potential moves without affecting the actual game. It's a clever way to let the AI "think ahead" by testing out different strategies in a controlled environment.

Decision making

Within this cloned state, the AI evaluates all possible legal moves, essentially asking, "What's the best move here?" This process isn't about random guesses; it's a calculated exploration of options to find the most strategically sound move. Once the AI selects the optimal move based on its calculations, we update the real game state accordingly, handing the turn back to the human player.

It's a bit stupid

At the moment of writing this, the algortihm is not really that smart. Right now it is only taking the Player 1 and Player 2 kalaha pits into account, and growing that as big as possible. To make it smarter, it would be prudent to add some evaluation about the amount of seeds in the other pits as well, to make it take longer turns and make it harder for the human player, but that will probably be added later on.

AlphaBeta Basic concepts explained

Alpha is the best value that the maximizer currently can guarantee at that level or above. Beta is the best value that the minimizer currently can guarantee at that level or above.

```
export function alphaBeta(game: Game, depth: number, alpha: number, beta: number, isMaximizi
    if (depth === 0) {
      return evaluateGameState(game);
    }

    const moves = getPossibleMoves(game, isMaximizingPlayer ? 2 : 1);

    if (isMaximizingPlayer) {
      let maxEval = -Infinity;
      for (const move of moves) {
        const newGame = simulateMove(cloneGame(game), move);
        const evaluation = alphaBeta(newGame, depth - 1, alpha, beta, false);
        maxEval = Math.max(maxEval, evaluation);
        alpha = Math.max(alpha, evaluation);
        if (beta <= alpha) break; // Alpha cut-off
      }
      return maxEval;

    } else {
      let minEval = Infinity;
      for (const move of moves) {
        const newGame = simulateMove(cloneGame(game), move);
        const evaluation = alphaBeta(newGame, depth - 1, alpha, beta, true);
        minEval = Math.min(minEval, evaluation);
        beta = Math.min(beta, evaluation);
        if (beta <= alpha) break; // Beta cut-off
      }
      return minEval;
    }
  }
```

Markdown to PDF

Relying on `pandoc` using `basictex` which can be used as the underlying LaTeX engine for generating PDFs from Markdown or other input formats. Installed both `pandoc` and `basictex` through `homebrew` for mac.

```
# Run the following to convert markdown to pdf
pandoc <markdown_filepath> -o <output_file_name.pdf>
```