Checkers

By Patrick and Sid

Project's Tools

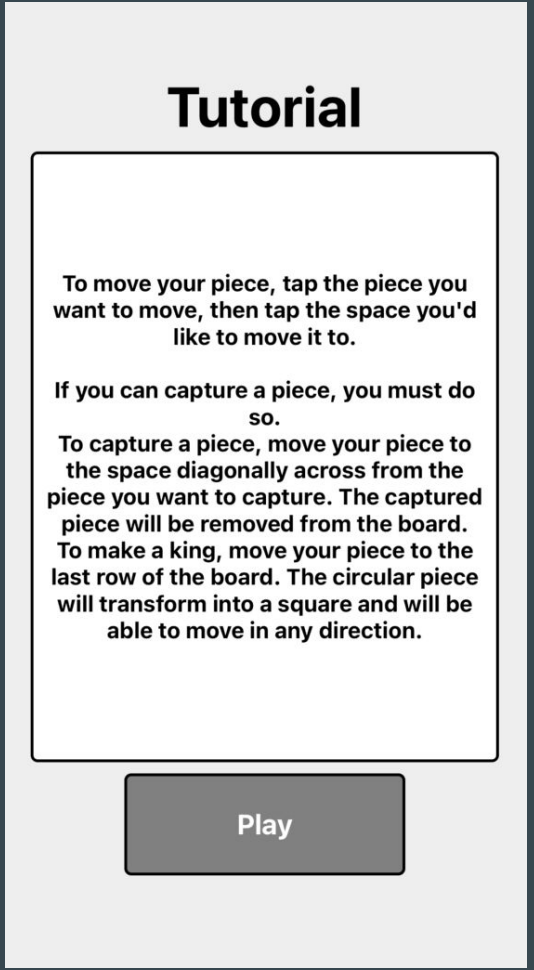
- Visual Studio Code: main programming interface
- GitHub / Git: sync changes across computers
- External websites such as Stack Overflow for guidance on issues

Home Screen

Allows user to pick between the two different game modes, as well as go to the customization screen.

Tutorial Screen

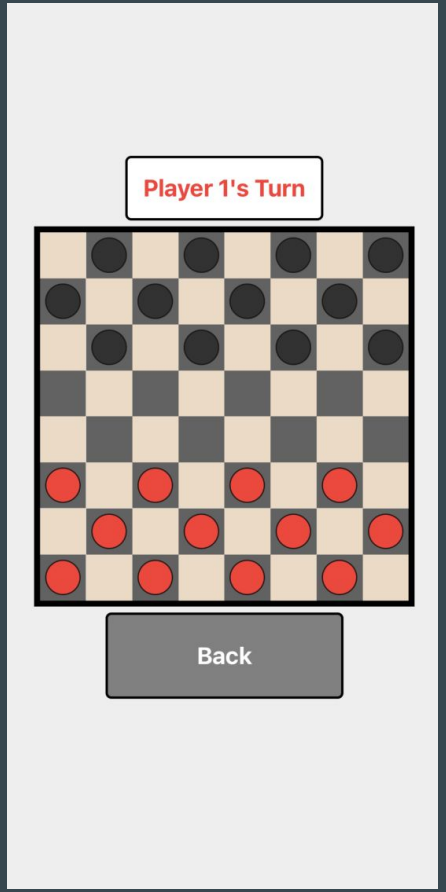
The first time a player selects a Checkers gamemode, a tutorial screen pops up, informing the player of the basic rules of the game.



Checkers Screen (Player vs. Player or AI)

Player 1 selects their piece and where to move it to, and the game moves their piece from there.

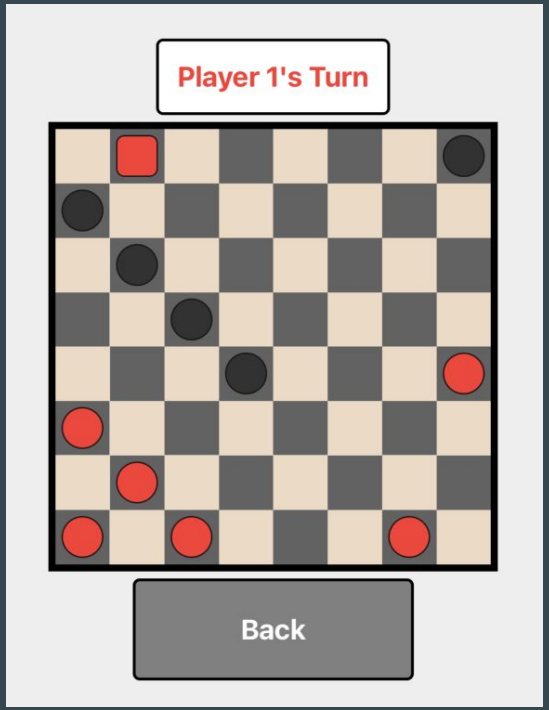
If the mode is Player vs. Player, Player 2 selects their piece and where to move it to, just like Player 1.
If the mode is Player vs. AI, the AI will automatically make a move after a short amount of time.



Checkers Screen (Player vs. Player or AI) - cont.

Once a player reaches the opposite side of their board, their piece transforms into a square piece, becoming a “King” piece.

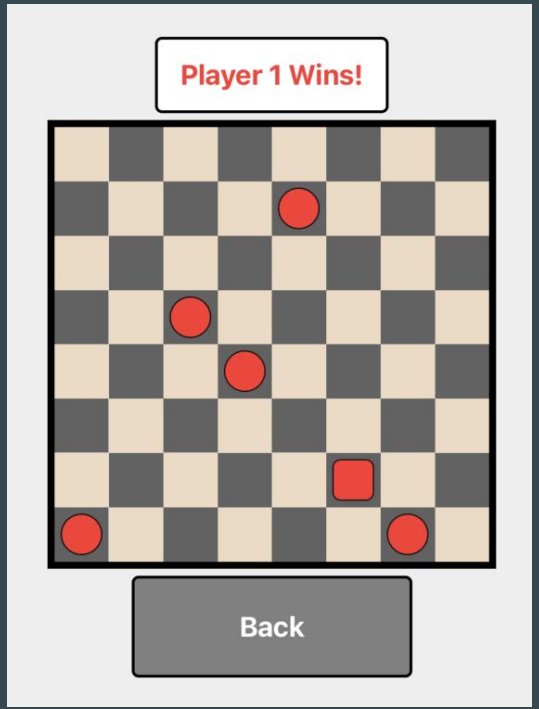
King pieces can move in any direction diagonally.



Checkers Screen (Player vs. Player or AI) - cont.

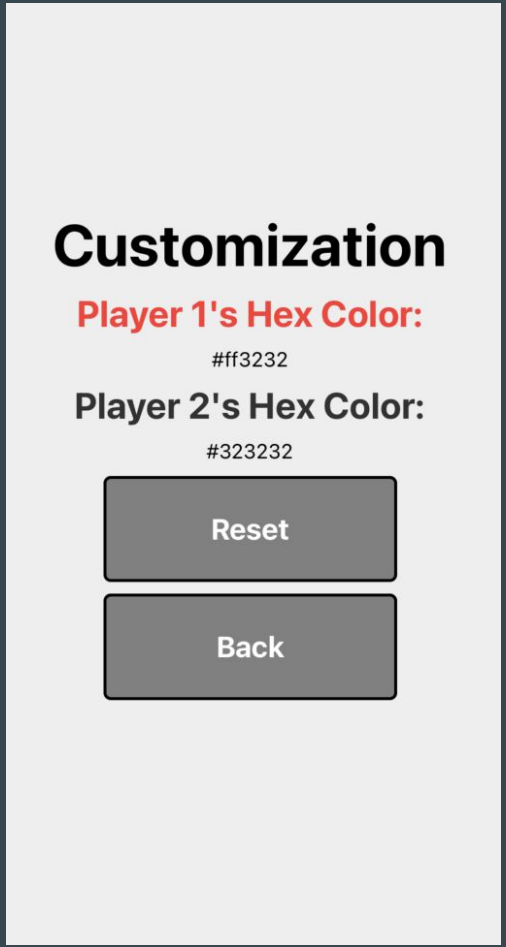
Once a player's opponent has no valid moves left (due to no pieces on the board or being blocked), the player wins!

No further moves can be made.



Customization Screen

Simple color customization screen, allowing the user to set (and reset) the hex color code of the pieces.



Function Highlight: isValidMove

This function helps the game determine if the user's inputted move is valid.

It uses several helper functions that will be shown in the next slide.

```
// Checks if the move is valid
isValidMove = (row1, col1, row2, col2, board, silent=false) => {
  if (this.state.winner != null) {
    // If there's a winner, no moves are valid
    if (!silent) console.log("Valid checker: Game is already over");
    return false;
  }

  if (!this.isSpaceAvailable(row2, col2, board)) {
    if (!silent) console.log("Valid checker: Space isn\'t available");
    return false;
  }
  if (!this.isDiagonal(row1, col1, row2, col2)) {
    if (!silent) console.log("Valid checker: Move isn\'t diagonal");
    return false;
  }
  if (!this.isDirectional(row1, col1, row2, board)) {
    if (!silent) console.log("Valid checker: Move doesn\'t match the piece\'s valid directions");
    return false;
  };
  if (!this.isValidDistance(row1, col1, row2, col2, board)) {
    if (!silent) console.log("Valid checker: Move is too far");
    return false;
  }

  return true;
};
```

Function Highlight: isValidMove - cont.

- isSpaceAvailable: checks if the space is available in the first place
- isDiagonal: checks if the move is diagonal
- isDirectional: check if the move matches the piece's direction (red up, black down, king both)
- isValidDistance: checks if the move is a valid distance (1 piece diagonally normally, 2 pieces diagonally if capture)

```
// Checks if the space is available
isSpaceAvailable = (row, col, board) => {
  if (
    row < 0 ||
    row > board.length - 1 ||
    col < 0 ||
    col > board[0].length - 1
  )
    return false;

  return board[row][col] == null;
};

// Checks if the move is diagonal
isDiagonal = (row1, col1, row2, col2) => {
  return Math.abs(row2 - row1) === Math.abs(col2 - col1);
};

// Checks if the move is in the correct direction
isDirectional = (row1, col1, row2, board) => {
  const piece = board[row1][col1];

  if (piece === PIECES.RED) {
    return row2 - row1 <= -1; // Red can only move up
  } else if (piece === PIECES.BLACK) {
    return row2 - row1 >= 1; // Black can only move down
  }

  return Math.abs(row2 - row1) >= 1; // Kings can move up or down
};

// Checks if the move is a valid distance (1 or 2 spaces, depending on
isValidDistance = (row1, col1, row2, col2, board) => {
  return (
    Math.abs(row1 - row2) === 1 ||
    this.hasCapturablePiece(row1, col1, row2, col2, board)
  );
};
```

Function Highlight: applyAIMove

This function helps the game determine and apply the AI's move.

It also uses several helper functions that will be shown in the next slide.

```
// Applies AI move to the board
applyAIMove = (board) => {
  if (this.state.winner != null) return; // If a winner exists, no moves

  const validMoves = this.getAllValidMoves(board, PIECES.BLACK);

  if (validMoves.length === 0) {
    console.log("AI: AI has no valid moves");
    return;
  }

  const bestMove = this.getBestMove(validMoves, board);

  this.updateBoard(
    bestMove.from.row,
    bestMove.from.col,
    bestMove.to.row,
    bestMove.to.col
  );
};
```

Function Highlight: applyAIMove - cont.

- getRandomMove: gets a random move from a list of moves
- getBestMove: determines the best move for the AI to make based on the following order:

  1. Moves that capture another piece

  2. Moves that make the piece into a king

  3. Moves that move vulnerable pieces

  4. Random

- isPieceVulnerable (not displayed): determines if a piece is vulnerable by checking the opponent's capturable moves

```
// Helper function to get a random move from a list of moves
getRandomMove = (moves) => {
  const rand = Math.random();

  return moves[Math.floor(rand * moves.length)];
};

// Returns the best move for the AI to make based on some rules
getBestMove = (moves, board) => {
  const captureMoves = moves.filter((move) => move.hasCapture);

  if (captureMoves.length > 0) { // Prioritize moves that capture pieces
    console.log("AI: AI capturing piece");
    return this.getRandomMove(captureMoves);
  }

  const kingMoves = moves.filter((move) => {
    const piece = board[move.from.row][move.from.col];
    return piece === PIECES.BLACK && move.to.row === 7;
  });

  if (kingMoves.length > 0) { // Prioritize moves that capture king pieces (i
    console.log("AI: AI making king");
    return this.getRandomMove(kingMoves);
  }

  const vulnerableMoves = moves.filter((move) => {
    return this.isPieceVulnerable(move.from.row, move.from.col, board);
  });

  if (vulnerableMoves.length > 0) { // Prioritize moves that move vulnerable
    console.log("AI: AI moving vulnerable piece");
    return this.getRandomMove(vulnerableMoves);
  }

  console.log("AI: AI making normal move");
  return this.getRandomMove(moves); // Make a random move if no special moves
};
```

Misc. notes

Biggest challenge?

- Coding the AI (looked at several different options)
- Setting up VS Code - Sid had to use CodeHS and send Patrick his commits (until the very end)

What we would've added?

- Color picker to the customization screen
- Achievements / points system for customization

Who did what?

- Fairly equal team effort
- Called for a few nights to work on the project and discussed how to plan out code

Thank you!

https://github.com/patricksemler/checkers