```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sabacc Analysis And Learning Tool</title>
  <script src="./js/cards.js"></script>
  <script src="./js/cardsfunctions.js"></script>
  <style>
    @font-face {
      font-family: 'StarJedi';
      src: url('./fonts/Starjedi.ttf');
    }
    .star-wars-title {
      font-family: 'StarJedi', sans-serif;
      font-size: 20px;
      color: #000000;
      text-align: center;
      letter-spacing: 1px;
      border: 8px solid #000000;
      padding: 6px;
      border-radius: 15px;
      display: inline-block; /* Set div width to fit the content */
      position: fixed; /* Fix the div position on the page */
      z-index: 9999; /* Make it overlap other elements */
      top: 0; /* Position div at the top */
      left: 50%; /* Center horizontally */
      transform: translateX(-50%); /* Adjust the horizontal centering */
      background-color: #d8d8d8; /* Set the background color */
      white-space: nowrap; /* Prevent text wrapping */
    }
    body {
      background-color: #d8d8d8;
      color: #333333;
      /* Changed to dark grey text */
      font-family: 'Arial', sans-serif;
    }

    button {
    font-family: 'StarJedi', sans-serif;
    font-size: 10px;
    color: #000000;
    background-color: #d8d8d8;
    letter-spacing: 1px;
    border: 3px solid #000000;
    padding: 7px;
    border-radius: 5px;
```

```
    cursor: pointer;
    margin: 2px;
  }

    .star-wars-dropdown {
    font-family: 'StarJedi', sans-serif;
    font-size: 10px;
    color: #000000;
    background-color: #d8d8d8;
    letter-spacing: 1px;
    border: 8px solid #000000;
    padding: 7px;
    border-radius: 5px;
    appearance: none; /* Remove default dropdown styling */
    -webkit-appearance: none; /* Remove default dropdown styling for WebKit browsers
*/
    -moz-appearance: none; /* Remove default dropdown styling for Mozilla browsers */
    cursor: pointer;
    margin: 2px;
    }

    button:hover {
      background-color: #6e6e6e;
      border-color: #b22222;
      /* Keep border color dark red on hover */
    }

    .modal {
      display: none;
      position: fixed;
      z-index: 1;
      left: 0;
      top: 0;
      width: 100%;
      height: 100%;
      overflow: auto;
      background-color: rgba(0, 0, 0, 0.4);
    }

    .modal-content {
      background-color: #f0f0f0;
      /* Changed to a really light grey */
      margin: 15% auto;
      padding: 20px;
      border: 1px solid #b22222;
      /* Changed border color to dark red */
      width: 80%;
      border-radius: 3px;
```

```
}

.card {
  width: 52px;
  height: 80px;
  /* Increase the height value */
  background-size: cover;
  border-radius: 5px;
}


.container {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.deck-discard-container {
  display: flex;
  gap: 20px;
  margin-bottom: 20px;
}

.deck {
  margin-bottom: 20px;
  cursor: pointer;
  background-size: cover;
  width: 52px;
  height: 80px;
}

.discard-pile {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
}

.player-area {
  display: flex;
  flex-wrap: wrap;
}

.player {
  margin: 10px;
  padding: 20px;
  border: 4px solid #add8e6;
  /* Changed border color to light blue */
```

```
      border-radius: 5px;
    }

    .player-cards {
      display: flex;
      flex-wrap: wrap;
      gap: 10px;
      padding-right: 10px;
      /* Add padding to the right */
      padding-bottom: 10px;
      /* Add padding to the bottom */
    }


    .page-wrapper {
      border: 8px solid #000000;
      border-radius: 8px;
      padding: 20px;
      max-width: 1200px;
      /* Adjust this value according to your preference */
      margin: 20px auto;
    }
  </style>

</head>

<body>
  <div class="star-wars-title">
    Sabacc Analysis And Learning Tool
  </div>
  <div class="page-wrapper">
    <div class="container">
      <div class="deck-discard-container">
        <div class="deck">
          <p>Deck:</p>
          <div class="deck" onclick="showPlayerSelection()" style="background-image:
url('./images/deckback.png');">
          </div>
        </div>
        <div class="discard-pile">
          <p>Discard Pile:</p>
          <div class="card" onclick="drawFromDiscardPile()" style="background-
image:;"></div>
        </div>
      </div><!-- discard container end-->


      <button onclick="addPlayer()">Add Player</button>
```

```
    <div class="player-area"></div>
  </div>
  <hr>
  <button onclick="discardFromDeck()">Discard Top Card From Deck</button>
  <br>
  <button onclick="openAssignModal()">Manually Assign Card to Player</button>
  <br>
  <br>
  <select class="star-wars-dropdown" id="player-id-dropdown">
    <option value="" disabled selected>Select Player</option>
  </select>
  <button onclick="openModal()">Show Player Perspective</button>
  <br>
  <br>
  <button onclick="checkDecks()">Check Table</button>
  <div id="deck-info"></div>
  <script>
    let players = [];
    let deck = [...cardsData.cards];

    function shuffle(array) {
      for (let i = array.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [array[i], array[j]] = [array[j], array[i]];
      }
    }

    function addPlayer() {
      const playerArea = document.querySelector('.player-area');
      const playerIndex = players.length;
      const playerId = generateUniqueId();

      players.push({
        id: playerId,
        cards: []
      });

      const playerDiv = document.createElement('div');
      playerDiv.classList.add('player');
      playerDiv.innerHTML = `
      <p>Player ${playerIndex + 1} (<span>${playerId}</span>)</p>
      <div class="player-cards" id="player-${playerIndex}-cards"></div>
    `;

      playerArea.appendChild(playerDiv);

      // Update dropdown
      const playerIdDropdown = document.getElementById("player-id-dropdown");
```

```
        const option = document.createElement("option");
        option.text = `Player ${playerIndex + 1} (${playerId})`;
        option.value = playerId; // Set the value as a string instead of a number
        playerIdDropdown.add(option);
      }




      function showPlayerSelection() {
        const currentPlayerIndex = players.length - 1;
        if (currentPlayerIndex < 0) {
          alert("Add a player first!");
          return;
        }

        const selectedPlayerIndex = prompt("Enter the player number to draw a card:",
"");
        if (selectedPlayerIndex === null || selectedPlayerIndex === "") {
          return;
        }

        if (selectedPlayerIndex < 1 || selectedPlayerIndex > players.length) {
          alert("Invalid player number!");
          return;
        }

        drawCard(selectedPlayerIndex - 1);
      }

      function drawCard(playerIndex) {
        if (deck.length === 0) {
          alert("No more cards in the deck!");
          return;
        }

        shuffle(deck);
        const drawnCard = deck.pop();
        players[playerIndex].cards.push(drawnCard); // Update this line
        placeCard(playerIndex, drawnCard);
      }


      function placeCard(playerIndex, card) {
        const playerCards = document.querySelector(`#player-${playerIndex}-cards`);
        const cardDiv = document.createElement('div');
        cardDiv.classList.add('card');
        cardDiv.style.backgroundImage = `url(${card.image})`;
```

```
        cardDiv.onclick = () => {
          discardCard(card);
          playerCards.removeChild(cardDiv);
          const cardIndex = players[playerIndex].cards.indexOf(card); // Update this
line
          players[playerIndex].cards.splice(cardIndex, 1); // Update this line
        };
        playerCards.appendChild(cardDiv);
      }


      let discardPile = [];

      function discardCard(card) {
        const discardPileCard = document.querySelector('.discard-pile .card');
        discardPile.push(card);
        discardPileCard.style.backgroundImage = `url(${card.image})`;
      }

      function drawFromDiscardPile() {
        if (discardPile.length === 0) {
          alert("No more cards in the discard pile!");
          return;
        }

        const currentPlayerIndex = players.length - 1;
        if (currentPlayerIndex < 0) {
          alert("Add a player first!");
          return;
        }

        const selectedPlayerIndex = prompt("Enter the player number to draw from the
discard pile:", "");
        if (selectedPlayerIndex === null || selectedPlayerIndex === "") {
          return;
        }

        if (selectedPlayerIndex < 1 || selectedPlayerIndex > players.length) {
          alert("Invalid player number!");
          return;
        }

        const card = discardPile.pop(); // Not sure about pop
        players[selectedPlayerIndex - 1].cards.push(card); // Update this line
        placeCard(selectedPlayerIndex - 1, card);

        const discardPileCard = document.querySelector('.discard-pile .card');
        //id the next discardpile card by the last in the list
```

```
        let discardNext = discardPile.length > 0 ? (discardPile.length - 1) : 0;
        discardPileCard.style.backgroundImage = discardPile.length > 0 ?
`url(${discardPile[discardNext].image})` :
          "none";
      }




      shuffle(deck);
      //document.querySelector('.discard-pile .card').style.backgroundImage =
`url(${deck[0].image})`;

      // extra functions
      function checkDecks() {
  const deckInfo = document.querySelector("#deck-info");
  const deckSize = deck.length;
  const discardPileSize = discardPile.length;

  let playerData = [];

  players.forEach((player, index) => {
    const totalValue = player.cards.reduce((sum, card) => sum + parseInt(card.value),
0);
    const cardNumber = player.cards.length;
    const positiveValueTotal = player.cards.reduce((sum, card) => sum +
(parseInt(card.value) > 0 ? parseInt(card.value) : 0), 0);
    const highestPositiveCard = Math.max(...player.cards.map(card =>
(parseInt(card.value) > 0 ? parseInt(card.value) : 0)));
    const [handName, handDesc, rank] = getHandName(player.cards);

    playerData.push({
      index,
      playerId: player.id,
      handName,
      handDesc,
      rank,
      totalValue,
      absValue: Math.abs(totalValue),
      cardNumber,
      positiveValueTotal,
      highestPositiveCard,
    });
  });

  // Sort playerData by rank, absolute value of totalValue, highest positive
totalValue, highest number of cards, highest total of positive cards, and then highest
positive card.
```

```
  playerData.sort((a, b) => {
    if (a.rank !== b.rank) {
      return a.rank - b.rank;
    }
    if (a.absValue !== b.absValue) {
      return a.absValue - b.absValue;
    }
    if (a.totalValue !== b.totalValue) {
      return b.totalValue - a.totalValue;
    }
    if (a.cardNumber !== b.cardNumber) {
      return b.cardNumber - a.cardNumber;
    }
    if (a.positiveValueTotal !== b.positiveValueTotal) {
      return b.positiveValueTotal - a.positiveValueTotal;
    }
    return b.highestPositiveCard - a.highestPositiveCard;
  });

  // The winning player is the first item in the sorted playerData array.
  const winningPlayerData = playerData[0];

  // Check for a complete tie.
  const completeTie = playerData.slice(1).some(player => player.rank ===
winningPlayerData.rank && player.absValue === winningPlayerData.absValue &&
player.totalValue === winningPlayerData.totalValue && player.cardNumber ===
winningPlayerData.cardNumber && player.positiveValueTotal ===
winningPlayerData.positiveValueTotal && player.highestPositiveCard ===
winningPlayerData.highestPositiveCard);

  // Generate the playerValues array containing player information strings.
  const playerValues = playerData.map(player =>
    `Player ${player.index + 1} (${player.playerId}): ${player.handName} (Desc:
${player.handDesc}, Rank: ${player.rank}, Value: ${player.totalValue})`
  );

  // Update the winning player information string.
  const winningPlayer = completeTie ? "Complete Tie" : `Player
${winningPlayerData.index + 1} (${winningPlayerData.playerId})`;

  // Determine the tiebreaker reason.
  let tiebreakerReason = '';
  if (!completeTie) {
    if (playerData[0].rank !== playerData[1].rank) {
      tiebreakerReason = ' (Tiebreaker: Rank)';
    } else if (playerData[0].absValue !== playerData[1].absValue) {
      tiebreakerReason = ' (Tiebreaker: Absolute Value)';
    } else if (playerData[0].totalValue !== playerData[1].totalValue) {
```

```
tiebreakerReason = ' (Tiebreaker: Highest Positive Total Value)';
} else if (playerData[0].cardNumber !== playerData[1].cardNumber) {
tiebreakerReason = ' (Tiebreaker: Highest Number of Cards)';
} else if (playerData[0].positiveValueTotal !== playerData[1].positiveValueTotal) {
tiebreakerReason = ' (Tiebreaker: Highest Total of Positive Cards)';
} else if (playerData[0].highestPositiveCard !== playerData[1].highestPositiveCard) {
tiebreakerReason = ' (Tiebreaker: Highest Positive Card)';
}
}

deckInfo.innerHTML = `

<p>Cards in deck: ${deckSize}</p>
<p>Cards in discard pile: ${discardPileSize}</p>
<p>Total values and hands of players' hands:</p>
<ul>
  ${playerValues.map(value => `<li>${value}</li>`).join('')}
</ul>
<p>Winning hand: ${completeTie ? 'Complete Tie' : `${winningPlayer} with a
${winningPlayerData.handName} (Desc: ${winningPlayerData.handDesc}, Rank:
${winningPlayerData.rank}) and a total value of
${winningPlayerData.totalValue}${tiebreakerReason}`}</p>
`;
}



      function discardFromDeck() {
        if (deck.length === 0) {
          alert("No more cards in the deck!");
          return;
        }

        shuffle(deck);
        const drawnCard = deck.pop();
        discardPile.push(drawnCard);
        const discardPileCard = document.querySelector('.discard-pile .card');
        discardPileCard.style.backgroundImage = `url(${drawnCard.image})`;
      }

      function generateUniqueId() {
        return Math.floor(Math.random() * (9999999999 - 1000000000) + 1000000000);
      }

      function openModal() {
        const modal = document.getElementById("modal");
        const playerIdDropdown = document.getElementById("player-id-dropdown");
        const playerId = playerIdDropdown.value;
```

```
        const playerIndex = players.findIndex(player => player.id == playerId); // Use
'==' instead of '==='

        if (playerIndex === -1) {
          alert("Player not found!");
          return;
        }

        modal.style.display = "block";

        const modalDeck = document.getElementById("modal-deck");
        modalDeck.style.backgroundImage = `url('./images/deckback.png')`;

        const modalDiscardPile = document.getElementById("modal-discard-
pile").querySelector(".card");
        modalDiscardPile.style.backgroundImage = discardPile.length > 0 ?
          `url(${discardPile[discardPile.length - 1].image})` : "none";

        const modalPlayerId = document.getElementById("modal-player-id");
        modalPlayerId.textContent = playerId;

        const modalPlayerCards = document.getElementById("modal-player-cards");
        modalPlayerCards.innerHTML = "";

        players[playerIndex].cards.forEach(card => {
          const cardDiv = document.createElement("div");
          cardDiv.classList.add("card");
          cardDiv.style.backgroundImage = `url(${card.image})`;
          modalPlayerCards.appendChild(cardDiv);
        });

        let otherPlayersContainer = document.getElementById("other-players-
container");
        if (!otherPlayersContainer) {
          otherPlayersContainer = document.createElement("div");
          otherPlayersContainer.id = "other-players-container";
          otherPlayersContainer.classList.add("other-players-container");
          modalPlayerCards.insertAdjacentElement("afterend", otherPlayersContainer);
        } else {
          otherPlayersContainer.innerHTML = "";
        }

        players.forEach((player, index) => {
          if (index !== playerIndex) {
            const playerDiv = document.createElement("div");
            playerDiv.classList.add("player");
```

```
            const playerTitle = document.createElement("p");
            playerTitle.textContent = `Player ${index + 1} (${player.id})`;
            playerDiv.appendChild(playerTitle);

            const playerCards = document.createElement("div");
            playerCards.classList.add("player-cards");

            player.cards.forEach(() => {
              const cardDiv = document.createElement("div");
              cardDiv.classList.add("card");
              cardDiv.style.backgroundImage = `url('./images/deckback.png')`;
              playerCards.appendChild(cardDiv);
            });

            playerDiv.appendChild(playerCards);
            otherPlayersContainer.appendChild(playerDiv);
          }
        });

        modalPlayerCards.insertAdjacentElement("afterend", otherPlayersContainer);
      }



      function openAssignModal() {
        const assignModal = document.getElementById("assignModal");
        const assignPlayerIdDropdown = document.getElementById("assign-player-id-
dropdown");
        const assignDeck = document.getElementById("assign-deck");

        assignPlayerIdDropdown.innerHTML = "";
        const defaultOption = document.createElement("option");
        defaultOption.value = "";
        defaultOption.disabled = true;
        defaultOption.selected = true;
        defaultOption.textContent = "Select Player";
        assignPlayerIdDropdown.add(defaultOption);

        players.forEach((player, index) => {
          const option = document.createElement("option");
          option.text = `Player ${index + 1} (${player.id})`;
          option.value = player.id;
          assignPlayerIdDropdown.add(option);
        });

        assignDeck.innerHTML = "";

        deck.forEach(card => {
```

```
          const cardDiv = document.createElement("div");
          cardDiv.classList.add("card");
          cardDiv.style.backgroundImage = `url(${card.image})`;
          cardDiv.addEventListener("click", () => {
            const playerId = assignPlayerIdDropdown.value;

            if (!playerId) {
              alert("Select a player first!");
              return;
            }

            const playerIndex = players.findIndex(player => player.id == playerId);

            if (playerIndex === -1) {
              alert("Player not found!");
              return;
            }

            const cardIndex = deck.indexOf(card);
            deck.splice(cardIndex, 1);
            players[playerIndex].cards.push(card);
            placeCard(playerIndex, card);

            assignDeck.removeChild(cardDiv);
          });
          assignDeck.appendChild(cardDiv);
        });

        assignModal.style.display = "block";
      }
    </script>

    <div id="modal"
      style="display:none; position: fixed; z-index: 1; left: 0; top: 0; width: 100%;
height: 100%; overflow: auto; background-color: rgba(0,0,0,0.4);">
      <div style="background-color: #fefefe; margin: 15% auto; padding: 20px; border:
1px solid #888; width: 80%;">
        <h2>Player's Cards (<span id="modal-player-id"></span>)</h2>
        <div id="modal-deck" class="deck" style="margin-bottom: 20px;"></div>
        <div id="modal-discard-pile" class="discard-pile">
          <p>Discard Pile:</p>
          <div class="card"></div>
        </div>
        <div id="modal-player-cards" style="display: flex; flex-wrap: wrap; gap:
10px;"></div>
        <button style="display: block; margin-top: 20px;"

onclick="document.getElementById('modal').style.display='none'">Close</button>
```

```
      </div>
    </div>

    <div id="assignModal"
      style="display:none; position: fixed; z-index: 1; left: 0; top: 0; width: 100%;
height: 100%; overflow: auto; background-color: rgba(0,0,0,0.4);">
      <div style="background-color: #fefefe; margin: 15% auto; padding: 20px; border:
1px solid #888; width: 80%;">
        <h2>Assign Card to Player</h2>
        <label for="assign-player-id-dropdown">Select Player:</label>
        <select class="star-wars-dropdown" id="assign-player-id-dropdown">
          <option value="" disabled selected>Select Player ID</option>
        </select>
        <div id="assign-deck" style="display: flex; flex-wrap: wrap; gap: 10px;
margin-top: 20px;"></div>
        <button style="display: block; margin-top: 20px;"

onclick="document.getElementById('assignModal').style.display='none'">Close</button>
      </div>
    </div>



  </div><!-- page wrapper-->
  <br>
  <br>
  Content used under Fair Use for educational purposes, promoting understanding of the
card game and its rules without commercial intent. Hand rankings are based on the
Outer Rim Sabacc League Hand Rankings.  Created by Stacknsleuth in order to learn the
Sabacc hands.
</body>

</html>
```

```
const cardsData = {
    "cards": [
      {
        "stave": "triangle",
        "value": "-1",
        "id": "triangle-1",
        "image": "./images/triangle-1.png"
      },{
        "stave": "triangle",
        "value": "-2",
        "id": "triangle-2",
        "image": "./images/triangle-2.png"
```

```
    },{
      "stave": "triangle",
      "value": "-3",
      "id": "triangle-3",
      "image": "./images/triangle-3.png"
    },{
      "stave": "triangle",
      "value": "-4",
      "id": "triangle-4",
      "image": "./images/triangle-4.png"
    },{
      "stave": "triangle",
      "value": "-5",
      "id": "triangle-5",
      "image": "./images/triangle-5.png"
    },{
      "stave": "triangle",
      "value": "-6",
      "id": "triangle-6",
      "image": "./images/triangle-6.png"
    },{
      "stave": "triangle",
      "value": "-7",
      "id": "triangle-7",
      "image": "./images/triangle-7.png"
    },{
      "stave": "triangle",
      "value": "-8",
      "id": "triangle-8",
      "image": "./images/triangle-8.png"
    },{
      "stave": "triangle",
      "value": "-9",
      "id": "triangle-9",
      "image": "./images/triangle-9.png"
    },{
      "stave": "triangle",
      "value": "-10",
      "id": "triangle-10",
      "image": "./images/triangle-10.png"
    },{
      "stave": "triangle",
      "value": "1",
      "id": "triangle1",
      "image": "./images/triangle1.png"
    },{
      "stave": "triangle",
      "value": "2",
```

```
      "id": "triangle2",
      "image": "./images/triangle2.png"
    },{
      "stave": "triangle",
      "value": "3",
      "id": "triangle3",
      "image": "./images/triangle3.png"
    },{
      "stave": "triangle",
      "value": "4",
      "id": "triangle4",
      "image": "./images/triangle4.png"
    },{
      "stave": "triangle",
      "value": "5",
      "id": "triangle5",
      "image": "./images/triangle5.png"
    },{
      "stave": "triangle",
      "value": "6",
      "id": "triangle6",
      "image": "./images/triangle6.png"
    },{
      "stave": "triangle",
      "value": "7",
      "id": "triangle7",
      "image": "./images/triangle7.png"
    },{
      "stave": "triangle",
      "value": "8",
      "id": "triangle8",
      "image": "./images/triangle8.png"
    },{
      "stave": "triangle",
      "value": "9",
      "id": "triangle9",
      "image": "./images/triangle9.png"
    },{
      "stave": "triangle",
      "value": "10",
      "id": "triangle10",
      "image": "./images/triangle10.png"
    },{
      "stave": "square",
      "value": "-1",
      "id": "square-1",
      "image": "./images/square-1.png"
    },{
```

```
      "stave": "square",
      "value": "-2",
      "id": "square-2",
      "image": "./images/square-2.png"
    },{
      "stave": "square",
      "value": "-3",
      "id": "square-3",
      "image": "./images/square-3.png"
    },{
      "stave": "square",
      "value": "-4",
      "id": "square-4",
      "image": "./images/square-4.png"
    },{
      "stave": "square",
      "value": "-5",
      "id": "square-5",
      "image": "./images/square-5.png"
    },{
      "stave": "square",
      "value": "-6",
      "id": "square-6",
      "image": "./images/square-6.png"
    },{
      "stave": "square",
      "value": "-7",
      "id": "square-7",
      "image": "./images/square-7.png"
    },{
      "stave": "square",
      "value": "-8",
      "id": "square-8",
      "image": "./images/square-8.png"
    },{
      "stave": "square",
      "value": "-9",
      "id": "square-9",
      "image": "./images/square-9.png"
    },{
      "stave": "square",
      "value": "-10",
      "id": "square-10",
      "image": "./images/square-10.png"
    },{
      "stave": "square",
      "value": "1",
      "id": "square1",
```

```
      "image": "./images/square1.png"
    },{
      "stave": "square",
      "value": "2",
      "id": "square2",
      "image": "./images/square2.png"
    },{
      "stave": "square",
      "value": "3",
      "id": "square3",
      "image": "./images/square3.png"
    },{
      "stave": "square",
      "value": "4",
      "id": "square4",
      "image": "./images/square4.png"
    },{
      "stave": "square",
      "value": "5",
      "id": "square5",
      "image": "./images/square5.png"
    },{
      "stave": "square",
      "value": "6",
      "id": "square6",
      "image": "./images/square6.png"
    },{
      "stave": "square",
      "value": "7",
      "id": "square7",
      "image": "./images/square7.png"
    },{
      "stave": "square",
      "value": "8",
      "id": "square8",
      "image": "./images/square8.png"
    },{
      "stave": "square",
      "value": "9",
      "id": "square9",
      "image": "./images/square9.png"
    },{
      "stave": "square",
      "value": "10",
      "id": "square10",
      "image": "./images/square10.png"
    },{
      "stave": "circle",
```

```
      "value": "-1",
      "id": "circle-1",
      "image": "./images/circle-1.png"
    },{
      "stave": "circle",
      "value": "-2",
      "id": "circle-2",
      "image": "./images/circle-2.png"
    },{
      "stave": "circle",
      "value": "-3",
      "id": "circle-3",
      "image": "./images/circle-3.png"
    },{
      "stave": "circle",
      "value": "-4",
      "id": "circle-4",
      "image": "./images/circle-4.png"
    },{
      "stave": "circle",
      "value": "-5",
      "id": "circle-5",
      "image": "./images/circle-5.png"
    },{
      "stave": "circle",
      "value": "-6",
      "id": "circle-6",
      "image": "./images/circle-6.png"
    },{
      "stave": "circle",
      "value": "-7",
      "id": "circle-7",
      "image": "./images/circle-7.png"
    },{
      "stave": "circle",
      "value": "-8",
      "id": "circle-8",
      "image": "./images/circle-8.png"
    },{
      "stave": "circle",
      "value": "-9",
      "id": "circle-9",
      "image": "./images/circle-9.png"
    },{
      "stave": "circle",
      "value": "-10",
      "id": "circle-10",
      "image": "./images/circle-10.png"
```

```
    },{
      "stave": "circle",
      "value": "1",
      "id": "circle1",
      "image": "./images/circle1.png"
    },{
      "stave": "circle",
      "value": "2",
      "id": "circle2",
      "image": "./images/circle2.png"
    },{
      "stave": "circle",
      "value": "3",
      "id": "circle3",
      "image": "./images/circle3.png"
    },{
      "stave": "circle",
      "value": "4",
      "id": "circle4",
      "image": "./images/circle4.png"
    },{
      "stave": "circle",
      "value": "5",
      "id": "circle5",
      "image": "./images/circle5.png"
    },{
      "stave": "circle",
      "value": "6",
      "id": "circle6",
      "image": "./images/circle6.png"
    },{
      "stave": "circle",
      "value": "7",
      "id": "circle7",
      "image": "./images/circle7.png"
    },{
      "stave": "circle",
      "value": "8",
      "id": "circle8",
      "image": "./images/circle8.png"
    },{
      "stave": "circle",
      "value": "9",
      "id": "circle9",
      "image": "./images/circle9.png"
    },{
      "stave": "circle",
      "value": "10",
```

```
        "id": "circle10",
        "image": "./images/circle10.png"
      },{
        "stave": "sylop",
        "value": "0",
        "id": "sylop1",
        "image": "./images/sylop1.png"
      },{
        "stave": "sylop",
        "value": "0",
        "id": "sylop2",
        "image": "./images/sylop2.png"
      }
    ]
  };
```

```
// Define a function named getHandName that takes a single parameter, cards
function getHandName(cards) {
    //console.log(JSON.stringify(cards))
    let checkpuresabacc = ""
    let checkfullsabacc = ""
    let checkfleet = ""
    let checkyeehaa = ""
    let checkrhylet = ""
    let checksquadron = ""
    let checkgeewhiz = ""
    let checkstraightkhyron = ""
    let checkbanthaswild = ""
    let checkruleoftwo = ""
    let checkpairedsabacc = ""
    // Calculate the total value of the cards by using reduce to sum the integer
values of each card
    const totalValue = cards.reduce((sum, card) => sum + parseInt(card.value), 0);
    // Get the number of cards in the hand
    const cardCount = cards.length;
    // Filter out the zero-valued cards and non-zero valued cards into separate arrays
    const zeroCards = cards.filter(card => card.value === "0");
    const nonZeroCards = cards.filter(card => card.value !== "0");

    // tiebreaker variables
    let cardnumber = cards.length

    //check for pure sabacc
    // Check if there are exactly two cards and their total value is zero and the
number of zero cards is 2
    if (cardCount === 2 && totalValue === 0 && zeroCards.length == 2) {
```

```
        // If true, return a Pure Sabacc hand with a description and a ranking value
        checkpuresabacc = "true"
        return ["Pure Sabacc", "two zero cards", 1];
    }

    // Check if there is exactly one zero card and four non-zero cards
    if (zeroCards.length === 1 && nonZeroCards.length === 4) {
        // Find positive tens (10 and 10) and negative tens (-10 and -10) among the
non-zero cards
        const positiveTens = nonZeroCards.filter(card => card.value === "10");
        const negativeTens = nonZeroCards.filter(card => card.value === "-10");
        // - example filter for two things - const negativeTens =
nonZeroCards.filter(card => card.value === "-10" || card.value === "0");
        // Check if there are exactly two positive tens and two negative tens
        if (positiveTens.length === 2 && negativeTens.length === 2) {
            // If true, return a Full Sabacc hand with a description and a ranking
value
            checkfullsabacc = "true"
            return ["Full Sabacc", "a zero card and four tens (2 positive and 2
negative) to sum all cards to zero", 2];
        }
    }

    const nonZeroValues = nonZeroCards.map(card => card.value);

    function areAllNumbersSame(arr) {
        if (arr.length === 0) {
            return true;
        }
        const firstNumber = Math.abs(parseInt(arr[0]));
        return arr.every((num) => Math.abs(parseInt(num)) === firstNumber);
    }
    const nonZeroCardsSame = areAllNumbersSame(nonZeroValues)

    // Check if there is exactly one zero card and four matching
    if (nonZeroCardsSame == true && zeroCards.length === 1 && nonZeroCards.length ===
4) {
        // Get the rank of each non-zero card
        const fleetpositiveTens = nonZeroCards.filter(card => card.value === "10");
        const fleetnegativeTens = nonZeroCards.filter(card => card.value === "-10");
        if (totalValue == 0 && fleetnegativeTens.length == 0 &&
fleetpositiveTens.length == 0) {
            // If true, return a Fleet hand with a description and a ranking value
            checkfleet = "true"
            return ["Fleet", "a zero card and four of a kind not tens (2 positive and
2 negative) all cards total equals zero", 3];
        }
    }
```

```
    // Check if there is exactly one zero card and two non-zero cards
    if (zeroCards.length === 1 && nonZeroCards.length === 2) {
        if (nonZeroCardsSame == true && totalValue == 0) {
            // If true, return a Yee-Haa hand with a description and a ranking value
            checkyeehaa = "true"
            return ["Yee-Haa", "a zero card and a pair (one positive and one negative)
all cards totalling zero", 4];
        }
    }

    // check for threesame and two same


    function areAllNumbersSameTwoThree(arr) {
        if (arr.length === 0) {
            return true;
        }

        const counts = countOccurrences(arr);
        const isThreeSame = Object.values(counts).some(count => count >= 3);

        // Filter out numbers with a count of 3 or more
        const filteredCounts = Object.fromEntries(Object.entries(counts).filter(([key,
value]) => value < 3));

        // Check for pairs after filtering out 3 or more occurrences
        const isTwoSame = Object.values(filteredCounts).some(count => count >= 2);

        return {
            isThreeSame,
            isTwoSame
        };
    }

    function countOccurrences(arr) {
        const counts = {};

        arr.forEach((num) => {
            const absNum = Math.abs(parseInt(num));
            if (counts[absNum]) {
                counts[absNum]++;
            } else {
                counts[absNum] = 1;
            }
        });

        return counts;
```

```
    }

    const nonZeroCardsSameThreeTwo = areAllNumbersSameTwoThree(nonZeroValues);
    //console.log(nonZeroCardsSameThreeTwo);

    // Check if there is exactly one zero card and two non-zero cards

    if (totalValue == 0 && nonZeroCardsSameThreeTwo.isThreeSame == true &&
nonZeroCardsSameThreeTwo.isTwoSame == true && zeroCards.length === 0 &&
nonZeroCards.length === 5) {
            // If true, return a Yee-Haa hand with a description and a ranking value
            checkrhylet = "true"
            return ["Rhylet", "positive of three of a kind and a negative pair or vice
versa totalling zero", 5];
    }



    // Check if there is 4 of a kind equaling zero
    if (totalValue == 0 && nonZeroCardsSame == true && zeroCards.length === 0 &&
nonZeroCards.length === 4) {
            // If true, return a Squadron hand with a description and a ranking value
            checksquadron = "true"
            return ["Squadron", "Four of a kind, (two positive, two negative) totaling
zero", 6];
    }

    // Check if there is 4 of a kind equaling zero
    if (totalValue == 0 && zeroCards.length === 0 && nonZeroCards.length === 5) {
        // If true, return a Gee Whiz hand with a description and a ranking value
            positiveCardone = nonZeroCards.some(card => card.value == "1");
            positiveCardtwo = nonZeroCards.some(card => card.value == "2");
            positiveCardthree = nonZeroCards.some(card => card.value == "3");
            positiveCardfour = nonZeroCards.some(card => card.value == "4");
            positiveCardten = nonZeroCards.some(card => card.value == "10");
            negativeCardone = nonZeroCards.some(card => card.value === "-1");
            negativeCardtwo = nonZeroCards.some(card => card.value === "-2");
            negativeCardthree = nonZeroCards.some(card => card.value === "-3");
            negativeCardfour = nonZeroCards.some(card => card.value === "-4");
            negativeCardten = nonZeroCards.some(card => card.value === "-10");
        if ((positiveCardone == true && positiveCardtwo == true && positiveCardthree
== true && positiveCardfour == true && negativeCardten == true) || (negativeCardone ==
true && negativeCardtwo == true && negativeCardthree == true && negativeCardfour ==
true && positiveCardten == true)) {
            checkgeewhiz = "true"
            return ["Gee Whiz", " 1 2 3 and 4 and -10 or -1 -2 -3 -4 and 10 totalling
zero", 7];
        }
    }
```

```
    const ZeroNonZeroValues = cards.map(card => card.value);

    // check for sequential
    function isSequential(arr) {
        // Convert string values to numbers and change negative numbers to positive
        const positiveArr = arr.map(Number).map(Math.abs);

        // Sort the array in ascending order
        const sortedArr = positiveArr.sort((a, b) => a - b);

        // Iterate through the sorted array and check if the difference between each
consecutive pair of numbers is 1
        for (let i = 0; i < sortedArr.length - 1; i++) {
          if (sortedArr[i + 1] - sortedArr[i] !== 1) {
            return false;
          }
        }

        return true;
      }

      const areCardsSequential = isSequential(ZeroNonZeroValues)
      //console.log(areCardsSequential)
      //console.log(cards.length)
      //console.log(ZeroNonZeroValues)

    // Check if there is 4 sequential equals zero
    if (totalValue == 0 && areCardsSequential == true && cards.length == 4) {
        // If true, return a Squadron hand with a description and a ranking value
        checkstraightkhyron = "true"
        return ["Straight Khyron", "a sequential run of four cards totaling zero, may
include a sylop", 8];
    }


    //console.log(nonZeroCardsSameThreeTwo.isThreeSame)
    //console.log(nonZeroCardsSameThreeTwo.isTwoSame)
    //console.log(cards.length)
    //console.log(ZeroNonZeroValues)

    if (totalValue == 0 && nonZeroCardsSameThreeTwo.isThreeSame == true &&
nonZeroCardsSameThreeTwo.isTwoSame == false && (cards.length === 4 || cards.length ===
5)) {
        // If true, return a Yee-Haa hand with a description and a ranking value
        checkbanthaswild = "true"
        return ["Banthas Wild", "Three of a kind plus one or two other cards totalling
zero", 9];
```

```
    }

    // check for two distinct pairs
    function checkTwoDistinctPairs(arr) {
        if (arr.length === 0) {
            return false;
        }

        const counts = countAbsoluteOccurrences(arr);
        const filteredCounts = filterCounts(counts, 2);

        // Check for two distinct pairs
        const pairCount = Object.values(filteredCounts).filter(count => count >=
2).length;
        const hasTwoDistinctPairs = pairCount >= 2;

        return hasTwoDistinctPairs;
    }

    function countAbsoluteOccurrences(arr) {
        const counts = {};

        arr.forEach((num) => {
            const absNum = Math.abs(parseInt(num));
            if (counts[absNum]) {
                counts[absNum]++;
            } else {
                counts[absNum] = 1;
            }
        });

        return counts;
    }

    function filterCounts(counts, maxCount) {
        return Object.fromEntries(
            Object.entries(counts).filter(([key, value]) => value <= maxCount)
        );
    }

    const paironeandpairtwo = checkTwoDistinctPairs(ZeroNonZeroValues)

    //console.log(paironeandpairtwo)
    if (totalValue == 0 && paironeandpairtwo == true && (cards.length === 4 ||
cards.length === 5)) {
        // If true, return a Yee-Haa hand with a description and a ranking value
        checkruleoftwo = "true"
```

```
        return ["Rule of Two", "two pairs with a total hand value of zero (may contain a fifth card)", 10];
    }

    const atleastonepair = areAllNumbersSameTwoThree(ZeroNonZeroValues)
    // Check if the total value of the cards is zero and there is a pair and none of the other hands apply
    if (totalValue === 0 && atleastonepair.isTwoSame == true && atleastonepair.isThreeSame == false && checkpuresabacc == "" && checkfullsabacc == "" && checkfleet == "" && checkyeehaa == "" && checkrhylet == "" && checksquadron == "" && checkgeewhiz == "" && checkstraightkhyron == "" && checkbanthaswild == "" && checkruleoftwo == "") {
        // If true, return a Regular Sabacc hand with a description and a ranking value
        checkpairedsabacc = "true"
        return ["Paired Sabacc", "a pair with a total hand value of zero (may or may not contain more cards)", 11];
    }


    // Check if the total value of the cards is zero
    if (totalValue === 0 && checkpuresabacc == "" && checkfullsabacc == "" && checkfleet == "" && checkyeehaa == "" && checkrhylet == "" && checksquadron == "" && checkgeewhiz == "" && checkstraightkhyron == "" && checkbanthaswild == "" && checkruleoftwo == "" && checkpairedsabacc == "") {
        // If true, return a Regular Sabacc hand with a description and a ranking value
        return ["Regular Sabacc", "any combination of cards that sum to equal a total of zero", 12];
    }
    // If none of the above conditions are met, return a Nulrhek hand with a description and a ranking value
    return ["Nulrhek", "any combination of cards that are not equal to zero", 14];

    // extra snippets
    //        // Find a positive and a negative card among the non-zero cards
    //const positiveCard = nonZeroCards.find(card => card.value !== "1" && card.value !== "0"); // finds and returns card object
    //const negativeCard = nonZeroCards.find(card => card.value === "1" || card.value === "0");

}
```