```
/// Gets the winning player for a game
///
/// Returns "X", "O", or "Draw" if a game has ended
///
/// Returns "Neither" if a game has not ended
///
fn get_winning_player(state: GameState) -> String {
  //possible combinations of boxes marked to be a winner
  let lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ]
  case check_lines(lines, state) {
    Neither -> {
      case list.contains(state.state, Neither) {
        True -> "Neither"
        _ -> "Draw"
      }
    }
    player -> {
      case player {
        X -> "X"
        _ -> "O"
      }
    }
  }
}

/// Goes through all possible combinations for getting a three in a row and checks if it exists on the current game grid
///
fn check_lines(lines: List(List(Int)), state: GameState) -> Player {
  case lines {
    [first, ..rest] -> {
      let assert [a, b, c] = first
      let player = get_from_index(state.state, a)
      let res = case player {
        X | O -> {
          case
            player == get_from_index(state.state, b)
            && player == get_from_index(state.state, c)
          {
            True -> {
              player
            }
            _ -> {
              Neither
            }
          }
        }
        _ -> Neither
      }
      case res {
        Neither -> check_lines(rest, state)
        _ -> res
      }
    }
    [] -> Neither
  }
}

/// Helper function to get an item from a list through its index
///
fn get_from_index(list: List(a), index: Int) -> a {
  let assert Ok(last) = list.first(list.split(list, index).1)
  last
}
```