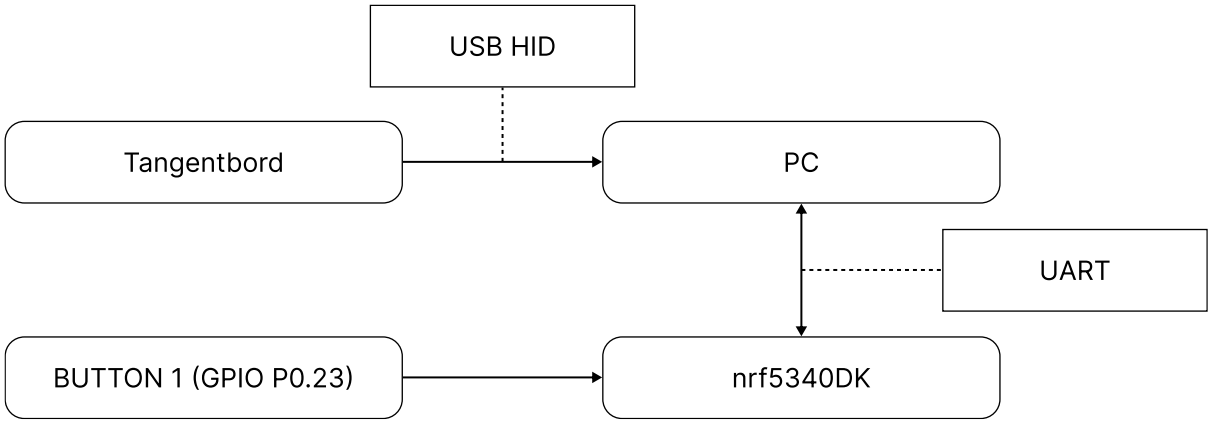
Harley Berglund & William Willén

Chess Game

1 How to play

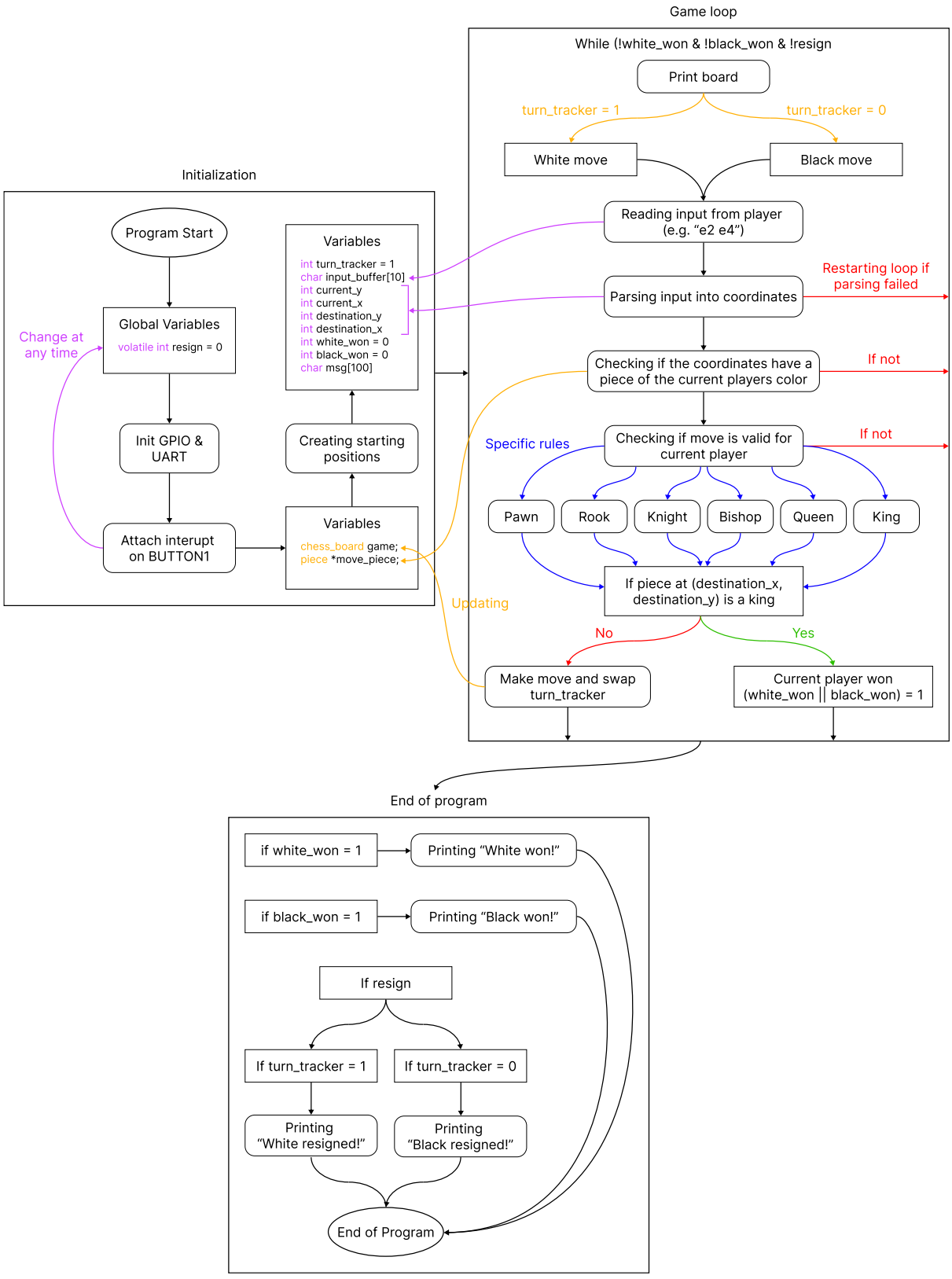
White starts the round. Each player gets to move one piece per round. The goal is to capture the opponents king with one of your pieces. A capture is done by placeing your piece on an opponent's. You can capture any of your opponent's pieces. You can resign with button 1 on the nrf board.

2 System overview



3 Implementation

3.1 Blockdiagram



3.2 Pseudokod

3.2.1 starting_positions()

The starting positions are determined using a matrix with characters representing each piece type. Where capital letters are white pieces and small letters are black pieces.

```
FUNCTION starting_positions(chess_board):

    FOR i = 0 TO 7:
        FOR j = 0 TO 7:
            chess_board[i][j] = NULL
        END FOR
    END FOR

    initial_setup[8][8] =
        [{'r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'},
        {'p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {'P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'},
        {'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'}]

    FOR y = 0 TO 7:
        FOR x = 0 TO 7:
            t = initial_setup[y][x]
            IF t != ' ':
                piece p
                p.type = t
                p.x_coordinate = x
                p.y_coordinate = y
                chess_board[y][x] = p
            END IF
        END FOR
    END FOR

END FUNCTION
```

3.2.2 print_board()

The print board function loops through the matrix setting every other square to “light” or “dark”. It also prints out the cordinates of each square and converts the letters used for board initialization to chess symbols in unicode.

```
FUNCTION print_board(chess):

    PRINT "---- Chess Game ----"
    PRINT "    A B C D E F G H"
    PRINT "  -----------------"

    FOR i = 0 TO 7:
        PRINT (8 - i) + " | "

        FOR j = 0 TO 7:

            IF (i + j) MOD 2 = 0:
                SET_BACKGROUND_COLOR("light")
            ELSE:
                SET_BACKGROUND_COLOR("dark")
            END IF

            piece = chess.board[i][j]

            IF piece != NULL:
                symbol = " "

                SWITCH (LOWERCASE(piece.type)):
                    CASE 'p':
                        symbol = "♟"
                    CASE 'r':
                        symbol = "♜"
                    CASE 'n':
                        symbol = "♞"
                    CASE 'b':
                        symbol = "♝"
                    CASE 'q':
                        symbol = "♛"
                    CASE 'k':
                        symbol = "♚"
                END SWITCH

                IF IS_UPPERCASE(piece.type):
                    SET_TEXT_COLOR("white")
                ELSE:
                    SET_TEXT_COLOR("black")
                END IF

                PRINT symbol + " "

            ELSE:
                PRINT "  "
            END IF

            RESET_COLORS()

        END FOR

        PRINT NEWLINE

    END FOR

    PRINT "--------------------"

END FUNCTION
```

3.2.3 is_white()

Checks if the type is white or black depending if the piece is represented by a lower case letter or an uppercase letter.

```
FUNCTION is_white(type):

    IF type = UPPERCASE:
        RETURN 1
    ELSE:
        RETURN 0
    END IF

END FUNCTION
```

3.2.4 pawn_moves()

Checks if the pawn is white to determine its direction that it can move. If its the pawns first move it can move to spaces at once otherwise it can only move one. It also checks if there are any pieces infront of the pawn diagionaly that the pawn can capture,

```
FUNCTION pawn_moves(chess_board, piece, dest_y, dest_x):

    IF is_white(piece.type):
        direction = -1
        startRow = 6
    ELSE
        direction = 1
        startRow = 1
    END IF

    dx = dest_x - piece.x_coordinate
    dy = dest_y - piece.y_coordinate


    IF dx = 0:
        IF dy = direction AND chess_board[dest_y][dest_x] = NULL:
            RETURN 1
        END IF
        IF piece.y_coordinate = startRow AND dy = 2 * direction AND
chess_board[dest_y][dest_x] = NULL:
            RETURN 1
        END IF
    END IF

    IF ABSOLUTE(dx) = 1 AND dy = direction AND chess_board[dest_y][dest_x] !=
NULL
    AND is_white(piece.type) != is_white(chess_board[dest_y][dest_x].type):
        RETURN 1
    END IF

    RETURN 0

END FUNCTION
```

3.2.5 rook_moves()

Checks all available spaces that the rook can move on one row and column. If there is another piece obstructing the rooks path it cant move further than that piece and if its an opponents piece it can take that piece.

```
FUNCTION rook_moves(chess_board, piece, dest_y, dest_x):

    dx = dest_x - piece.x_coordinate
    dy = dest_y - piece.y_coordinate

    IF dx != 0 AND dy != 0:
        RETURN 0
    END IF
    IF dx = 0 AND dy = 0:
        RETURN 0
    END IF

    IF dx != 0:
        IF dx > 0:
            step_dir_x = 1
        ELSE:
            step_dir_x = -1
        END IF
    ELSE
        step_dir_x = 0
    END IF

    IF dy != 0:
        IF dy > 0:
            step_dir_y = 1
        ELSE:
            step_dir_y = -1
        END IF
    ELSE
        step_dir_y = 0
    END IF

    current_step_x = piece.x_coordinate + step_dir_x
    current_step_y = piece.y_coordinate + step_dir_y

    WHILE current_step_x != dest_x OR current_step_y != dest_y
        IF chess_board[current_step_y][current_step_x] != NULL:
            RETURN 0
        END IF
        current_step_x = current_step_x + step_dir_x
        current_step_y = current_step_y + step_dir_y
    END WHILE

    IF chess_board[dest_y][dest_x] == NULL OR
    is_white(piece.type) != is_white(chess_board[dest_y][dest_x].type):
        RETURN 1
    END IF

    RETURN 0

END FUNCTION
```

3.2.6 knight_moves()

Determines the knights move set by checking for square that are 2+-2 x squares and +-1 y square away. This can also be flipped to allow the knight to move any direction.

```
FUNCTION knight_moves(chess_board, piece, dest_y, dest_x):

    dx = dest_x - piece.x_coordinate
    dy = dest_y - piece.y_coordinate

    IF (ABSOLUTE(dx) = 2 AND ABSOLUTE(dy) = 1) OR (ABSOLUTE(dx) = 1 AND
ABSOLUTE(dy) = 2):
        IF chess_board[dest_y][dest_x] = NULL OR
        is_white(piece.type) != is_white(chess_board[dest_y][dest_x].type):
            RETURN 1
        END IF
    END IF

    RETURN 0

END FUNCTION
```

3.2.7 bishop_moves()

Checks all diagional rows from the bishops position and allows for moves until another piece is encountred. If another piece is encountered it cant move futher than that piece. If its an enemy piece it is allowed to take that piece.

```
FUNCTION bishop_moves(chess_board, piece, dest_y, dest_x):

    dx = dest_x - piece.x_coordinate
    dy = dest_y - piece.y_coordinate

    IF ABSOLUTE(dx) != ABSOLUTE(dy):
        RETURN 0
    END IF
    IF dx = 0 AND dy = 0:
        RETURN 0
    END IF

    IF dx != 0:
        IF dx > 0:
            step_dir_x = 1
        ELSE:
            step_dir_x = -1
        END IF
    ELSE
        step_dir_x = 0
    END IF

    IF dy != 0:
        IF dy > 0:
            step_dir_y = 1
        ELSE:
            step_dir_y = -1
        END IF
    ELSE
        step_dir_y = 0
    END IF

    current_step_x = piece.x_coordinate + step_dir_x
    current_step_y = piece.y_coordinate + step_dir_y

    WHILE current_step_x != dest_x AND current_step_y != dest_y
        IF chess_board[current_step_y][current_step_x] != NULL:
            RETURN 0
        END IF
        current_step_x = current_step_x + step_dir_x
        current_step_y = current_step_y + step_dir_y
    END WHILE

    IF chess_board[dest_y][dest_x] == NULL OR
    is_white(piece.type) != is_white(chess_board[dest_y][dest_x].type):
        RETURN 1
    END IF

    RETURN 0

END FUNCTION
```

3.2.8 queen_moves()

Works like a bishop and a rook combined. Se aformentioned functions above.

```
FUNCTION queen_moves(chess_board, piece, dest_y, dest_x):

    dx = dest_x - piece.x_coordinate
    dy = dest_y - piece.y_coordinate

    IF dx = 0 AND dy = 0:
        RETURN 0
    END IF
    IF NOT ABSOLUTE(dx) != ABSOLUTE(dy) OR (dx != 0 OR dy != 0):
        RETURN 0
    END IF

    IF dx != 0:
        IF dx > 0:
            step_dir_x = 1
        ELSE:
            step_dir_x = -1
        END IF
    ELSE
        step_dir_x = 0
    END IF

    IF dy != 0:
        IF dy > 0:
            step_dir_y = 1
        ELSE:
            step_dir_y = -1
        END IF
    ELSE
        step_dir_y = 0
    END IF

    current_step_x = piece.x_coordinate + step_dir_x
    current_step_y = piece.y_coordinate + step_dir_y

    WHILE current_step_x != dest_x OR current_step_y != dest_y
        IF chess_board[current_step_y][current_step_x] != NULL:
            RETURN 0
        END IF
        current_step_x = current_step_x + step_dir_x
        current_step_y = current_step_y + step_dir_y
    END WHILE

    IF chess_board[dest_y][dest_x] == NULL OR
    is_white(piece.type) != is_white(chess_board[dest_y][dest_x].type):
        RETURN 1
    END IF

    RETURN 0

END FUNCTION
```

3.2.9 king_moves()

Checks one square away from itself in any direction.

```
FUNCTION king_moves(chess_board, piece, dest_y, dest_x):

    dx = dest_x - piece.x_coordinate
    dy = dest_y - piece.y_coordinate

    IF dx = 0 AND dy = 0:
        RETURN 0
    END IF
    IF ABSOLUTE(dx) > 1  OR ABSOLUTE(dy) > 1:
        RETURN 0
    END IF
    IF chess_board[dest_y][dest_x] = NULL OR
    is_white(piece.type) != is_white(chess_board[dest_y][dest_x].type):
        RETURN 1
    END IF

    RETURN 0

END FUNCTION
```

3.2.10 is_valid_move()

Checks if the the moves are legal. This prevents pieces moving outside of the board.

```
FUNCTION is_valid_move(chess_board, piece, dest_y, dest_x):

    IF piece = NULL:
        RETURN 0
    END IF
    IF dest_y < 0 OR dest_y > 7 OR dest_x < 0 OR dest_x > 7:
        RETURN 0
    END IF

    SWITCH (LOWERCASE(piece.type))
        CASE 'p':
            RETURN pawn_moves(chess_board, piece, dest_y, dest_x)
        CASE 'r':
            RETURN rook_moves(chess_board, piece, dest_y, dest_x)
        CASE 'n':
            RETURN knight_moves(chess_board, piece, dest_y, dest_x)
        CASE 'b':
            RETURN bishop_moves(chess_board, piece, dest_y, dest_x)
        CASE 'q':
            RETURN queen_moves(chess_board, piece, dest_y, dest_x)
        CASE 'k':
            RETURN king_moves(chess_board, piece, dest_y, dest_x)
        default:
    END SWITCH

    RETURN 0

END FUNCTION
```

3.2.11 make_move()

Makes the move of the piece and updates it position and checks if a king is captured.

```
FUNCTION make_move(chess_board, piece, dest_y, dest_x):

    IF chess_board[dest_y][dest_x] != NULL:
        IF chess_board[dest_y][dest_x].type = 'k'
            RETURN 1
        END IF
    END IF

    chess_board[piece.y_coordinate][piece.x_coordinate] = NULL
    piece.x_coordinate = dest_x
    piece.y_coordinate = dest_y
    chess_board[dest_y][dest_x] = piece

    RETURN 0

END FUNCTION
```

3.2.12 parse_input()

Parses the players input so that an input like [e2 e4] is allowed to be entered instead of just numbers.

```
FUNCTION parse_input(input, from_y, from_x, to_y, to_x):

    IF LENGTH(input) < 5 OR input[2] != ' ':
        PRINT "Invalid input format. Use format like 'e2 e4'."
        RETURN 0
    END IF

    from_x = ASCII(input[0]) - ASCII('a')
    to_x = ASCII(input[3]) - ASCII('a')

    from_y = 8 - (ASCII(input[1]) - ASCII('0'))
    to_y = 8 - (ASCII(input[4]) - ASCII('0'))

    IF from_x < 0 OR from_x > 7 OR from_y < 0 OR from_y > 7 OR
       to_x < 0 OR to_x > 7 OR to_y < 0 OR to_y > 7:
        PRINT "Coordinates out of range. Use a-h and 1-8."
        RETURN 0
    END IF

    RETURN 1

END FUNCTION
```