//

Chess Engine チエス

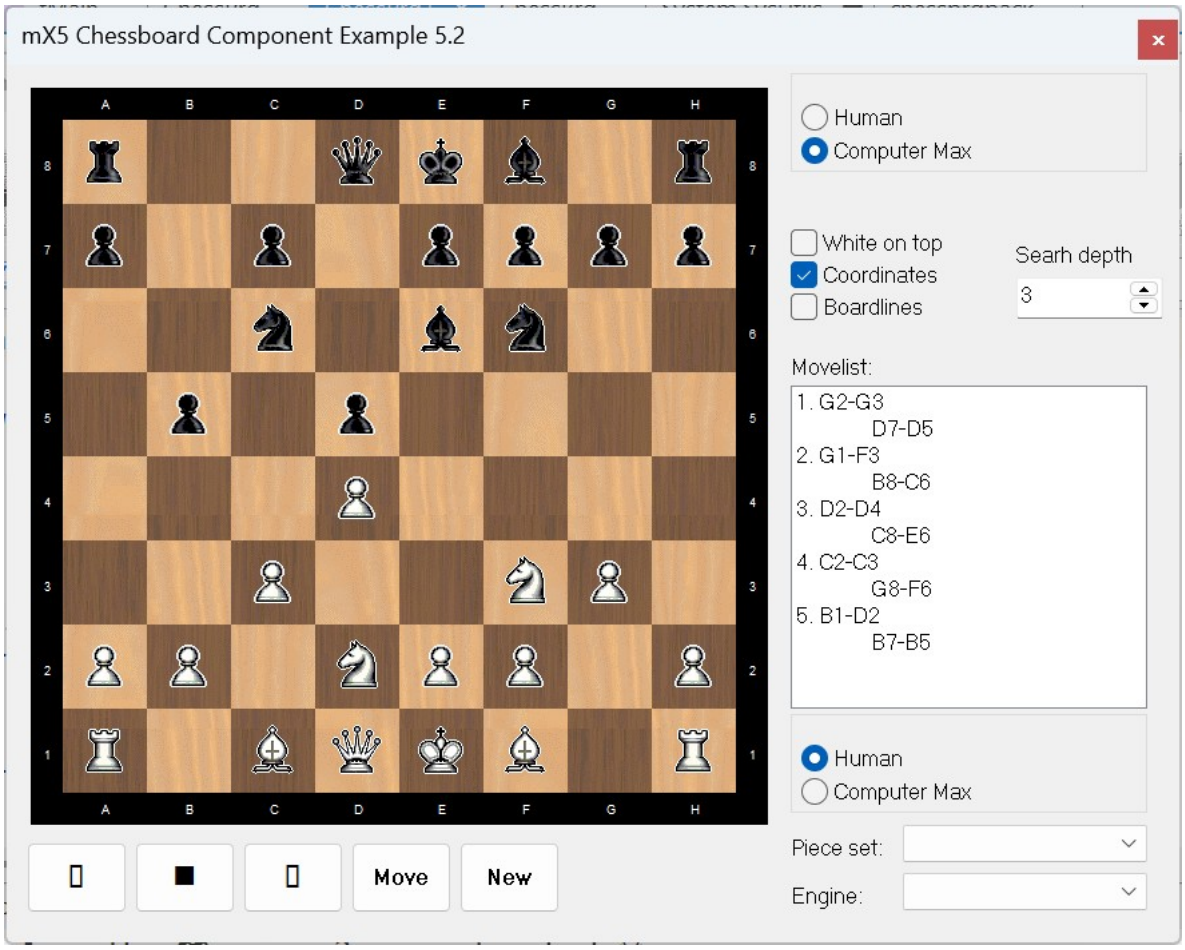
maXbox Starter 150 - Get a Chess Game.

"Non ridere, non lugere, neque detestari, sed intelligere.[1] - Spinoza.

Source: 1416_gemini_14_ai_py_uc_compact.txt
Executable: **maXbox5.exe menu/Options/Chess V52**

maxkleiner/Pascal-Chess: Delphi/Pascal chess game

This 64bit integration contains translated source from Tom's Simple Chess Program, but its still going strong. The difficult part to 64bit migration follows below, mainly to convert the PAnsiChar to a PWideChar. In Delphi, PWideChar is a pointer to a wide character (Unicode WideChar) string. It is commonly used when working with APIs or functions that require null-terminated wide strings.



1416_Gemini2025-06-24_164737.png

This is the basic description of a move. promote is what piece to promote the pawn to, if the move is a pawn promotion. bits is a bit-field that describes the move, with the following bits:

1 (ethik. 9,13) - Not to laugh, not to cry, not to hate, but to understand. -

```
//     1    capture
//     2    castle
//     4    en passant capture
//     8    pushing a pawn 2 squares
//    16    pawn move
//    32    promote
```

The thinking thread contains mainly code from Tom's Simple Chess Program:

```
TChessThread = class(TThread)
    private
        // pcsq stands for piece/square table, indexed by the piece's color,
        // type, and square. The value of pcsq[LIGHT,KNIGHT,e5] might be 310
        // one of the outer squares. //
        // instead of just 300 because a knight on e5 is better than one on

        pcsq:                Array [0..1,0..5,0..63] of Integer;
        flip:                Array [0..63] of Integer;
        pawn_pcsq:           Array [0..63] of Integer;
        kingside_pawn_pcsq:  Array [0..63] of Integer;
        queenside_pawn_pcsq: Array [0..63] of Integer;
        minor_pcsq:          Array [0..63] of Integer;
        king_pcsq:           Array [0..63] of Integer;
        endgame_king_pcsq:   Array[0..63] of Integer;
        color: Array [0..63] of Integer;  // LIGHT, DARK, or EMPTY
        piece: Array [0..63] of Integer;  // PAWN, KNIGHT, BISHOP, ROOK, QUEEN,
                                          // KING, or EMPTY
        side:  Integer;  // the side to move
        xside: Integer;  // the side not to move
        castle: Integer; // a bitfield with the castle permissions. if 1 is set,
                         // white can still castle kingside. 2 is white
                            queenside.
                         // 4 is black kingside. 8 is black queenside.
        ep: Integer ;    // the en passant square. if white moves e2e4, the en
                             passant
                         // square is set to e3, because that's where a pawn
                            would move
                         // in an en passant capture
        fifty: Integer;  // the number of moves since a capture or pawn move,
                            used
                         //   to handle the fifty-move-draw rule
        ply: Integer;    // the half-move that we're on
```

A PWideChar is similar to a PChar, but it supports Unicode characters, making it suitable for modern applications that require internat-ionalization.
Key Points:

- Null-Terminated: PWideChar strings are null-terminated, so you must ensure the string is properly terminated when working with it.
- Conversion: You can easily convert WideString or UnicodeString to PWideChar using a typecast.
- Memory Management: Be cautious when working with PWideChar to avoid memory leaks or invalid memory access. If you allocate memory manually, ensure it is freed appropriately.

So the masterpiece was to modify the **StrPas()** Routine references from AnsiString to UnicodeString:

```
function StrPas3(const Str: PAnsiChar): AnsiString;
begin
  Result:= Str;
end;

function StrPas2(const Str: PWideChar): UnicodeString;
begin
  Result:= Str;
end;
```

In the following we use several routines and also the game memory list was involved:

```
function TChessBrd.Get_Position: String;
begin
     Result:={ansistrings.}StrPas2(@_position[1]);
end;
```

Also the perform move was a point of interest:

```
MoveList[_CurrentMove,1-
Integer(_whiteToMove)].position:={ansistrings.}StrPas2(@_position[1]);
    MoveList[_CurrentMove,1-Integer(_whiteToMove)].Castling:=_castlingallowed;
    MoveList[_CurrentMove,1-Integer(_whiteToMove)].EnPassant:=_EnPassant;
    MoveList[_CurrentMove,1-Integer(_whiteToMove)].OldSquare:=Square(oldsq);
    MoveList[_CurrentMove,1-Integer(_whiteToMove)].NewSquare:=Square(newsq);
```

In Delphi 2007 and earlier, PChar is an alias for PAnsiChar. In Delphi 2009 and later, PChar is an alias for PWideChar. So by changing compiler you change the meaning of the code.
In modern Unicode Delphi it would be more natural to use string (alias of **UnicodeString**) instead of the COM WideString. You might also use of one the many library routines to perform UTF-8 conversion or in our case convert a PwideChar to a UnicodeString.

maxbraintrain_9-7-2025_15919maxbox52_small.jpeg

When UnicodeString was introduced in Delphi 2009, the old ANSI-based RTL functions were updated (but not renamed) to support Unicode, including WideCharToString(), making it now merely copy the 16bit data as-is rather

than convert it. And since a PWideChar can also be assigned directly to a UnicodeString (via the RTL's System._UStrFromPWChar() function), WideCharToString() is now completely redundant.

The list above is the move stack. Gen_dat is basically a list of move lists, all stored back to back. Gen_begin[x] is where the first move of the ply x move list is (in gen_dat). Gen_end is right after the last move.
Slide array, offsets, and offset are basically the vectors that pieces can move in. If slide for the piece is FALSE, it can only move one square in any one direction. offsets is the number of directions it can move in, and offset is an array of the actual directions.

Conclusion

ChessBoard Component: As always with Chessboards it can natively process multiple data types, including text, images, audio, and video, within a single unit if you want to enlarge the game.

- The board and engine from the year 2000 is still alive in maXbox5.
- It contains graphics from Andrew Gate adapted for mX5 by me.
- Tom Kerrigan's Simple Chess Program (TSCP) is a small, open-source chess engine that he made in 1997. It's a tutorial engine, i.e., it's designed to teach people how chess engines work.

On its home page [2] one can find not only the program itself, but also a couple of derivatives, like a version with null move pruning or with a bitboard move generator. TSCP 1.81 was the test engne in Mannen's and Wiering's computer chess learning experiments, where they trained several different chess evaluation functions (neural networks) by using TD(λ) learning on a set of database games, published in 2004/05 [3] [4].

Script:
https://sourceforge.net/projects/maxbox5/files/examples/655_arduino_chess64.txt/download

References:
Google Gemini API – Code Blog

TSCP - Chessprogramming wiki

maxkleiner/Pascal-Chess: Delphi/Pascal chess game

Doc and Tool: maXbox5 - Manage Files at SourceForge.net

Max Kleiner 10/07/2025

maXbox4exe
Digitally signed by maXbox4exe
Date: 2025.07.10 16:01:28 +02'00'