Collégial international Sainte-Anne

C++ Project

Chess Game Datasheet

William Tremblay

Presented to
Florence Morin

For the course
Case Studies in Applied Sciences and Mathematics

February 14, 2020

Introduction

My console chess game features a graphical interface and mouse-based input. It implements all standard rules of chess *except* forced draws by repetition and 50-move rule and the impossibility of castling if the king is in check or moving into check. Apart from that, the players can only make legal chess moves and the game is won upon checkmate (or drawn upon stalemate). The game was programmed using object-oriented programming in an extremely modular fashion such that it is very easy to modify to add new piece types and rules, with a maximum of 15 different piece types.

Project Structure

Note that since there is over 2000 lines of code, I will not explain every single function (there are probably more than a hundred) in detail. Comments already do this. All headers also contain their source to simplify file management.

Header files

- `IVec2.h`: Class implementing a 2D vector with integer components. Has two fields, `x` and `y`.
- `Byte88.h`: Class implementing an 8×8 byte matrix and related arithmetic/bitwise operators.
  - ⋆ `BoardState.h`: Subclasses `Byte88` to make an 8×8 `Piece` matrix.
- `PieceDef.h`: Abstract class for a piece type. Defines the piece's sprite, ID, and provides two important functions: `isValidMove` which calculates if a move is pseudolegal (follows the moveset of the piece) and `makeMove` which performs the required board manipulations for moving a piece.
  - ⋆ `UnitMovePiece.h`: `PieceDef` for any piece which moves linearly along a set of direction vectors. Used to implement the knight, bishop, rook and queen.
  - ⋆ `Pawn.h`: `PieceDef` implementing all the rules of a chess pawn (including en passant capture).
  - ⋆ `King.h`: `PieceDef` implementing all the rules of a chess king (including castling).
- `PixelFont.h`: Defines an array of 64-bit unsigned integers containing 8×8 pixel art for text characters called `PixelFont` and an helper function `getCharSprite` used to easily get a `Byte88` sprite from a text character. This function also adds an outline to the character.
- `GameWindow.h`: Class which enables drawing pixel art into the console using several Windows API functions. Changes the screen buffer and window size and sets the font so that a space character is a perfect square and can be used to draw pixels. Supports a colormap of 16 different RGB colors. Also includes the `drawText` function which allows to render text as pixel art. It is also the class that reads key and mouse events and dispatches them to event handlers (in this case, `ChessGame::onMouse`).
- `Layer.h`: Class implementing a 2D byte buffer of variable size used to store a layer of graphics data. Each color is stored in the upper part of the byte as a 4-bit number that represents the index of the color in the `GameWindow`'s colormap. Includes function to the color, overlay with another `Layer`, drawing a sprite held in a `Byte88`, etc.
- `SpriteDefs.h`: Defines the `Byte88` pixel art sprites for the chess pieces.
- `ChessGame.h`: The main class that is responsible on the game's general interactivity and behaviour. Includes functions to computer check, checkmate, legal moves, mouse and keyboard event handlers, etc.

Source files

- `ConsoleChess.cpp`: Program entry point containing the `main` function, instantiates the piece definitions, board state and chess game and starts the main loop of the latter.

Explanation of `ChessGame.h`

Since this class is responsible for most of the behaviour of the chess game, I will be explaining what each function in it does.

- `void init(std::vector<PieceDef*> pieces)`
  Initializes the fields of the `ChessGame`, setting up the `PieceDefs`. The `GameWindow` is setup with the right size, colormap and and layers.
- `bool isAttacked(IVec2 pos)`
  Calculates if the piece at a given position is attacked by the enemy pieces. This is done using a simple brute-force approach where all enemy piece's `isValidMove` is called on the target position.
- `bool makeMove(IVec2 start, IVec2 end)`
  Dispatches the call to the piece's `makeMove` and stores the previous board setup. Also flips the current team. Returns a boolean that indicates that the moved piece should be promoted.
- `void undoMove()`
  Undoes the previous move, resetting the board state to what is was before.
- `bool inCheck(bool team)`
  Verify if a team is in check by iterating through all its pieces to find critical pieces (pieces that can't be captured, which in normal chess is just the king) and check if they are attacked using the `isAttacked` function.
- `int computeChecks(bool team)`
  Calculates all the critical pieces that are in check and sets a boolean `Byte88` to `true` at their position. Also returns the number of pieces in check.
- `int calculateLegalMoves(bool team)`
  Calculates all the legal moves of a team and stores them in a array of 64 `Byte88`. Simply goes over all pieces, and for each of them all 64 board positions are fed into `isValidMove`. If a move is found to be pseudolegal, it is executed temporarily with `makeMove/undoMove` so that we can verify that critical pieces are safe with `inCheck`, ensuring the move is completely legal. Returns the number of legal moves.
- `void redraw()`
  Updates the graphical interface and draws the board, text, selected square, etc. using methods from the `GameWindow`.
- `void beginGame()`
  Initializes the board state to its initial state, sets hover/selected square and other graphical features, calculates the legal moves of the current team (white) and finally calls `redraw`.
- `void mainloop()`
  Begins the game and waits listens for key/mouse events in an infinite while loop. This is what "starts" the game.
- `void onKey(KEY_EVENT_RECORD evt)`
  Event handler for key events. Implements a shortcut to exit the game by pressing Q and one to restart the game by pressing R.
- `void finalizeMove()`
  Finalizes the execution of a move by recalculating attacked crits / legal moves and deselecting pieces. Also calls `redraw`.
- `void onMouse(MOUSE_EVENT_RECORD evt)`
  Mouse event handler. Handles all interactivity logic, including selecting and moving pieces, hovering over a square, promotions and restarting the game using the "Restart" button.

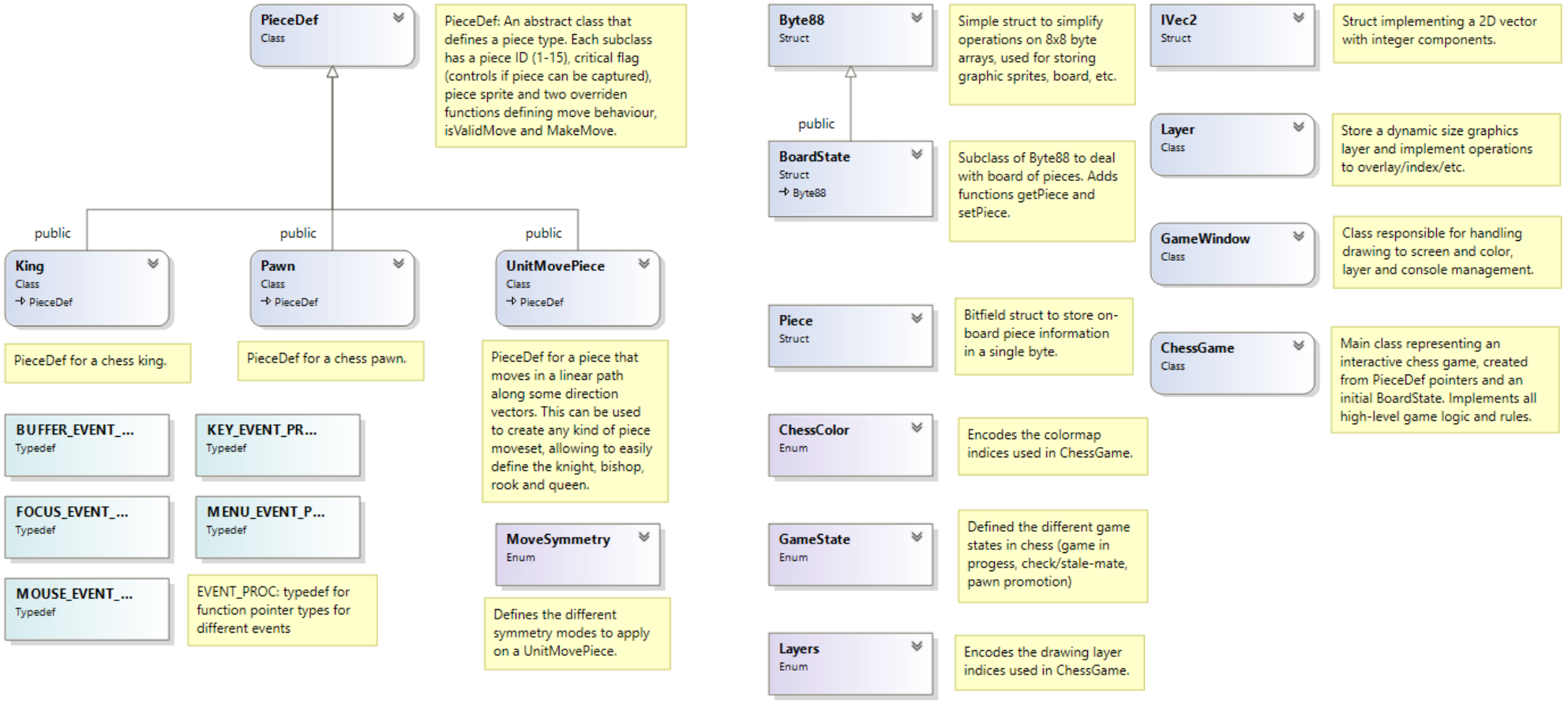

Figure 1: General structure of the chess game