Munir Gargour

Documentation of bitboard C project

I used C for this project as it was the language recommended in class for the assignment. It has bitwise operations that are not complicated to use compared to those in Java. I had to spend time learning how to use C properly with online video tutorials for this project to work.  
This project is split into 3 files: bitboard checkers.c (main driver), utility.h(header for utility class) and utility.c(utility class defining all functions). I first cover main, then utility, skipping utility.h because it’s just a header defining the format of the functions contained in utility.c.

MAIN():

VARIABLES:  
int redturn: whose turn it is, 1 for red, 0 for black.

Int from,to : which tile to move from and to, read from user input.

Int redwin: if red wins, null by default. 1 if red wins 0 if black wins, returns win message at end of main().

Int move\_success: if a move is successful, return 1 or 2 depending on if a capture occurred. If unsuccessful return 0.

I made the end game condition either bitboard being empty because I figured a simplified win condition would suffice. In the future, I would make a more fleshed-out win condition including checking whether each piece can possibly make a move.

I decided to make a reference bitboard for the user to view that prints each turn, because I wasn’t sure how else to make the user move apart from entering two values. In the future, I maybe would have implemented a row and column move system, E.G.” A 0 to B 1” rather than “0 to 7”.

Some of my code is contained in main, “bitboard checkers.c”, and in the future I would move it all to the utility class, but I felt a portion of it was easier to modify and more readable when in the main() class rather than the utility class which is full of other functions.

I made “move\_success” variable to check what type of move was made. An unsuccessful one repeats the turn and lets the player try again. Successful move ends turn. Success with capture prolongs turn 1 additional turn.

UTILITY CLASS:

Any reference to “position” , position is an int.

When clearing setting toggling, etc. bitboard is referenced using bitboard \*board (with a pointer), but when getting information about a position like in function “is\_king”, bitboard is referenced directly.  
BITBOARDS: red\_pieces,black\_pieces,kings,blank  
I used unsigned 64 bit bitboard rather than 32 even though it includes excess spaces, because it made move calculation easier for me to understand. This being said, in the future, for time complexity and space complexity reasons, I would change this to be 32 bit.

PrintBoad():

Taking in all 3 playable boards: king, black, and red boards, this function loops through each individual bit in each bitboard to first check if bit number 0 contains ONLY a red piece, then ONLY a black piece, then red and king, then black and king. Then it prints accordingly “r” “R” (red, red king), “b”, “B” (black, black king), or “ . “ or “ “, which are empty spaces, the dot being one that can be moved to and the space being a tile that cannot be moved to (white tile in checkers)

It checks with if(bitwise ‘and’ to combine conditions) a bit is found to equal 1 among any concerned bitboards. For example, it checks if the value at black bitboard [i] and the value at king bitboard[i] are equal to 1, and then prints a capital B if both conditions are met.

I had issues at first with the implementation of this function, as it would only print red, but it was because I had incorrectly used bitwise operators to check for conditions, and when I figured out where the issue was I was able to fix it.

PrintBlankBoard():

I made a simplified version of printboard() to print a blank board containing numbers in place of pieces to show the end user what the indices of any board position is, so that they can make moves without counting squares. Just uses one bitboard, it doesn’t matter which, but I made a blank bitboard for it. I could have used any of the others. Top left is 0 and bottom right is 63. Only playable tiles are shown. This did not take any bitwise operators and only prints the bit position at each state.

Is\_playable\_tile(position):

If the row is even and column of position is odd, it’s a playable tile. If the row is odd and column is even, it’s a playable tile. Otherwise it isn’t.

Is\_king(king bitboard, position): checks if king has a 1 in its bitboard at the position specified in the.

Is\_opponent\_piece(opponent’s bitboard, position): used in the try\_move function to check if an opponent piece is captured after a jump. Checks opponent bitboard at position and returns its value.

Make\_move(\*players bitboard, \*opponents bitboard, \*king bitboard, int from int to int is\_red\_turn):

This function serves only to move a game piece by clearing its from position and flipping its to position. Checks if the piece is a king and moves the king position as well to prevent a king with no color. At the end of any move, check\_promotion is called.

check\_promotion(uses pointers to bitboards):

if it’s red’s turn and they successfully move to a position in the last row (56+) or black’s turn and they move to a position in their last row(0-7) then that piece is promoted to king and the “kings” bitboard is updated with a 1 in the “to” position , which will follow that game piece until it is captured.

Bit arithmetic : division, subtraction, addition, and multiplication are all defined in the utility class, though they are not all used for calculating moves.  
  
set, clear, toggle, and get bit:  
These are basic bitwise operations some of which are used throughout the utility class. I put these here near the end of my work on this project, and they therefore do not see as much use as I would have liked. That said, clear\_bit and set\_bit are used in the try\_move function.

Try\_move(Bitboard \*player, Bitboard \*opponent, Bitboard \*kings, int from, int to, int is\_red\_turn);

This function is the biggest and most complex function in the project, and checks if a variety of conditions are met before allowing a move to be made.   
It takes in all bitboards (except blank), the from and to movement positions that the user inputs, and a check of whose turn it is.   
A challenge I faced with this involved my switch from 32 to 64 bit bitboards: Some of the logic used the number 4 for row length rather than 8, and I had forgotten to update it accordingly. This was a frustrating bug until I figured out what was causing it.   
Another bug I dealt with was pieces being able to be constructed from thin air by moving from a position with no pieces to an otherwise legal position. This was remedied by using get\_bit() to check the “from” position for a game piece of the color of the player.

Columns/rows of from and to positions are calculated.

A check if the piece is a king is made.

Difference in rows and columns between from and to are stored in variables.

Row difference is checked, and if row and col difference are equal to 1 or negative 1 (depending on the turn and whether piece is a king), it tells players they can only move forward and they get another try.

If the user wants to jump, after validity is checked of end position and direction, the function checks if the row and column difference are 2 or negative 2, and allows a jump to an empty tile, only if an opponent is in the middle. Is\_opponent\_piece is checked to make sure of this.   
Once the jump is validated, the tile index is saved and the bits of opponent and king at that index are cleared.   
If the input fails any of the conditions for a legal move, the move fails and player has to try again.