3D Chess - Team Cole Report

Github Handle: https://github.com/intro-graphics/team-project-cole
ALL COMMITS: https://github.com/intro-graphics/team-project-cole/commits/master

TEAM MEMBERS

Eli Katz[redacted]

Aditya Sriram[redacted]

Adam Cole,[redacted]

Jake Wallin,[redacted]

IDEA

Our idea was to use our newfound knowledge of computer graphics to implement a 3-dimensional playable Chess board. We were excited to design aesthetically pleasing pieces and a nice board to play on, as well as figure out how to map from a game's logical state to it's graphics state.

HOW TO PLAY

The game is played just like normal chess; on your turn, you choose a piece to move and our project will display the available moves to make. Select one of these red indicators, and your piece will slide to the new position. If an enemy piece occupies that square, your piece will "eat" the enemy, and the piece will be removed from the game. The taken piece will then be displayed next to the board on the correct side. Once a move is selected, the board will rotate and the opposing side will have their turn. <u>It is important that you click on the square that the piece is on and not the piece itself</u>, this is how we implemented mouse clicking.

Another important aspect of our project is that our chess board <u>does not implement check and checkmate logic</u>. In this way, it is the user's responsibility to understand how the situational rules in these circumstances apply to the game.

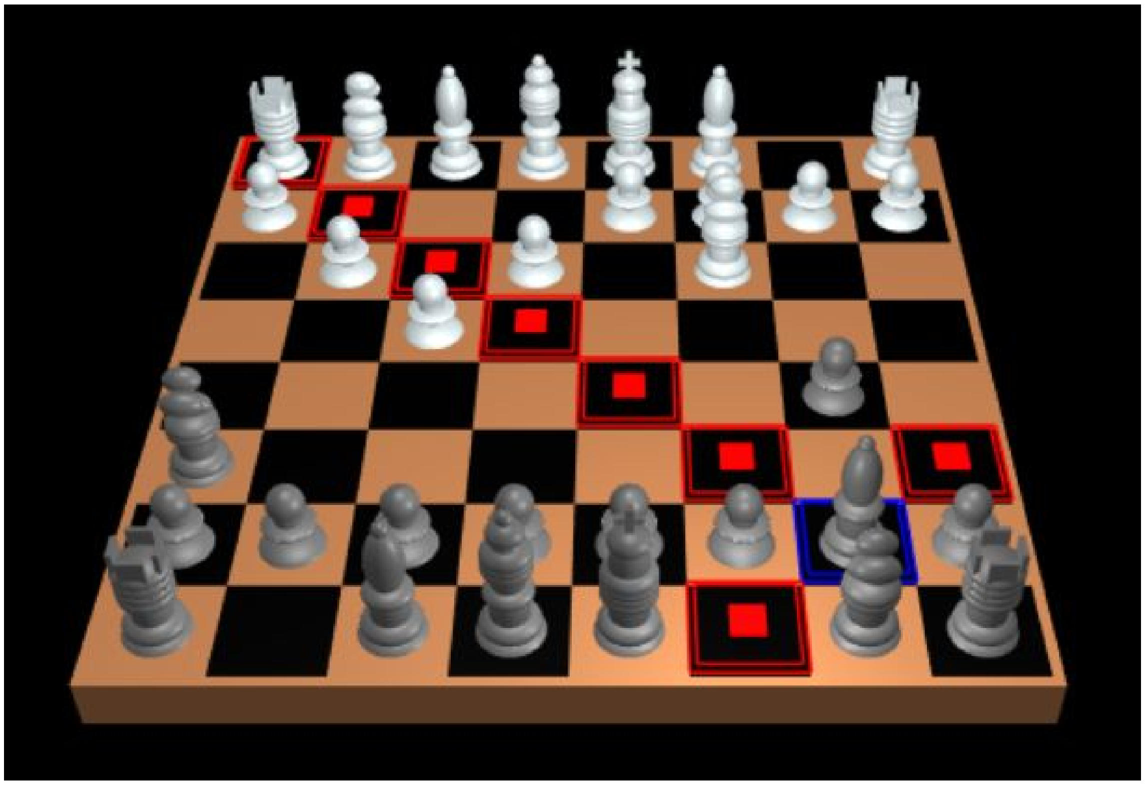

Figure 1: A selected bishop displays the available moves to be made.

IMPLEMENTATION

Features:

1. Graphics Design
    a. The game board was designed using tiny-graphics.js cubes, where we made an 8x8 grid and alternated the colors of the board tan and black. If the "change colors" button is selected, the tan squares become white. The edge of the board was constructed in the same way, except with cubes scaled down to be the edge.
    b. The backend of the chess board was a 2-dimensional array of strings containing the piece and color, or whether the square was empty. Using this class object, we were able to implement the limited functionality of the game.
    c. Designing pieces was done by overlaying scaled spheres and cubes on top of each other. This proved a useful plan of attack until designing the knights - which became their own unique creation.
2. Mouse Picking
    a. We ideally wanted to find a way to use our model, projection, and camera matrices to translate the x,y positions of a mouse click to our board to accurately get mouse picking working from any camera angle, alas we couldn't figure out how to do that and no one responded to our Piazza question :(so instead we

opted to go a more “hacky” route and hardcode an array of clickable areas corresponding to only two camera positions, white and black’s POV, with each clickable area corresponding to a tile on the board.

   b. In terms of the game’s state, it will initially be in a “no piece selected” state. After a tile is clicked with a piece of the current player on it, the game will enter a “piece selected” state and display the possible next moves on the board as red oscillating tiles. Now clicking on a red tile will trigger an animation of that piece to that tile; any other tile will leave go back to the “no piece selected” state.

3. Animations
   a. Animations are triggered when a piece in the “piece selected” state begins to move to a valid tile. Global variables are used for this; a state variable is flipped to signify we are in the animation state, this.animateTo and this.animateFrom are set based on results from the mouse picking (as described above), and over a period of 1 second the piece is drawn at a position determined as a function of time interpolating between this.animateFrom and this.animateTo. At the end of the animation, the game’s board state (a 2D array of chars representing pieces and empty tiles) is updated and the state is set back to “no piece selected”. Another global var is triggered to tell the camera to flip to the other player’s perspective.
4. Camera Positioning
   a. In order to implement the board rotation, we designed matrices to rotate the camera view around our stationary world. Since we only needed 2 views, the white perspective and the black perspective, this only required a few matrices. I started by finding the first perspective by moving around the world until I found a good spot, and noted the (x,y,z) position. This gave me the eye vector of both perspectives, and then I tested until determining the correct top and center of interest vectors. From this, I used the Mat4.lookat() library function to transition the camera from the current position to our desired one. For extra fun, I added a top view, which looks directly down at our chess board.
5. Captured pieces
   a. We maintain a list of captured pieces for both the white and black side. We initially have the lists as empty since the game starts with no pieces captures. Every time the user selects a move that will take an opponent's piece, we add that piece to the list of captured pieces. We then immediately display it to the side of the gameboard. (Captured black pieces will be next to the white side) Once we have captured 5 pieces, we then make another row of pieces which is 2 units closer to the middle of the board.

INTERESTING DETAILS

Aside from graphics, our group put effort into implementing the actual logic of the game. This includes the next_moves function, pawn promotions to queens, and animating the movement of pieces. The next_moves function is responsible for deciding which squares to highlight once a square has been clicked with a piece on it. The promotion functionality was implemented by checking the last rows for pawns, updating them to be queens.

TEAM MEMBERS CONTRIBUTIONS

Eli: Made the first iteration of the chess board, implemented mouse picking for both selecting a piece and choosing its move. I also implemented animations for the selected chess piece sliding.

Links:

https://github.com/intro-graphics/team-project-cole/commit/7f308647b19d8f43a59f49f3ff4173bbbc4750c9

https://github.com/intro-graphics/team-project-cole/commit/a6f52fcd1290e0e77d0714fbb5f0b5bd598845a1

Aditya: I worked on the mapping in order to get the mouse clicks to work. First, we had to find the positions at which the squares were located and hard code them into an array. We then had to convert the mouse click in 2D to a position on the board in 3D. One challenge we faced was that when the board flipped, none of our conversions worked anymore as they corresponded to the opposite piece. I had to work to fix issues like this. Along with that, I also worked on some smaller details including the logic for promotion when a pawn reaches the other side of the board(it turns into a queen).

Links:

https://github.com/intro-graphics/team-project-cole/commit/c0f85ef3652b5fcce26049956957b4ecefbe4873

https://github.com/intro-graphics/team-project-cole/commit/dcb89014266ed87cf2035201d12c68d9aecd41de

Adam: I implemented all game pieces and finished designing the second iteration of the game board that Eli had created (Link 1). I also implemented the camera views to rotate the board between the two perspectives (Link 2). I also implemented the original backend design of the game, creating the 2D gameboard array. Finally, I implemented the next_moves function, which displayed valid move indicators on the game board for the selected piece (Link 3).

Links:

https://github.com/intro-graphics/team-project-cole/commit/542301901121fb5eba99b8614019ed6937e525d3

https://github.com/intro-graphics/team-project-cole/commit/69c2853c04b70949c462448685c6c230df7932d3

https://github.com/intro-graphics/team-project-cole/commit/02c0a4d6717d3ac749280a5cbe028e4245d5d848

Jake:

I also did the Captured Pieces display, but was having issues with committing so I just sent my new code to Adam to push.

https://github.com/intro-graphics/team-project-cole/commit/9b8f33914235b33ed975b9b965a955884aecde9d

https://github.com/intro-graphics/team-project-cole/commit/0e4ea24ee159caf02c9a5e2743b47ece24b8e890