Group members:

Team Checkers

Tomer Laufer (Team Representative)

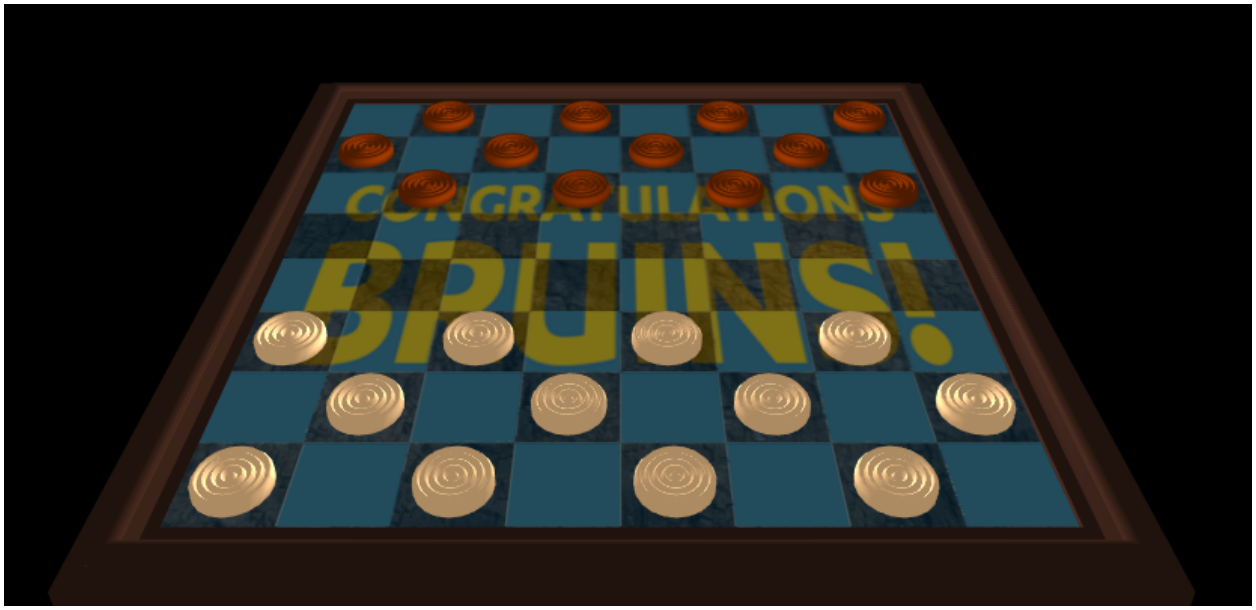
- email: tlaufer@ucla.edu
- Github: tomerlaufer123
- UID: 004937731

Minsuk Oh

- email: ohmin0922@gmail.com
- Github: min0922
- UID: 805360776

Kaitlyn Wang

- email: wang.kaitlyn.j@gmail.com
- Github: kaitlyn-wang
- UID: 005170129



Idea:

A mouse-based game of English draughts (American checkers) with an AI opponent to play against. Illegal moves are prevented automatically by the game.

How to play:

The player can select pieces to move using their mouse. Player pieces are white. Once a piece is selected, it will be highlighted for clarity. Clicking on a valid checker square will move the selected piece to that square. If an invalid square is selected, nothing will happen. After the player makes a valid move, the AI opponent will take their turn. The game ends when either player runs out of turns (when one player loses all their checkers or is trapped and cannot make

a move). The camera angle can be shifted using the control panel or by selecting one of the preset angles (player view, opponent view, or overhead). Click “Detach Camera” to move the camera freely.

The standard rules of English draughts (American checkers) are applied:

1. Each move can consist of:
    a. Moving a single checker piece diagonally by one square. Uncrowned pieces may only move forward (towards the opponent’s side of the board) while crowned pieces may move diagonally forward or backwards.
    b. If a piece is diagonally adjacent to an opponent piece, that piece may be “jumped”. The player piece moves to the square directly beyond the adjacent opponent piece (this square must be free) “jumping over” the opponent piece. Multiple jumps (even in different diagonal directions i.e. diagonally left or diagonally right) using the same piece can be linked in succession during one move. The opponent pieces that are jumped over are captured and removed from the game.
2. If a checker piece reaches the end of the opponent’s side of the board, it becomes “crowned”. Crowned pieces can move diagonally forwards and backwards. Crowning a piece ends the current move. In the game, crowned pieces are represented by two stacked checkers to differentiate them.
3. If a jump can be made, it is mandatory

Losing because piece is stuck

Advanced features:

Mouse picking

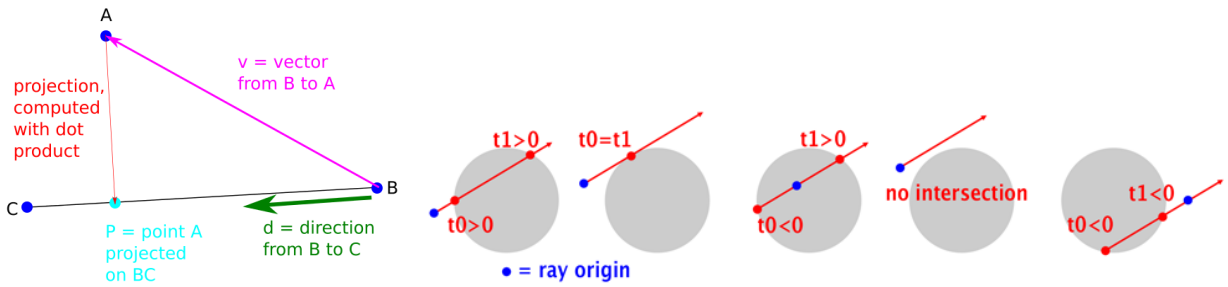
AI opponent

Implementation:

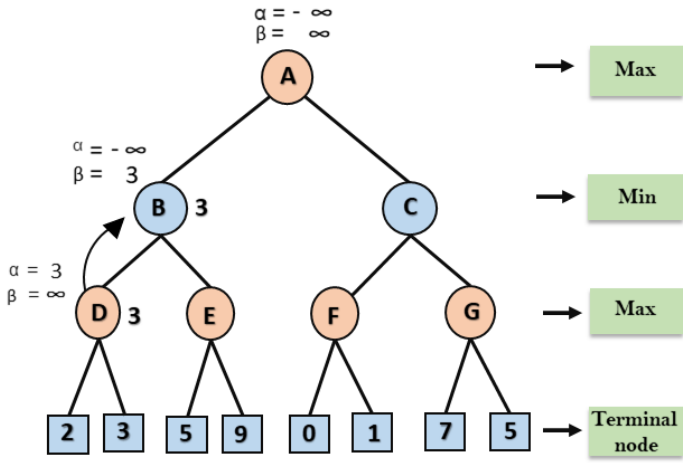
The mouse picking in our project was implemented using ray tracing. A new scene component “Pick_Checker” was added to track mouse inputs. The x and y coordinates of the mouse are retrieved upon detection of a click (mouse down). These coordinates are then normalized and converted from view space to world space as a vec3 ray. This is accomplished by first creating a homogeneous point using the normalized x and y coordinates (undo window-to-viewport mapping) and -1 as the z value (heading out of the camera), then multiplying by the inverse of the projection transformation matrix to convert the point into eye space. A ray vector is then created using the x and y of the eye space point (z of -1 again) and converted to world space by multiplying by the inverse camera transformation matrix. The final ray vector is formed by taking the xyz values of the point and normalizing them. This ray originates from the camera and heads in the direction indicated by the mouse location. Objects that are intersected by this ray will fall underneath the mouse from the perspective of the viewport.

An array of the player checkers’ locations (bottom center points of the checkers) is passed to Pick_Checker. Intersection between the line drawn by the ray and checker objects can be calculated by placing a spherical “hitbox” around each location (y values of the centers are slightly increased to better match the center point of the objects). If the distance between a center point and the closest point on the ray is less than the radius of the sphere, it indicates that the checker has been clicked on. The equation of a sphere can be written as $x^2 + y^2 + z^2 = radius^2$, or $center^2 - radius^2 = 0$. Let the ray from the camera coordinates to the center of the checker be b. b will represent the location of the center, so $b^2 - radius^2 = 0$. Let c = the vector perpendicular to the ray going to the center point (ray - b). If b is replaced with c in the equation, a quadratic is formed: $(ray - b)^2 - radius^2 = 0$. If the discriminant of this quadratic is not zero, then there is an intersection between the ray and the sphere. The solution of the quadratic also shows if the intersection is in the positive direction of the ray (in front of the camera, not behind it), and the checker with the smallest solution will be the closest.

Finding the location on the board to move a checker piece to is done similarly. A ray is calculated as before. A t value (for a ray equation ray_origin + t * ray_vector) is found such that the y value of the point on the ray is equal to the set y value of all the checker centers (checkers lie on the xz plane). The new x and z values of the checker is found using this t.



The AI was implemented using a *depth-limited search algorithm*. The AI considers all his moves, and wants to choose the “best” one (“best” is explained later). However, to choose the best move, the AI will have to think ahead *d* moves, in order to outsmart the player (*d* is a parameter set to 5 to increase performance and this suffices to beat most novice players). But to think some moves ahead, the AI also has to predict how the player will play. For that reason, this specific algorithm is called *Minimax*: the AI tries to *maximize* his current position in the game, knowing that the player is trying to *minimize* the AIs position. The notion of “best” is determined by evaluating a *heuristic*, a function that looks at the board and hints to the AI, how well he is doing. The heuristic that this AI uses is based on how his pieces are clustered (protecting each other), how many queens he has, etc. But the time complexity of expanding such a tree, can be a real bottleneck in our graphics system (we would need to evaluate (13 ^ 5) ~370,000 possible positions). We can dramatically reduce this number using a technique called *alpha-beta* pruning (pruning is where we skip parts of the search tree that we do not need to evaluate). The basic principle of *alpha-beta* pruning is that the AI will never choose a move that is worse than his current best option. At the same time, the player will never choose a move that helps the AI more than his current best option. With this in mind, we can prune many of the positions that violate this principle and reduce our computational costs.



Interesting points:

- The titles beneath the pieces are actually a single texture on a large square (that way we easily add text that runs across the board). The single texture was created in Photoshop
- The checker pieces themselves are formed via solids of revolution. The cross-sectional top of the piece is a sin wave.
- The different camera angles that the player can view the board from use transition animations. The selected angle is mapped in such a way that the transition from one camera angle to another is not instantaneous, but rather, smooth.

- When a piece is mouse-picked, it is highlighted. This effect is implemented by dramatically increasing the ambient light on the piece.
- The queen's crowns look slightly different so they are more visible
- Please visit the following link for screenshots: https://github.com/intro-graphics/team-project-team-checkers/tree/master/screenshots

Contributions:

Tomer's Contributions

- Created Checker piece
    - via solid of revolution https://github.com/intro-graphics/team-project-team-checkers/commit/6d1be77ca5a1f7a5ab57d98029ad7bbc2ec7ceeb
    - Color + Stacked 2 pieces to make a queen: https://github.com/intro-graphics/team-project-team-checkers/commit/fbaeec69c928d68da9c3257e0861269d024cbd07
- Implemented Game logic
    - Rules https://github.com/intro-graphics/team-project-team-checkers/commit/648eb08be9bced1c8b911ea2f4f3c0642612e627
    - Legal move checking
    - gameover checking https://github.com/intro-graphics/team-project-team-checkers/commit/00ffd8818655e8feece63eefaf4f56e6b05ffc92
- Implemented AI
    - AI + Depth Limited Minimax https://github.com/intro-graphics/team-project-team-checkers/commit/ee15d9fdb27dd82299c597be5f3f5ec1d215b2cf
    - alpha-beta pruning https://github.com/intro-graphics/team-project-team-checkers/commit/ea6fa5c61192981c53fc047765b5457aafd3eaeb
    - Heuristic function https://github.com/intro-graphics/team-project-team-checkers/commit/8ba3c54788aa4a6de59547290e0d974cc589ab70
- Added Camera Angles + transitions between them https://github.com/intro-graphics/team-project-team-checkers/commit/420eee671ea2859fc303aecfa613803680c9b9dc
- Helped create texture to lay atop tiles https://github.com/intro-graphics/team-project-team-checkers/commit/1d548a3e49f2ffeced64f8ff557fcae2d4c1c1ec
- README https://github.com/intro-graphics/team-project-team-checkers/commit/7ff73cd2ec0946276a7c0a37ace4429d467f98b0

- Added some screenshots https://github.com/intro-graphics/team-project-team-checkers/commit/67ab510765faf0eed8042da1247d9a86604ca654
- Helped with project report

Kaitlyn's Contributions

- Created Frame object
    - Modeled in Tinkercad, edited obj file with Blender
    - https://github.com/intro-graphics/team-project-team-checkers/commit/271f564483322239864c21c9fad9f9df9d3e6d53
- Created table object
    - Modeled in Blender
    - https://github.com/intro-graphics/team-project-team-checkers/commit/621e56c95d6b4039a3b8a5c4a0802af19b8efd44
- Implemented Mouse Picking
    - Checker selection
    - Move selection
    - https://github.com/intro-graphics/team-project-team-checkers/commit/2e32d62cce4437cd6e59c310875192d2afdba6c5
- Integrated feedback from game logic into display
    - Board refresh/checker placement
    - https://github.com/intro-graphics/team-project-team-checkers/commit/53e1e610895f03c8f3bbc0afd6d5ad7b73937c95
- Selected checker highlighting
    - https://github.com/intro-graphics/team-project-team-checkers/commit/87f59d8cb1eaf7f55d2957ade312c26e867d51eb
- Added on screen popup message (victory/defeat) at end of game
    - https://github.com/intro-graphics/team-project-team-checkers/commit/b045e870dcf1ec9319b938a85443b22af48e23fb
- Helped edit image for the board tile texture
    - https://github.com/intro-graphics/team-project-team-checkers/commit/1dc8db3ffe58a4ff2f370e830df8027aa31de6e3
- Drew textures for background and created room
    - https://github.com/intro-graphics/team-project-team-checkers/commit/621e56c95d6b4039a3b8a5c4a0802af19b8efd44
- Added some screenshots
    - https://github.com/intro-graphics/team-project-team-checkers/commit/d8318c339cf4d2461b2d0cb48981e03288047b63
- Helped with project report