Demo

- All proved in files demonstrate the functionality of the expected text and graphical board rendering.
- Note that all memory is managed using smart pointers and STL containers, and the only time raw pointers are used are if they are not meant to express ownership.

SETUP AND CHECKMATE INPUT (run the setup.in file):

- Add a white king, try to exit
    - (you can't exit setup with less than one king, should say "black player does not have a king on the board. Must have one king per player.")
- Add 2 black kings, try to exit
    - (you can't exit setup with more than one king, should say "black player has too many kings on the board. Maximum of one king is allowed per player.").
- Then remove one of the black kings
    - (shows functionality of - command).
- Add a white pawn to the first row, try to exit, then remove the pawn.
    - (you can't have a pawn on the first row, should say "white player has a pawn on the first/last row of the board. Invalid setup.").
- Add a white pawn to the last row, try to exit, then remove the pawn.
    - (you can't have a pawn on the last row, should say "white player has a pawn on the first/last row of the board. Invalid setup.").
- Add a white knight that puts the black king in check, try to exit, then remove the knight.
    - (neither king can start in check, should say "black player's king is in check. Invalid setup.").
- Add a white rook on the same row as the white king
    - (this does not put the white king in check since it is the same colour piece, so exiting setup is possible).
- Add another white rook
    - (shows you can add more than once of the same piece type).
- Make the current turn be the black player
    - (shows functionality of = command, should say "It is black player's turn" as opposed to the default of white).
- Use an invalid input command for +
    - (throws an error but does not crash)
    - "Invalid Input." If input does not match any of the accepted patterns ("+ *Colourchar Piecechar Colchar RowInt*")
    - "Specified square not in bounds!" If piece parameter matches accepted patterns, but given coordinate is not within board limits
    - "Invalid Colour character!" If the piece parameter is of correct length, but does not give a valid colour.
    - "Invalid Piece character!" If the piece parameter is of correct length, but does not give a valid piece.

- Use an invalid input command for -
    - Will say “Specified space not in bounds”
- Exit setup.
- Start a game with the following invalid inputs (should throw an error, but not crash)
    - game human
    - game human human human
    - game human computer[5]
- Start a proper game between 2 human players
    - (the board will have the curated setup from earlier)
- Resign
    - (Since the black player started, the black player resigns, letting the white player win. Game ends, displaying a win message)
- Start another game between 2 human players
    - (Board should now be using the default setup, and white player starts)
- Put white in checkmate, black should win
- At the end of the program, the score should be 1 to 1.

PAWN INPUT (run the pawn.in file)

- Main features this shows:
    - Pawn can move 1 space forward
    - Pawn can move 2 spaces forward on its first move
    - Pawn can capture other pieces by moving diagonal forward
    - Pawn can only execute valid moves
    - Valid en passant
    - Invalid en passant when not made immediately after the double move (not accepted)
    - Pawn promotion with specified input
    - Pawn promotion with no specified input, asks user for promotion piece

- move a2 a3 (shows pawn moving one space forward)
- move b7 b5 (shows pawn moving 2 spaces forward)
- move c2 c4
- move b5 c4 (black pawn captures white pawn)
- move a3 a2 (pawns cannot move backwards, invalid move, still their turn)
- move d2 d4
- move c4 d3 (valid en passant)
- move h2 h4
- move c7 c5
- move h4 h5
- move c5 c4
- move h5 h6
- move a7 a6
- move b2 b4
- move d7 d6
- move h6 g7 (white pawn captures black pawn)

- move c4 b3 (invalid en passant, was not immediately after the double move)
- move g8 h6
- move g7 g8 q (valid pawn promotion)
- move c4 c2 (invalid move, pawns can only move 2 spaces forward on their first move)
- move c4 b3 (invalid move, pawns can only move diagonally when capturing a piece)
- resign
- setup
- + K a1
- + k a3
- + P a7
- done
- game human human
- move a7 a8 (pawn promotion without specifying value, will ask for user input)
- r (changes it to a rook)
- resign

ROOK INPUT (run the rook.in file)

- Main features this shows:
    - Rook can move in any of the four vertical/horizontal directions, any distance, but cannot move past any piece that blocks its path
    - Rook can only execute valid moves
    - Rook can capture other pieces

- move a1 a3 (rooks cannot move past a piece that blocks its path, invalid move, still their turn)
- move a2 a4
- move h7 h5
- move a1 a3 (rook can move forward any amount)
- move h8 h6 (rook can move forward any amount)
- move a3 h3 (rook can move right any amount)
- move h6 d6 (rook can move right any amount)
- move h3 h5 (white rook captures black piece)
- move d6 e6 (rook can move left any amount)
- move h5 h4 (rook can move backward any amount)
- move e6 e1 (rooks cannot move past a piece that blocks its path, invalid move, still their turn)
- move e6 f5 (invalid path for rook, still their turn)
- resign

KNIGHT INPUT (run the knight.in file)

- Main features this shows:
    - Knight can move to square (x ±2, y ±1) or (x ±1, y ±2) if it sits on square (x, y).
    - Knight can “jump over” any piece that blocks its path.
    - Knight can only execute valid moves
    - Knight can capture other pieces
    - Putting a player in check

- move g1 f3 (knight can jump over pieces, moves (x-1, y+2))
- move g8 f6 (knight can jump over pieces, moves (x-1, y-2))
- move f3 d4 (x-2, y+1)
- move f6 d5 (x-2, y-1)
- move d4 e6 (x+1, y+2)
- move d5 e3 (x+1, y-2)
- move e6 g7 (x+2, y+1) (white knight captures black piece, putting black player in check)
- move f8 g7 (to get black out of check)
- move b1 b3 (invalid move, still their turn)
- move b1 c3
- move e3 g2 (x+2, y-1) (black knight captures white piece, putting white player in check)
- resign

BISHOP INPUT (run the bishop.in file)

- Main features this shows:
    - Bishop can move in any of the four diagonal directions, any distance, but cannot move past any piece that blocks its path.
    - Bishop can only execute valid moves
    - Bishop can capture other pieces
    - Putting a player in check

- move c1 e3 (bishops cannot move past a piece that blocks its path, invalid move, still their turn)
- move d2 d4
- move e7 e5
- move c1 d2 (bishop can move diagonal forward right any distance)
- move f8 c5 (bishop can move diagonal forward right any distance)
- move d2 b4 (bishop can move diagonal forward left any distance)
- move c5 d4 (bishop can move diagonal forward left any distance, black bishop captures white piece)
- move b4 c3 (bishop can move diagonal backward right any distance)
- move d4 c3 (bishop can move diagonal forward right any distance, black bishop captures white piece, white is in check)
- move d1 d2 (to get white out of check)
- move c3 e5 (bishops cannot move past a piece that blocks its path, invalid move, still their turn)
- move c3 d4 (bishop can move diagonal backward left any distance)
- resign

QUEEN INPUT (run the queen.in file)

- Main features this shows:
    - Queen moves in any of the eight possible directions, any distance, but cannot move past any piece that blocks its path.
    - Queen can only execute valid moves
    - Queen can capture other pieces

- move c7 c5
- move d1 d3 (queen can move forward any amount)
- move d8 c7 (queen can move diagonal forward right any distance)
- move d3 d2 (queen can move backward any distance)
- move c7 h2 (queen can move diagonal forward left any distance, black queen captures white piece)
- move d2 d3 (queen can move forward any amount)
- move h2 f4 (queen can move diagonal backward right any distance)
- move d3 a3 (queen can move left any distance)
- move f4 c4 (queen cannot move past a piece that blocks its path, invalid move, still their turn)
- move f4 h6 (queen can move diagonal backward left any distance)
- move a3 h3 (queen can move right any distance)
- resign

KING INPUT (run the king.in file)

- Main features this shows:
    - King moves one square in any direction.
    - King can only execute valid moves
    - King can capture other pieces
    - Cannot make a move that puts your player in check
    - Valid castle
    - Invalid castle (must be the first move for the king and rook)

- move e2 e4
- move e7 e5
- move g1 h3
- move e8 e6 (invalid move, king can only move one square over)
- move e8 e7 (king can move forward one square)
- move f1 e2
- move e7 f6 (king can move diagonal forward left one square)
- move e1 g1 (valid castle, king moves 2 spaces towards rook and rook takes the space skipped over)
- move f6 g6 (king can move left one square)
- move g2 g4
- move g6 f5 (INVALID MOVE, WILL PUT YOU IN CHECK, still their turn)
- move g6 f6 (king can move right one square)
- move g1 g2 (king can move forward one square)
- move f6 e7 (king can move diagonal back right one square)
- move g2 g1 (king can move backward one square)
- move e7 d6 (king can move diagonal forward right one square)
- move g4 g5
- move d6 e7 (king can move diagonal back left one square)
- move b1 c3

- move e7 e6
- move g5 g6
- move e6 f6
- move a2 a3
- move f6 g6 (black king takes white piece)
- resign
- game human human
- move g1 h3
- move b8 c6
- move e2 e3
- move b7 b5
- move f1 e2
- move c8 a6
- move e1 f1
- move d7 d5
- move f1 e1
- move d8 d7
- move e1 g1 (invalid castle, not the first move for the king, still their turn)
- move a2 a3
- move a8 b8
- move a3 a4
- move b8 a8
- move a4 a5
- move e8 c8 (invalid castle, not the first move for the rook, still their turn)
- resign

STALEMATE INPUT (run the stalemate.in file)

- Puts the black player in stalemate, game ends

COMPUTER PLAYER DEMOS:

In order to establish a way for the computer players to understand what pieces are most valuable, we used the standard chess scoring system that rates pawns: 1 point, Knights and Bishops: 3 points, Rooks : 5 points, Queens : 9 points, and we established King to be 10 points so that the computer knows it is the most valuable piece.

The following tests will outline and demonstrate situations where the computer player's decision making is put to the test.

Computer Player 1:

Test 1: computerplayerdiff1.in

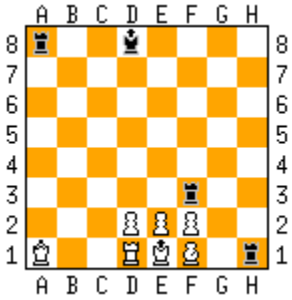
This test initializes a standard 2 player chess game between 2 Level 1 computer players. Running with this file as input multiple times will demonstrate how level 1 randomly picks valid moves all the way through.

Computer Player 2:

Test1: takeBestPiece.in

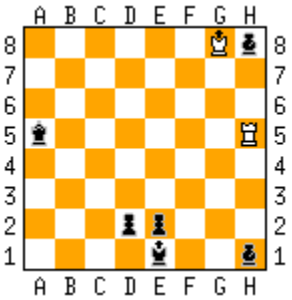
Running the program with this input file will generate a board as such, with black going first. Since computer player level 2 prioritizes takes and checks, it will at any point prioritize taking the highest value piece (in this case e4 taking the queen at e1). Once that move is done, white will take the highest value piece it can, which is b1 rook taking e1 rook.

Test 2: takeBestPiece2.in



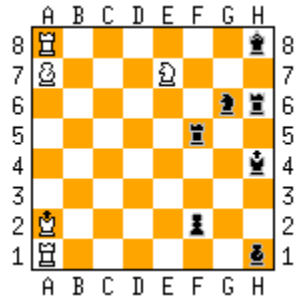
This input file sets the board as such, with black going first. Even though a8 can take queen at a1, black instead opts to take the lower valued bishop at f1 by taking with h1, because this is the only move that results in a take **AND** a check. From there white must take the rook at f1 with its king, at which point black has another opportunity to take and check by moving f3 to take f2. White must respond by taking f2 with its king, at which point black can take a1 with a8 since that is the best move available to it. Finally, once the black rook is in a1, white takes it with its rook at d1.

Test 3: takeandcheck.in



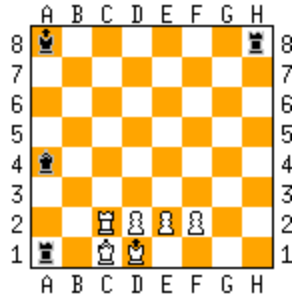
This input file sets the board as such, with white going first. Even with the enticement of the queen at a5, computer player level 2 prioritizes moves that allow for both a take and a check, thus it takes the bishop at h1, placing black in check.

Test 4: chooseBestCheck.in



This input file sets the board up as above, with white going first. As can be seen, there are 4 ways to take and check (a7->f2, a1->h1, e7->f5, e7->g6) and one way to just take (a8->h8). Since the rook at f5 is the highest value piece we can take that also causes a check, e7->f5 is the chosen move for white

Test 5: prioritizecheckmate.in



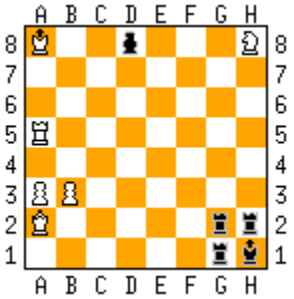
This input file sets the board as above, with black going first. Here we demonstrate how computer player level 2 prioritizes checkmates above all else, as even when presented with 2 situations where it can take and check, the checkmate caused by h8->h1 is played instead.

Computer Player 3:

Test 1: avoidcapture.in

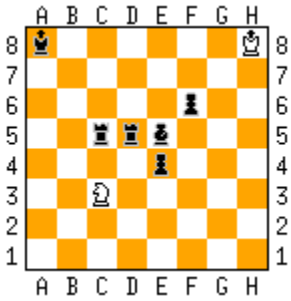
This input file sets the board as above, with black going first. Since computer player level 3's first priority is avoiding capture, it opts to move the black pawn forward out of range from the rook rather than capturing at h4 with its knight. From there, white wishes to avoid the rook at h4 getting captured by the enemy knight, and moves it to any location where it isn't threatened.

Test 2: saveBestPiece.in



The board is set up as such with white going. Note that the white queen is not savable. Whatever it does, it will be captured next turn. With this in mind, it instead saves the rook at a5 by moving it out of range from the bishop, while not moving it into the range of the rooks in the bottom right corner.

Test 3: **bestescape.in**



This input file sets the board up as above with white going. White wants to avoid its knight getting captured by the black rook or bishop, but instead of electing to move to squares a2, b1 or e2, it moves to e4, where it is equally safe, and on top of that it can capture a piece.

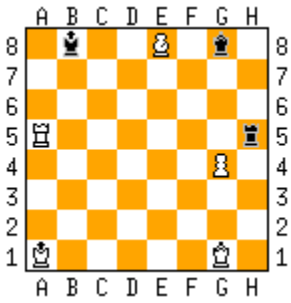
Test 4: **bestescape2.in**

This input file sets up as above with white going, and demonstrates how computer player 3 will never try to save one piece if it puts a better piece in immediate danger. For instance, even though the knight at c3 can be saved by moving to a2, b1, e2 or e4, doing so would put the more valuable rooks at a1 and c1in range of blacks bishop at e5 and rook at c5. Since it cannot save the rook at g7 protecting the king, it chooses to save the pawn at f7, promotes it to a queen, and places black in check.

Test 5: **bestescape3.in**

This board is set up very similarly to bestescape2.in with white going first, however now black has a rook at e2 and a knight at f8. Similarly to bestescape2.in, the rook at g7 cannot be moved, and the knight at c3 cannot be saved without endangering more valuable pieces. Even the added incentive of being able to take the more valuable rook at e2, the knight is not moved, and white computer player 3 instead saves the pawn at e7, this time taking the knight at f8 and once again promoting to queen, causing a check on black.

Test 6: dontexpose.in

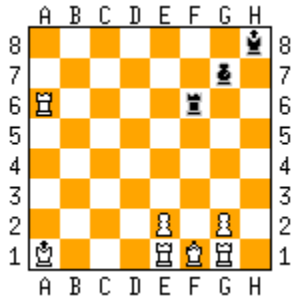


Here we demonstrate again how computer player 3 will not make moves that endanger higher value pieces in danger. White goes first, and since the rook at a5 is under threat from the rook at h5, and we have 3 pieces who can safely capture it (since black is not defending that square), it chooses to take it. Usually computer player level 3 uses the lowest value piece it can to make such captures, however since this is the pawn at g4, and moving it would expose the queen at g1 to the queen at g8, it opts for the next best thing: the bishop at e8.

Test 7: offenceisbestdefence.in

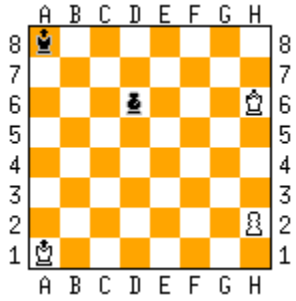
This test sets the board up as such, with white going. Note that the white queen at h1 is not savable, since any move it makes will result in a capture on blacks turn. Instead, white tries to save the pawn at d4. Although it could move it forward to d3. Taking the rook at b4 with its knight not only saves the pawn, but even if the knight is immediately captured, white trades a knight for a pawn, resulting in a +2 piece value difference on that turn in favour of white. Although this move does put a piece in danger, this is a situation where it is smart to do so.

Test 8: **sacrificepiece.in**



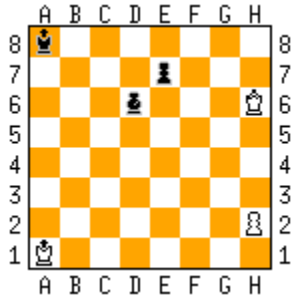
In this situation, white goes first and the queen is unable to escape, since even by taking the rook at f6 it places itself within range of the black bishop. In this case, white chooses to save its higher value piece by taking f6 with a6, which does endanger its white bishop but saves its more valuable queen as a result. In this situation it is effectively minimizing its loss by giving up the rook to save the queen.

Test 9: **safetodefend.in**



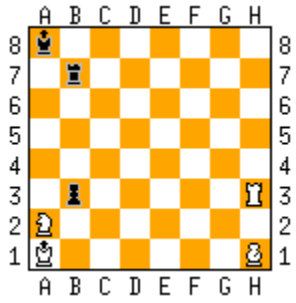
White goes first. Here the only white piece in danger is the white pawn, and since the white king is able to safely take the bishop without putting itself at risk, white saves its pawn by taking it with the queen.

Test 10: **unsafetodefend.in**



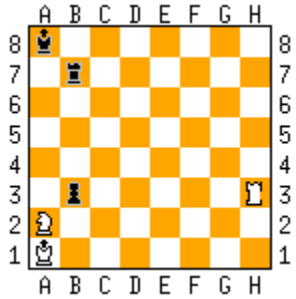
This setup is identical to test 9: safetodefend.in, except now the bishop is defended by a black pawn. Since the queen can no longer safely take, computer player level 3 opts to move the pawn instead.

Test 11: safetotake.in



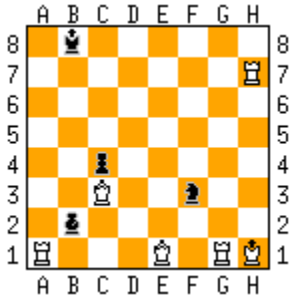
White goes first with the board set up as such. The white knight at a2 is in danger from the pawn at b3. Although at first glance it seems that this pawn is defended by the black rook, that rook is unable to move because it is defending a check from the bishop at h1. This means that it is safe to take this pawn with the white rook at h3, since black cannot move its rook to trade on the white rook.

Test 12: unsafetotake.in



This is identical to test 11, however now the black rook is no longer pinned, meaning the white rook can't safely take the black pawn. This means that the computer player level 3 instead opts to move the knight out of danger.

Test 13: diff3prioritizecheckmate.in



Finally, since checkmates are the ultimate win condition, even when presented with a situation where it can save either of 2 queens or a rook, it instead prioritizes the checkmate move from g1 to g8 to win the game.

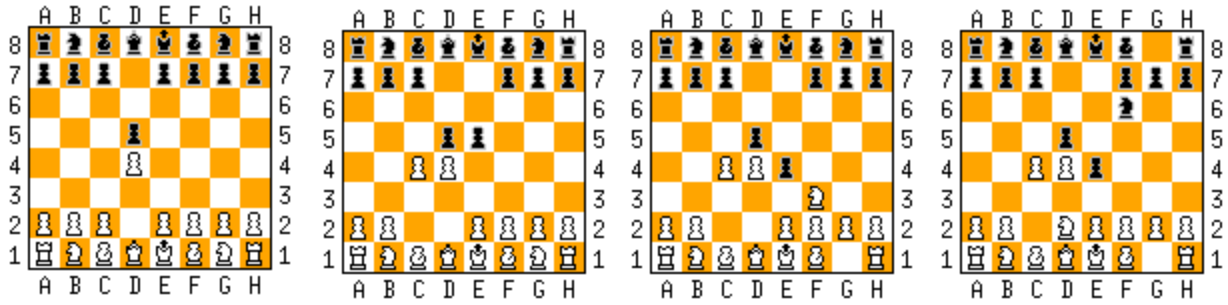
Computer Player 4:

- Computer Player 4 has a collection of possible standard openings and setups it can choose from depending on the opponent's chosen moves. However, it will randomly select one out of the matching openings to follow on each move. Therefore, we cannot give test cases that will be 100% recreatable, as it is always possible that the Computer Player chooses another opening out of the possible pool, changing the flow of the game and what will be outputted.
- To see the implemented openings, they are listed in the file computerplayerdiff4.cc as "openings" and "setups." To test these out, one may initialize a human player and a computer[4] player and feed the computer player the respective moves for an opening. Alternatively, you may initialize two computer[4] players and follow their move sequences to see if they are appropriately following a stored one.
- If it reaches a point where no standard opening move sequence can be followed, the computer player checks if any of its pieces are under attack, if it can capture any enemy pieces, or if it can put any enemies in check or checkmate. If so, it follows the logic of computer player level 3, which has test cases provided above.
- Otherwise, the computer player uses board control to select its move. This means it will look through all of its own pieces' possible moves, then see how many of the opponent's pieces' possible moves it can target from there, then chooses the move that leads to the most opponent moves targeted. If there is more than one move with the biggest amount, it will pick randomly from those moves. The idea behind this was to get the computer player to limit the opponent's moves while expanding its own control. However, this tends to be pretty random still, since in the early game, there are many moves that all result in the same max amount of enemy moves targeted, hence why we did not include a demo. While playing against a computer player level 4, you should find that it tends to extend out very quickly after having finished its opening sequence, in an attempt to block your potential moves.

Test 1: Albin Counter Gambit 1: albin1.in

White is Human, Black is Computer

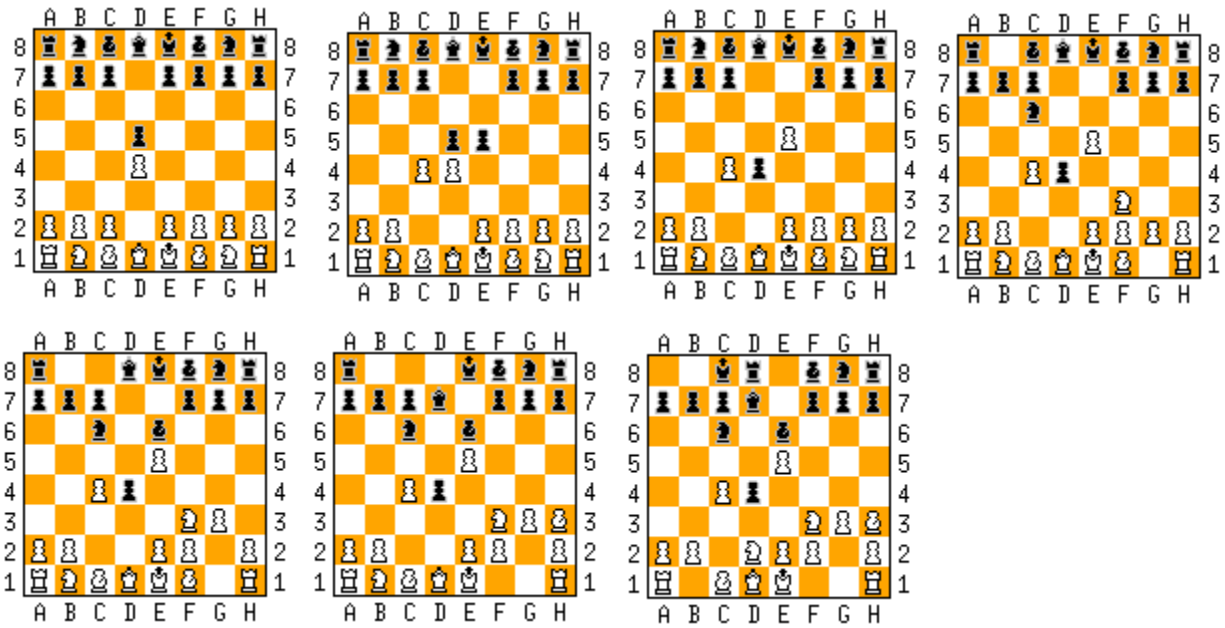
- move d2 d4
- move
- move c2 c4
- move
- move g1 f3
- move
- move f3 d2
- move



Test 2: Albin Counter Gambit 2: albin2.in

White is Human, Black is Computer

- move d2 d4
- move
- move c2 c4
- move
- move d4 e5
- move
- move g1 f3
- move
- move g2 g3
- move
- move f1 h3
- move
- move b1 d2
- move



BONUS FEATURES:

4 PLAYER SETUP (run the fourPlayerSetup.in file)

- Add a white king, try to exit
  - you can't exit setup with less one king per player, should say "black player does not have a king on the board. Must have one king per player."
- Add a black king, try to exit
  - should say "red player does not have a king on the board. Must have one king per player."
- Add a red king, try to exit
  - should say "green player does not have a king on the board. Must have one king per player."
- Add 2 green kings, try to exit
  - (you can't exit setup with more than one king, should say "green player has too many kings on the board. Maximum of one king is allowed per player.").
- Then remove one of the green kings
  - (shows functionality of - command).
- Add a white pawn to the first row, try to exit, then remove the pawn.
  - (you can't have a pawn on the first row, should say "white player has a pawn on the first/last row of the board. Invalid setup.").
- Add a white pawn to the last row, try to exit, then remove the pawn.
  - (you can't have a pawn on the last row, should say "white player has a pawn on the first/last row of the board. Invalid setup.").
- Add a white pawn to the left-most column (first row for red), try to exit, then remove the pawn.
  - (you can't have a pawn on the first row, should say "white player has a pawn on the first/last row of the board. Invalid setup.").
- Add a white pawn to the right-most column (last row for red), try to exit, then remove the pawn.
  - (you can't have a pawn on the last row, should say "white player has a pawn on the first/last row of the board. Invalid setup.").
- Add a green knight that puts the black king in check, try to exit, then remove the knight.
  - (neither king can start in check, should say "black player's king is in check. Invalid setup.").
- Add some more pieces to fill the board
- Make the current turn be the green player
  - (shows functionality of = command, should say "It is green player's turn" as opposed to the default of white).
- Use an invalid input command for +
  - (throws an error but does not crash)
  - "Invalid Input." If input does not match any of the accepted patterns ("+ *CharChar CharInt*")
  - "Specified square not in bounds!" If piece parameter matches accepted patterns, but given coordinate is not within board limits
  - "Invalid Colour character!" If piece parameter is of correct length, but does not give a valid colour.

        - "Invalid Piece character!" If piece parameter is of correct length, but does not give a valid piece.
- Use an invalid input command for -
    - (throws an error but does not crash, should say "There is no existing piece in the specified space!").
- Exit setup.
- Start a game with the following invalid inputs (should throw an error, but not crash)
    - game human
    - game human human
    - game human human human
    - game human human human human human
        - These 4 should give the error message "Invalid Player Amount."
    - game human human computer[5] human
        - This should give the error message "Invalid Player Initialization. Must be from human or computer[1-4]."
    - game human human human human
        - Will start proper 4 player game
        - (the board will have the curated setup from earlier)
- Resign 3 times starting with Green, then White, then Red
    - Should output "Black wins!" and print final scores

4 PLAYER CHECKMATE AND GAME WIN (run the fourPlayerCheckmate.in file)

- Main features this shows:
    - Game end condition when other 3 players have been checkmated
- 4 game human human human human

4 PLAYER DRAW BY REQUEST AND ZOOM (run the fourPlayerDraw.in file)

- Main features this shows:
    - Using the zoom functionality within setup and game
    - Having players request and confirm a draw
- 4
- Setup
- zoom 4 (should give error message that zoom value will make board exceed max window dimensions)
- zoom 2 (shows zoom within setup functionality)
- + Bk e1
- clear (shows clear functionality)
- + Bk e1
- + Wk h8
- + Rk j10
- + Gk i12
- done
- game human human human human
- zoom (shows zoom within game functionality)

- draw
- yes
- yes
- yes
- yes

4 PLAYER UNDO (run the fourPlayerUndo.in file)

- Main features this shows:
    - Basic undoing of multiple moves
    - Undoing a checkmate
- 4
- game human human human human
- move d2 d3
- move e13 e11
- move b4 c4
- move m7 l7
- undo 3 (show the undo confirmation for 4 players)
- yes
- yes
- yes
- yes
- move h13 h12 (now will move towards checkmate situation)
- move b4 c4
- move m9 l9
- move h2 h3
- move i13 i12
- move b9 c9
- move m7 l7
- move g1 n8 (checkmate move)
- move d13 d12
- undo 3 (will undo checkmate move)
- yes
- yes
- yes
- move m7 l7 (do those last 3 moves again to show checkmate will do the same thing)
- move g1 n8
- move d13 d12
- resign (end this game)
- resign
- resign
- game human human human human
- move f2 f3 (try to checkmate white)
- move g13 g12
- move b6 c6

- move m7 l7
- move h2 h3
- move g12 g11
- move a7 g1 (checkmate move)
- move m4 l4 (random green move
- undo 2 (move after should be red again)
- yes
- yes
- yes
- move a7 g1 (checkmate again)
- resign
- resign
- resign

4 PLAYER PAWN (run the fourPlayerPawn.in file)

- Main features this shows:
    - Moving pawns in all 4 directions
    - Double moves
    - Invalid paths
    - Pawn valid moves when in check
    - Pawns checking other players
- 4
- game human human human human
- move g2 g4 (double moves)
- move j13 j11
- move b7 d7
- move m7 l7
- move g4 g5 (normal moves)
- move j11 j10
- move d7 e7
- move m10 k10
- move g5 g6
- move j10 k9 (en-passant a piece that isn't moving directly forwards towards you)
- move e7 f7
- move n6 k9 (taking pawn with bishop)
- move g6 f7 (taking pawn with pawn)
- move e13 e12
- move a7 f7 (taking pawn with queen)
- resign
- resign
- resign
- setup (demonstrate invalid paths)
- + Wk h1
- + Wp i2

- + Wp m5
- + Bk g13
- + Bp h13
- + Rk a8
- + Rp b8
- + Rr j13
- + Gk n7
- + Gb k5
- + Gb l5
- + Gn e6
- = green
- done
- game human human human human
- move e6 c7 (put red in check)
- move i2 i3 (can't move since king will be in check)
- move i2 i4
- move i2 j3
- move i2 h3 (invalid path, nothing to take)
- move m5 m6 (just move king, only valid move)
- move h13 h12 (same thing as above)
- move h13 g12
- move h13 i12
- move h13 h11
- move g13 g12 (last black move)
- move b8 d8 (moving pawn won't take you out of check)
- move b8 c8
- move b8 c9
- move b8 c7 (taking the piece that's putting you in check, only valid move)
- move k5 j5 (being checked by a pawn, same rules apply)
- move k5 j6
- move k5 j4
- move k5 i5
- move n7 m6
- resign
- resign
- resign

4 PLAYER PAWN PROMOTION (run the fourPlayerPromotion.in file)

- Main features this shows:
    - Basic promotion by moving 8 pawns squares in
    - Case of taking a piece and promoting on same move
    - Case of taking a pawn via en-passant and promoting on same move
    - Show that promotions can be undone and redone
- 4

- game human human human human
- move f2 f4 (this first batch of moves are to demonstrate promotion by simply moving forward 8 squares)
- move g13 g11
- move b4 d4
- move m11 k11
- move f4 f5
- move g11 g10
- move d4 e4
- move k11 j11
- move f5 f6
- move g10 g9
- move e4 f4
- move j11 i11
- move f6 f7
- move g9 g8
- move f4 g4
- move i11 h11
- move f7 f8
- move g8 g7
- move g4 h4
- move h11 g11
- undo 20 (undoes all moves up until now)
- yes
- yes
- yes
- yes
- move f2 f4 (now we want to show that promotion can still occur after having been undone)
- move e13 e11
- move b4 d4
- move m10 k10
- move f4 f5
- move e11 e10
- move d4 e4
- move k10 j10
- move f5 f6
- move e10 e9
- move e4 f4
- move j10 i10
- move f6 f7
- move e9 e8
- move f4 g4
- move i10 h10
- move f7 e8 (show that promotion can occur on a standard taking move)

- move h13 h11
- move g4 h4 (red still promotes the same way before the first undo to show this promotion method remains constant)
- move h10 g11 (show that promotion can occur on an en-passant, which is unique to 4 player chess due to the promotion row still being in range of possible enemy pawn double first moves)
- undo 8 (undoes 2 moves from each player)
- yes
- yes
- yes
- yes
- move f6 f7 (redo undone moves to show that undo truly returns every piece to its past state)
- move e9 e8
- move f4 g4
- move i10 h10
- move f7 e8
- move h13 h11
- move g4 h4
- move h10 g11
- resign (white, then black, then red, so green should win)

4 PLAYER DIAGONAL MOVEMENT (run fourPlayerDiagonalMoves.in)

- Main features this shows:
    - Diagonal moving pieces cannot cross through the out of bounds corner spaces nor does check get detected through those squares
- 4
- setup (set up bishops and queens near the out of bounds corners)
- + Wk k1
- + Wb d1
- + Rk a4
- + Bk d14
- + Rq a10
- + Bb d13
- + Bb j13
- + Bp m5
- + Bp i12
- + Gq n4
- + Gb m11
- + Gk n11
- + Gp m4
- + Gp m10
- done
- game human human human human
- move d1 a4 (bishop diagonal across out of bounds should be invalid)
- move d1 e2 (valid move)

- move d13 a10 (bishop diagonal across out of bounds should be invalid)
- move j13 k14 (valid move)
- move a10 d13 (queen diagonal across out of bounds should be invalid)
- move a10 a11 (valid move)
- move m11 j14 (bishop diagonal across out of bounds should be invalid)
- move n4 k1 (queen diagonal across out of bounds should be invalid)
- resign resign resign

4 PLAYER CASTLING (run fourPlayerCastles.in)

- Main features this shows:
    - Basic kingside castling on each axis
    - Undo castling then redo
    - Try to castle with obstructed path (should be invalid)
    - Move rook then move back and try castling (should be invalid)
    - Undo rook moves then castle (should be valid)
    - Basic queenside castles
- 4
- game human human human human
- move h2 h3 (prepping for initial castle)
- move g13 g12
- move b8 c8
- move m7 l7
- move i1 h2
- move f14 g13
- move a9 b8
- move n6 m7
- move h1 j1 (try to castle while knight still in the way, then move knight)
- move j1 i3
- move g14 e14 (invalid black)
- move e14 f12
- move a8 a10 (invalid red)
- move a10 c9
- move n7 n5 (invalid green)
- move n5 l6
- move h1 j1 (valid castles now)
- move g14 e14
- move a8 a10
- move n7 n5 (valid castles end)
- undo 4 (undo the castles and try to do them again)
- yes
- yes
- yes
- yes
- move h1 j1 (castle after undo)

- move g14 e14
- move a8 a10
- move n7 n5
- undo 4 (undo them again)
- yes
- yes
- yes
- yes
- move k1 j1 (time to start moving rooks one square adjacent)
- move d14 e14
- move a11 a10
- move n4 n5
- move j1 k1 (now move the rooks back to starting spot)
- move e14 d14
- move a10 a11
- move n5 n4
- move h1 j1 (try to castle, but rooks aren't on first move anymore)
- move h1 i1
- move g14 e14
- move g14 f14
- move a8 a10
- move a8 a9
- move n7 n5
- move n7 n6
- undo 12 (undo all those rook moves so castles are valid again)
- yes
- yes
- yes
- yes
- move h1 j1 (valid castles once more)
- move g14 e14
- move a8 a10
- move n7 n5
- undo 4 (now undo to test queenside castles)
- yes
- yes
- yes
- yes
- move e2 e3 (start prepping to clear path)
- move j13 j12
- move b5 c5
- move m10 l10
- move f1 e2 (move bishops out)
- move i14 j13

- move a6 b5
- move n9 m10
- move e1 f3 (move knights out)
- move j14 k12
- move a5 c4
- move n10 l11
- move h1 f1 (try to castle before moving queen)
- move g1 h2 (move queen)
- move h14 g13 (try castle before queen)
- move g14 i14 (move queen)
- move a8 a6 (etc)
- move a7 b8
- move n7 n9
- move n8 m7 (all queens moved)
- move h1 f1 (castle move commands)
- move g14 i14
- move a8 a6
- move n7 n9
- resign (end test)
- resign
- resign

4 PLAYER CHECKMATES (run the fourPlayerCheckmate.in file)

- Main features this shows:
    - Checkmating two players in one move, then undoing, then doing it again
    - Capturing inactive, checkmated pieces, then undoing capture and checkmate
    - Stalemate
- 4
- setup (set board up for 2 checkmates in one move, then stalemate after)
- + Wk d1
- + Bk n11
- + Gk l11
- + Gp k12
- + Gp k11
- + Wq m8
- + Wr b10
- + Bp d13
- + Gp e12
- + Bp f13
- + Bp g13
- + Gp g12
- + Rp d6
- + Rp d7
- + Rk a4

- + Wq b6
- + Wb f1
- + Wr k13
- + Rr j13
- done
- game human human human human
- move m8 m10 (checkmates black and green)
- move j13 g13 (another random move to test undo)
- undo 2
- yes
- yes
- move m8 m10 (redo the checkmate move)
- move d7 e7 (random move)
- move b6 d6 (take pawn, try to get red into stalemate)
- move a4 a5 (move king)
- move d6 e7 (take other pawn)
- move a5 a4 (move king back)
- move m10 f10 (move queen into stalemate position)
- move j13 i13 (filler move)
- move k13 i13 (initiate stalemate by taking last non-king piece)