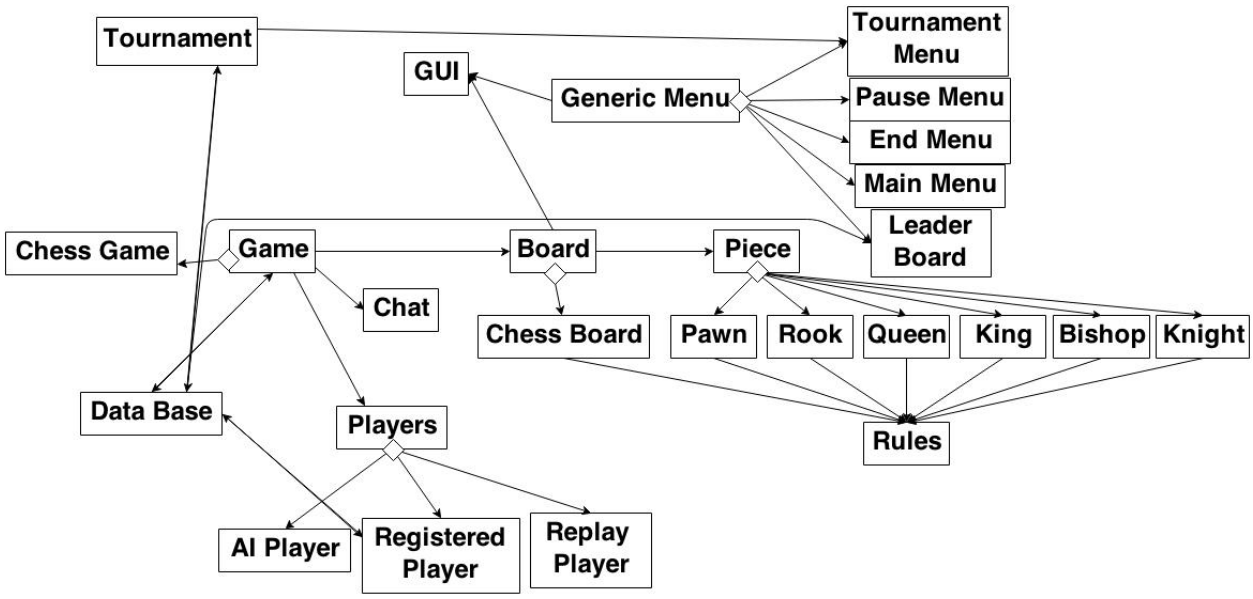
2720 Chess Design

Marko Illievski, Lukas Grasse, Brandon Robertson, Zack Shortt, Michael Wilson

Marko Illievski - Rules, Board, Pieces
Lukas Grasse - Database, Chat, Tournament
Brandon Robertson - Gui, Menus
Michael Wilson - Gui, Menus
Zack Shortt Game, Players



Project Diagram

GUI

load main menu and wait for click event

If single player ~~display ask for registered player.~~

- ~~If registered player is selected, asks for player name and then loads it from database.~~
- ~~If registered player is not selected then a generic player is loaded for the single player.~~
- Board is loaded and AI player is loaded.
- Pieces are constructed and assigned to player and AI. I Have changed this for the sake of simplicity, that the user should be defaulted to guest, until they click a Login/Register button.

- Save, Load and replay should direct them to the same screen as Login/Register if they are currently as guest.

If multiplayer is selected then player1 ~~is asked if registered player.~~
- ~~If registered player is selected, asks for player name and then loads it from database.~~
- ~~If registered player is not selected then a generic player is loaded for the single player.~~

Player2 is then asked if registered player.
- ~~If registered player is selected, asks for player name and then loads it from database.~~
- ~~If registered player is not selected then a generic player is loaded for the single player.~~

A game is constructed. This game has a board that is automatically initialized with pieces.
- Player1 is assigned to black and player2 to white.

GUI waits for event, either player makes a move or a menu is selected.

If a move is made GUI checks with rules to see if it's a valid move.
If not GUI waits for valid move.

If pause menu is selected then pause menu is loaded and displayed.
- If save is selected then the user is prompted for save game name.
- game is saved with name in database.
- If main menu is selected then game loads main menu.

When game is finished GUI will update player database if registered players are playing.

Generic Menu

- base class for end game menu, main menu , and pause classes.

Behaviours
- mouseEvent(int x, int y, GenericMenu* menu) - if a mouse event happens it will receive the x,y coordinates and decide if a menu was pressed. pure virtual
- keyEvent(unsigned char key) - if a keys is pressed we can make something happen. if esc is pressed exit the menu. pure virtual

- display(sf::Window &window) - Passing in a reference of the window, the class will know what it looks like, and will draw itself onto the window. This keeps from having to modify GUI when creating new windows. pure virtual
- Exit - (guest/user) - allows the user to close the window. pure virtual

Main Menu

- a class that allows the user to choose the following options
    - single player
    - two player
    - load game
    - replay
    - options
    - exit
- is a subclass of Generic Menu

Behaviours

- 1 Player -(guest/user) can play against an AI.
- 2 Player - (guest/user) - two players can play against each other.
- Load - (user) - player can load a previous game.
- Replay - (user) - player can re-watch a previous game in the database.
- Exit - player can exit the game.
- Login - Allow the player to access a profile

- mouseEvent(int x, int y, GenericMenu* menu) - if a mouse event happens it will check where it was pressed on the screen.
- keyEvent(unsigned char key) - if a keys is pressed we can make something happen. If esc is pressed exit the menu.
- display(sf::Window &window) - Passing in a reference of the window, the class will know what it looks like, and will draw itself onto the window.

Pause Menu

- a class that lets the user choose to save or exit the game. Subclass of Generic Menu.

Behaviours

- Save - (user) - player can save the game.
- Exit - (guest/user) - player can exit the game.Main - returns player to main menu
- Return - returns player the the game
- mouseEvent(double x, double y) - if a mouse event happens it will check where it was pressed on the screen.
- keyEvent(unsigned char key) - if a key is pressed we can make something happen. If esc is pressed exit the menu.

End of Game Menu

- a class for the user to see his rank and score. Subclass of Generic Menu class.

Behaviours

- Score - (user) - shows the users score on the leaderboard.
- Ranking - (user) shows where the user ranks on the list of users in the database
- Exit - (guest/user) - allows player to close the window.

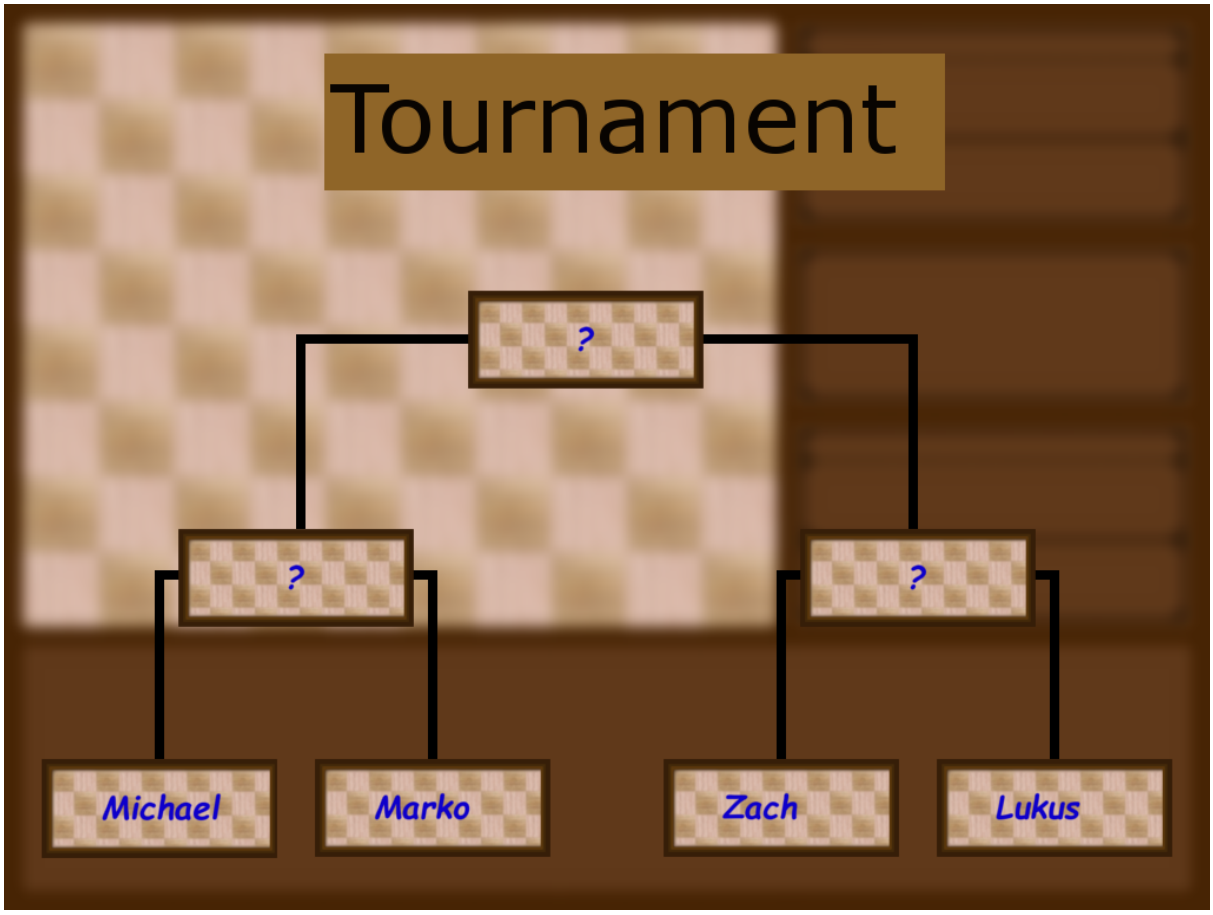
- mouseEvent(double x, double y) - if a mouse event happens it will check where it was pressed on the screen.
- keyEvent(unsigned char key) - if a key is pressed we can make something happen. If esc is pressed exit the menu.

Tournament Menu

- a class that displays the tournament standings, subclass of generic menu

Behaviours

- constructor
- destructor
- createTournament(int tournamentType) - Creates a Tournament
- loadTournament(string tournamentName) - Loads the given tournament from file
- saveTournament(string Name) - Saves open tournament to file



LeaderBoard

A class for displaying LeaderBoards.

Attributes

- vector<string> names - a vector of the player names to show on leaderboard
- vector<int> wins - a vector of ints storing wins for each player. Indices match names vector.
- vector<int> losses- a vector of ints storing wins for each player.

Behaviour

- getWins(names) - returns number of wins for a player.

- getLosses(names) - returns number of losses for a player.
- getElo(names) - returns the E.L.O for the player.

Tournament

Attributes

- vector<string> names - the names of the players still in the tournament
- map<int:string> - rank: the location of each player in the tournament. String represents name, and int represents location from top to bottom, left to right, in tournament pyramid.
- string tournamentType - a string representing the tournament type. Types are round robin or single elimination
- int tournamentSize - size of tournament. Number represents count of initial contestants.

- int currentMatch - the current match being competed for.

Behaviours

- constructor
- getNames - returns pointer to vector of names
- generateOrder - randomly assigns players to tournament
- setTournamentType(string) - sets the tournament type
- getTournamentType - returns the tournament type
- getTournamentSize - returns size of tournament
- getCurrentMatch - returns the match currently being played
- getNextMatch - returns first match available in map, based on previous winners in rank vector
- setMatchWinner - Increases rank of winning player. Adds player to spot in rank map.
- getMatchWinner - returns player with a specific rank

Game

-a base class to represent all two player board based games

Attributes

- Board - GameBoard: The main board of a game

- Player - Player1: the first player.
- Player - Player2: the second player.
- Chat - GameChat: Allows text based communication between players
- bool - gameOver: a boolean variable that is set to false until game end condition is met

Behaviours

- constructor
- virtual destructor
- getBoard - returns pointer to current board
- setBoard(Board b) - sets board
- getClock - returns pointer to current clock
- getPlayer1 - returns a pointer to player 1
- setPlayer1(Player p) - sets player1 to new player
- getPlayer2 - returns a pointer to player 2
- setPlayer2(Player p) - sets player2 to new player
- getChat - returns GameChat pointer
- newGame - initializes game to beginning state
- checkGameOver - returns state of gameOver
- setGameOver - sets gameOver to true

ChessGame

- a class for the 2 player chess game, subclass of game

Attributes

- Board - deadBoard: a board that contains pieces that have been eliminated
- ChessBoard - GameBoard: overrides GameBoard to use a ChessBoard
- vector<ChessBoard> - gameHistory: a vector containing the history of GameBoards

- vector<Board> - deadHistory: a vector containing the history of deadBoards
- bool - staleMate - set to false initially. True if in a stalemate.
- time_t p1Time - time for player 1
- time_t p2Time - time for player 2

Behaviours

- constructor
- virtual destructor
- getDeadBoard - returns pointer to dead board
- setDeadBoard(Board b) - sets dead board to parameter board
- undo - restores boards to previous state.
- getStaleMate - returns bool
- setStaleMate - sets stalemate
- pauseTime(time_t*) - pauses a players time
- resumeTime(time_t*) - resumes a players time
- getP1Time - returns time_t for P1
- getP2Time - returns time_t for P2
- getGameHistory - returns pointer to gameHistory vector
- getDeadHistory - returns pointer to deadHistory vector

Board

-a base class for boards used in games

Attributes

- vector<Piece *> - gamePieces: A vector containing all the pieces in the game
- int - height: height of board
- int - width: width of board

Behaviours

- constructor
- copy constructor
- virtual destructor
- getPiece - returns piece from vector
- setPiece - sets piece at location in vector and returns old piece that used to be in location
- getHeight - returns height
- setHeight - sets height
- getWidth - gets width
- setWidth - sets width
- movePiece(Board& b, int ix, int iy, int dx, int dy) - attempts to move piece. returns true if move is successful, false if unsuccessful. ix and iy represent the pieces initial location, dx and dy represent a piece's destination location.

ChessBoard

-a class for chess boards, subclass of board

Attributes

- int - countTurn: counts the number of turns since beginning of game.
- int - fiftyCount: counts how many moves have passed since a pawn has moved, or piece has been captured on either side

Behaviour

- getTurnCount - returns the turn count
- getFiftyCount - returns the fifty count
- resetFiftyCount - the fiftyCount is reset to 0
- incrementTurnCount - increments turnCount variable
- incrementFiftyCount - increments fiftyCount variable

Piece

-a base class for game pieces

Attributes

- int - color: the color of the piece
- Move - moveRules: A set of the rules to verify piece moves

Behaviours

- getColor - returns color
- setColor - sets piece color
- virtual getType - returns type
- validateMove(Board& b, int ix, int iy, int dx, int dy) - validates a move based on the Pieces Move attribute

PawnPiece

-a class for Pawn Pieces, subclass of Piece

Attributes

- static const string - type: the type of piece represented as a string

Behaviours

- overrides validateMove - makes a call to the appropriate move class

RookPiece

-a class for Rook Pieces, subclass of Piece

Attributes

- static const string - type: the type of piece represented as a string

Behaviours

- overrides validateMove - makes a call to the appropriate move class

KnightPiece

-a class for Knight Pieces, subclass of Piece

Attributes

- static const string - type: the type of piece represented as a string

Behaviours

- overrides validateMove - makes a call to the appropriate move class

BishopPiece

-a class for Bishop Pieces, subclass of Piece

Attributes

- static const string - type: the type of piece represented as a string

Behaviours

- overrides validateMove - makes a call to the appropriate move class

QueenPiece

-a class for Queen Pieces, subclass of Piece

Attributes

- static const string - type: the type of piece represented as a string

Behaviours

- overrides validateMove - makes a call to the appropriate move class

KingPiece

-a class for Knight Pieces, subclass of Piece

Attributes

- static const string - type: the type of piece represented as a string

Behaviours

- overrides validateMove - makes a call to the appropriate move class

Chat

-a class for chat functionality between players

Attributes

- string - player1ID: First player identifier
- string - player2ID: Second player identifier

- vector< pair<string, string> > - messages: a vector containing all messages between players. Messages are stored as a pair of strings representing the player id and the message from the player.

Behaviours

- constructor
- getPlayer1ID: return first player's id
- setPlayer1ID: set first player's id
- getPlayer2ID: return second player's id
- setPlayer2ID: set second player's id
- postMessage:post a message from a player
- getMessages: returns reference to vector of messages

Database

-a class for saving the state of the game, player names, and player statistics and for loading previous game states

Attributes

- string directoryLocation: a string containing the location of the directory to be

Behaviours

- loadGame(string gameName): loads a move by move play onto the gameHistory vector, then returns it.
- saveGame(string gameName): saves a vector of gameHistory to the gameName
- getSaveList(): returns the list of saved games found in the save game folder
- getTopPlayers(int): returns a vector of memento objects containing the top "int" number of players
- savePlayer(Memento playerMemento) - saves a memento object to given filename
- loadPlayer(string playerName) - returns memento object of player
- saveTournament(Tournament, string) - saves a Tournament to file
- loadTournament(string) - loads tournament rank map from file

Player

-A default class for unregistered, human players. Registered and AI players inherit from this class.

Behaviors

- constructor(int colour) - initializes which color the player is
- copyConstructor
- getColour - returns the color of the player

AI Player (Wishlist)

-a class for AI Player subclass of Players

Behaviours

- nextMove(Board): Generates a move and returns it to board.

Registered Player

-a class for Registered Player, subclass of Players

Attributes

- int elo: Players rank that is compared to other players.
- int gamesWon
- int gamesLost
- int tournamentsWon
- string name - name of player

Behaviors

- constructor
- destructor
- setElo(): updates the elo based on the end of game score and whether the play won or lost
- getElo(): returns calculated elo score
- getGamesWon - returns the number of games won
- incrementGamesWon - increments the number of games won
- getGamesLost - returns the number of lost games
- incrementGamesLost - increments the number of games lost
- getName - returns the name of the player.
- generateMemento() - returns a memento object
- restoreMemento(Memento memento) - restores registered player object using memento data

Memento

-a memento class for Registered Player objects

Attributes

- eloState - state of elo int
- int gamesWonState - state of gamesWon int

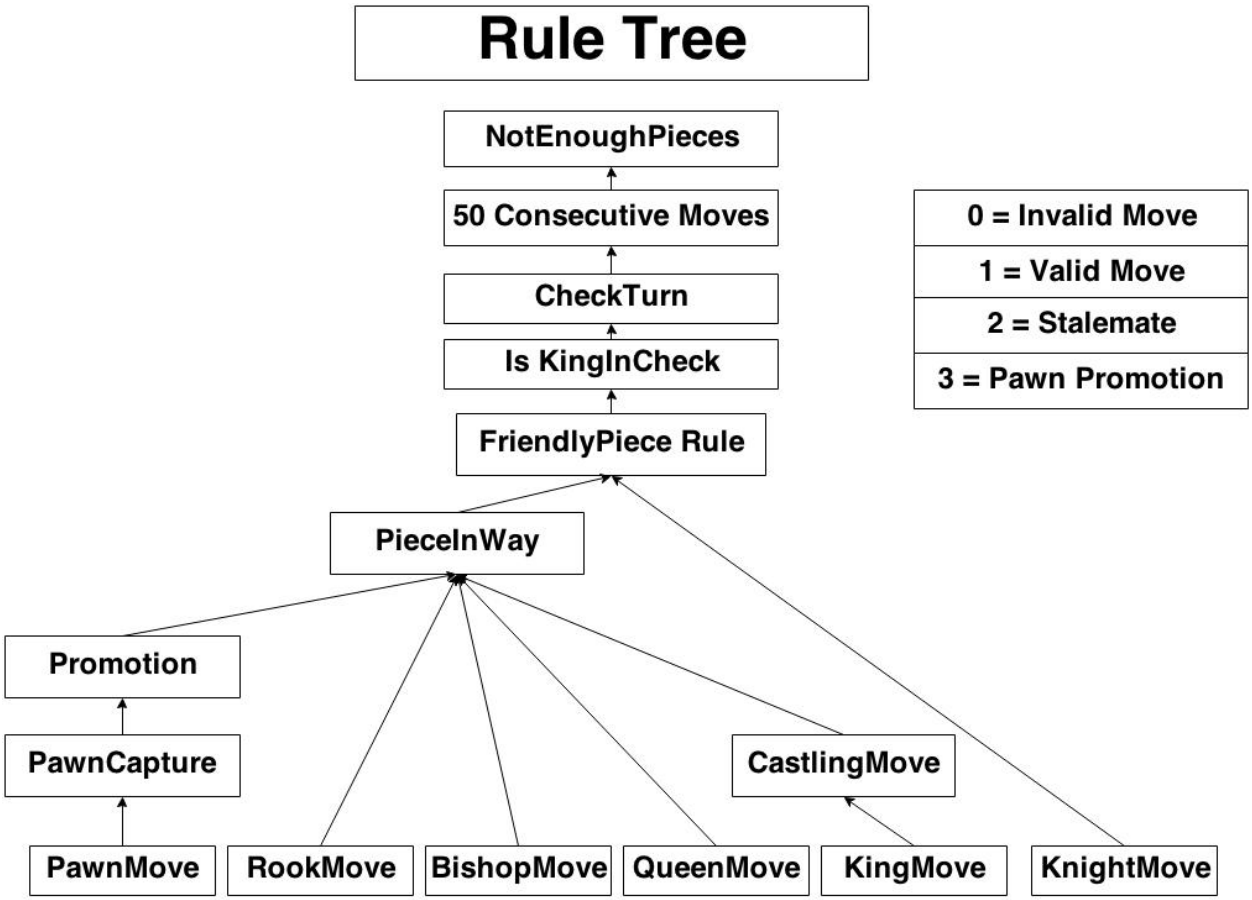
- int gamesLostState - state of gamesLost int
- int tournamentsWonState - state of tournamentsWon int

Behaviours
- constructor
- writeMemento(ostream, string) - writes itself to a file
- readMemento(string) - restores itself from a file

Replay Player

Behaviors
- nextMove(Piece): Generates a move based on history and returns it to board.

Rules



Rule

-abstract base class

- if any rules fail going up the rule chain it will tell the piece that it can not move.

Behavior

- constructor(Board& b, int ix, int iy, int dx, int dy)
- validMove()

PawnMove

-a class for Pawn Moves, subclass of Rule.

Behaviors

- overrides validMove() - checks if the move made by the user is a valid move or Pawns to make (e.i. 1 square forward or 2 squares forward if the pawn is in its original position ect.) If valid calls PawnCapture. If invalid return 0.

PawnCapture

-a class for Pawn Capture, subclass of Rule.

Behaviors

- overrides validMove() - If the player is trying to move forward go to promotion class. If that is false, check if there is an valid attack for pawn, return 1 else return 0.

Promotion

-a class for Promotion, subclass of Rule.

Behaviors

- overrides validMove() - If the location at where the pawn is requesting to move is at the end of the board return 3 (so the pawn moves to that location and gets promoted) if the location at where the pawn requested to be is not at the end of the board then go to PieceInWay.

RookMove

-a class for Rook Moves, subclass of Rule.

Behaviors

- overrides vaildMove() - If the rook requested location is either going strictly on the x axis or strictly on the y axis then go to PieceInWay else return 0.

KnightMove

-a class for Knight Moves, subclass of Rule.

Behaviors

- override validMove() - If the knights requested location is ((y+2 or y-2) and (x+1 or x-1)) or ((y+1 or y-1) and (x+2 or x-2)) then go to FriendlyPiece Rule else return 0.

BishopMove

-a class for Bishop Moves, subclass of Rule.
Behaviors

- overrides validMove() - checks if the move made by the user is a valid move for Bishop to make a perfect angle(ex. x+1, y+1). If valid calls PieceInWay. If invalid return 0.

QueenMove

-a class for Queen Moves, subclass of Rule.
Behaviors

- override validMove() - If the queens requested location is either on the x axis or on the y axis, or if it is a on a perfect angle(ex. x+1, y+1) if any are true go to PieceInWay, else return 0.

KingMove

-a class for King Moves, subclass of Rule.
Behaviors

- override validMove() - checks if the move made by the user is a valid move for the King to make (e.i. Move 1 square in any direction around the king or 2 squares towards a rook.) If valid calls CastlingMove. If invalid return 0.

CastlingMove

-a class for Castle Move, subclass of Rule.
Behaviors

- override validMove() - checks if the king is moved two squares towards a rook. If its not F the move made by the user is a valid move that sacrifices the Castling rule. If valid calls PieceInWay. If invalid return 0.

PieceInWay

-a class for Piece In Way Rule, subclass of Rule.
Behaviors

- override validMove() - checks if the there is any piece in between where the piece currently is and where it wants to move. If valid calls FriendlyPiece. If invalid return 0.

FriendlyPiece

-a class for Friendly Piece Rule, subclass of Rule.

Behaviors

- override validMove() - checks if the there is a friendly piece in the place where your current piece wants to move. If valid calls KingInCheck. If invalid return 0.

KingInCheck

-a class for King In Rule, subclass of Rule.

Behaviors

- override validMove() - checks if the king is in check. If valid call CheckTurn. If invalid return 0.

Check Turn

-a class for Checking Turn, subclass of Rule.

Behaviors

- override validMove() - checks if you can move the piece you are attempting to move (i.e black can't more opponents pieces). If valid call ConsecutiveMove. If invalid return 0.

ConsecutiveMove

-a class for Consecutive Move, subclass of Rule.

Behaviors

- override validMove() - if a pawn is moved or a piece is taken resetFiftyCount, FiftyCount is increased by one every time a white piece is moved. if FiftyCount increases to 50 moves return 2. If invalid call NotEnoughPieces.

NotEnoughPieces

-a class for Is there enough pieces, subclass of Rule.

- a class

Behaviours

- override validMove() - checks if the pieces left alive are the bishop and king or the knight and king it will return 2 else return 1.

Prebuilt Chess Puzzles - There will be some pre-configured chess puzzles, requiring the user to solve the puzzle based on the rules for standard chess games. This optional functionality will be implemented by creating a Load Chess Puzzles option on the main menu screen. This option will call the loadGame function and load the puzzles, which are saved game files stored in a distinct location from actual saved games.

Patterns:

Memento

We will be implementing the memento pattern in the Save/Load/Replay functionality. Database will store and recall memento class objects for registered players.

Chain of Responsibility

The chain of responsibility pattern will be implemented in the rules for pieces. The classes will commence a series of checks through a hierarchy of classes, committing the appropriate action should a check on any stage pass or fail.