New Bulgarian University

Bachelor Faculty

Bachelor programme Network Technologies

NETB 375 Programming Practice

AI Chess Game

Prepared by:

Vladimir Petkov, F 74482

Sofia, 2017

Content:

Project Description

Using the programming language C++ and programming framework Qt implement a computer program the famous chess game. The game must gave two modes of play: human against human (on the same computer), and human against CPU. The program must have the following features:

- fully functioning GUI that represents the chess board and all pieces;
- white piece can be moved if it is a white player's move, and the same for black pieces;
- each piece is allowed to perform only valid moves according to game rules;
- AI complexity is left to project developers, and it can be the simplest possible -- a random move.

Project Architecture Overview

The chess game is organized as a three-layer application. The three main layers are the presentation logic (GUI), the ‘chess-logic’ which contains the business logic of the game and the ‘chess-state’ which stores the current state of the game.

The presentation logic (the GUI) knows and communicates only with the ‘controller’ which is the main part of the chess-logic. This way the GUI is responsible only for visualization and interaction with the user. It does not contain any chess specific logic, it does not perform any direct changes the chess state.

The ‘chess-logic’ contains all of the logic specific to a chess game. It is formed by the Controller, CheckChecker, SpecialMovesHandler and AI classes. They are responsible for enforcing the rules of the game and changing the state.

The ‘chess-state’ represents the state of a chess game. It is formed by the State, Player, Cell, GameType and all Piece classes. They model the chess game and store all the necessary information to replicate the state of a chess game.

The chess state

As stated earlier, the ‘chess-state’ represents the state of a chess game. It is formed by the State, Player, Cell, GameType and all Piece classes. In this document the Piece classes will not be explained as they are written by Vasil Yoshev.

The ‘**Cell**’ class represents a cell from a chess board. It has a default and a copy constructor.

It has the following member fields:

- Piece* piece - A reference to a piece object. If there is no object on the cell the reference is a NULL reference.

And the following methods:

- void setPiece(Piece* piece) – A setter for the piece reference.
- Piece* getPiece() – A getter for the piece reference.

The ‘**GameType**’ is an enum that lists the possible game types:

- Not Selected - the initial value when no specific game type is selected.,
- Player vs Player - the game type when two players play against each other
- Player vs CPU - the game type when a player plays against the AI.

The ‘**Player**’ class represents a Player. It has a default and a copy constructor.

It has the following member fields:

- string name – the name of the player
- Color color – the color of the player
- bool inCheck - flag to indicate if the player is in check
- bool inCheckmate - flag to indicate if the player is in checkmate

And the following methods:

- void setName(string name) – setter for the name of the player
- string getName() – getter for the name
- void setColor(Color color) – setter for the color of the player
- Color getColor() – getter for the color
- void setInCheck(bool inCheck) – setter for the inCheck flag
- bool isInCheck() – getter for the inCheck flag

- void setInCheckmate(bool inCheckmate) - setter for the inCheckmate flag
- bool isInCheckmate() – getter for the inCheckmate flag

The '**State**' class represents the actual state of the chess game. Just like the other classes it has a default and a copy constructor.

It has the following member fields:

- vector<vector<Cell>> board – two-dimensional vector of Cell objects. This represent the chess board it is always initialized
- Player players[2] – static array of Player objects of size 2. This array contains the players of the game.
- Player* currentPlayer – reference to the player in turn
- int currentPlayerIndex – the index of the current player in the array
- TGameType gameType – the type of the game
- bool inPawnPromotion – flag that indicates if there is a pawn in promotion. A pawn is in promotion when it reaches 8^{th} rank. Then it should be changed with either a queen, rook, bishop or knight
- Coordinate pawnInPromotionCoordinates – the coordinates of the pawn in promotion

And the following methods:

- void initPieces() – which creates and places the needed pieces for the game
- void setPawnPieces(int row, Color color) – places pawn pieces on every cell of the provided row number of the provided color
- void setMajorPieces(int row, Color color) – places major pieces on every cell of the provided row of the provided color
- void setPiece(Piece* piece, Coordinate coordinate) – places the provided piece on the desired coordinates
- Piece* getPiece(Coordinate coordinate) – return the Piece object which is placed on the provided coordinates. If there is no piece on the coordinates NULL reference is returned
- Player* getCurrentPlayer() – returns a reference to the player in turn
- void nextPlayer() – changes the player in turn to the other player
- void setPlayer1(string name, Color color) – sets the name and the color of player one

- void setPlayer2(string name, Color color) – sets the name and the color of player two
- void setGameType(TGameType gameType) – sets the type of the game
- TGameType getGameType() – returns the type of the game
- bool getCheckStatusCurrentPlayer() – returns the inCheck flag for the player in turn
- bool isInPawnPromotion() – getter for inPawnPromotion flag
- void setInPawnPromotion(bool inPawnPromotion) – setter for inPawnPromotion flag
- void setPawnInPromotionCoordinates(Coordinate pawnInPromotionCoordinates) – sets the Coordinate of the pawn in promotion (setter for pawnInPromotionCoordinates)
- Coordinate getPawnInPromotionCoordinates() – returns the coordinates of the pawn in promotion if there is such, if there is not one the coordinates will be (-1, -1)

The chess logic

As stated earlier the ‘chess-logic’ contains all of the logic specific to a chess game. It is formed by the Controller, CheckChecker, SpecialMovesHandler and AI classes. In this document the Controller class and the CheckChecker class will be explained, because SpecialMovesHandler is written by Vasil Yoshev and AI is written by Atanas Chorbadjiiski.

The ’**CheckChecker**’ class implements the logic for check and checkmate checks. Since the class does not contain any state it does not have any member fields and all its methods are static.

It has the following methods:

- Coordinate locateCurrentPlayerKing(State state) – this private helper method that finds the king of the current player it will be needed later for checking for check and checkmate.
- bool checkForCheck(State state) – This method checks if the current player is in check. It first find the location of the king of the current player, finds the color of the attacking player and then checks if the cell of the current player’s king is under attack from the other player via calling isCellUnderAttack() method
- bool isCellUnderAttack(State state, Coordinate cell, Color attackingColor) – checks if the provided cell is under attack by the provided player color. It does so by checking if any of the attacking pieces has a valid move on the provided cell.
- bool checkForCheckmate(State state) – checks if the current player is in checkmate. It does so by checking if the player is check and if it has a valid move. If it has a valid move it means that the current player can escape from the chess.
- vector<Coordinate> filterCheckMoves(State state, vector<Coordinate> possibleMoves, Coordinate pieceCoordinates) – Filters the provided possibleMoves by removing any moves that places the current player under chess. To do so a copy of the current state is made, the possible move is made and then a check for check is performed.

The ‘**Controller**’ class is responsible for making all the necessary changes to the state class e.g. moving piece, changing player and so on.

It has only one member field:

- State state – this is the state of the chess game that the controller can manipulate

It has the following methods:

- void changePlayer() – private method that changes the current player. After the player in turn is updated a check for check is made. If the current player is in check a check for checkmate is made.
- void checkAndSetPawnPromotion(Piece* sourcePiece, Coordinate pieceCoordinate) – Private method that checks if the provided piece is a pawn in promotion and if this is true the corresponding changes are made in the state.
- void movePiece(Coordinate source, Coordinate target) – Moves a piece from the source coordinates to the target coordinates and changes the current player. If the move is a castle or places a pawn in promotion the necessary changes will be made.
- vector<Coordinate> getValidMoves(Coordinate click) – Returns the valid moves that the piece from the provided coordinates can make. If there is no piece on the provided coordinates empty vector is returned.
- Piece* getPiece(Coordinate coordinate) – Returns a reference to the piece on the provided coordinates. If there is no piece null reference is returned.
- Player* getCurrentPlayer() – Returns a reference to the current Player object.
- void setFirstPlayer(string name, Color color) – sets the name and color of the first player
- void setSecondPlaeyr(string name, Color color) - sets the name and the color of the second player
- void setGameType(TGameType gameType) – sets the type of the game
- void setWhitePlayerInTurn() – sets the player which color is white to be in turn
- void promotePawn(PieceType pieceType) – promotes the pawn that is in promotion to the provided type

- bool isInPawnPromotion() – returns true if there is a pawn in promotion.

The methods isLeftCastle(), isRightCastle(), doCastle() are implemented by Vsail Yoshev and he will provide the documentation for them.