SID: 520268838

Object-Oriented Design:

To develop the chess game, the first crucial step is to extend the app from PApplet. The "settings" method is utilized to create an instance of the app, and the "setup" method is used to load resources such as images and initialize elements like player pieces, CPU pieces, and the game board.

The main object-oriented programming (OOP) features employed in this game are inheritance and interfaces. Each chess piece is treated as an object and shares common properties such as position, value, speed, and images. To avoid code duplication, an abstract class called "Piece" is created to encompass these shared properties. Additionally, methods like drawing images, animation, and drawing tiles can be shared among the pieces by writing them once in the abstract class. Specific pieces extend this abstract class, overriding the "targetTile" method to define their unique movement rules.

Furthermore, certain pieces, such as the Amazon, Archibishop, General, and Chancellor, have the ability to move like a knight. To accommodate this, an interface named "KnightMove" is introduced. Pieces that possess knight-like movement extend the abstract "Piece" class and implement the "KnightMove" interface. The interface includes a default method that adds all available knight-move tiles to a list based on the provided game board. By utilizing interfaces and inheritance, different types of piece classes can be easily created and incorporated into the chess game. Creating a new piece type simply involves extending the "Piece" class and overriding one or two methods to specify its movement. This approach adheres to the principles of the Single Responsibility Principle and Dependency Inversion Principle, as modifications can be made to one part of the system without breaking the existing structure or affecting other parts.

In essence, all pieces extend the "Piece" class and implement their own movement rules. The "Tile" class denotes the location of a piece (row, column), and the "Board" class holds instances of both "Tile"(14 * 14 Tile matrix) and "Piece" classes. Therefore, the app can utilize a "Board" object to initiate the chess game.

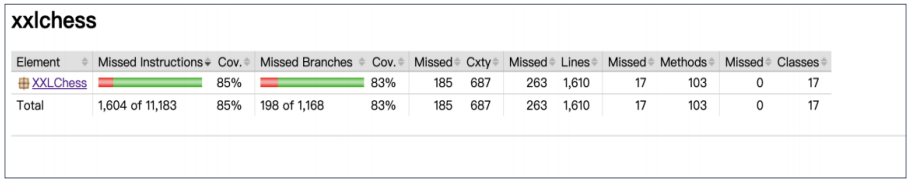
Extension:

For the extension, I have implemented sound effects and background music for this chess game to enhance the user experience. When the game starts, background music will play to create a pleasant atmosphere. Additionally, every time a chess piece is moved, a corresponding sound effect is generated, adding realism to the gameplay. Furthermore, when a piece captures another piece, a unique sound effect is produced to signify the action.

Moreover, when you either win or lose the game, specific celebratory or defeat sound

effects are played to provide feedback and immerse you in the game's outcome. If your king is in check and you attempt to move an illegal piece, a warning sound effect is triggered to alert you of the invalid move.

Furthermore, the AI has been enhanced to be more advanced. It will seize any opportunity to capture your pieces, so you need to exercise caution and think strategically when moving your own pieces.



xxlchess

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
XXLChess		85%		83%	185	687	263	1,610	17	103	0	17
Total	1,604 of 11,183	85%	198 of 1,168	83%	185	687	263	1,610	17	103	0	17

The Jacoco TestReport is given in the build/Test folder.

XXLChess

Camel
+Camel(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+addTargetTile(board : Board) : void
+targetTile(board : Board) : void

Board
-tileSize : int = 48
-playerWhite : boolean
+boardMatrix : Tile[][]
+playerCollection : Piece
+enemyCollection : Piece
+allPlayerTile : Tile
+allEnemyTile : Tile
+specialPlayerTile : Tile
+specialEnemyTile : Tile
+playerKing : King
+enemyKing : King
+Board()
+clone() : Board
-cloneBoardMatrix() : Tile[][]
-clonePieces(pieces : ArrayList<Piece>) : ArrayList<Piece>
+initializeBoard(app : PApplet) : void
+gereratePiece() : void
+setPiece(layout : String, playerIsWhite : String) : void
+playerLegalMove1(playerKing : Piece, currentCheckEnemy : Piece) : ArrayList<Piece>
+playerLegalMove2(playerKing : Piece, currentCheckEnemy : Piece) : ArrayList<Tile>
+enemyLegalMove1(enemyKing : Piece, currentCheckPlayer : Piece) : ArrayList<Piece>
+enemyLegalMove2(enemyKing : Piece, currentCheckPlayer : Piece) : ArrayList<Tile>
+aiMove() : Piece
+pawnPromte(app : PApplet, speed : double, maxMoveTime : int) : Piece
+addAllTiles() : void
+Checking(piece : Piece) : boolean

+board

App
+SPRITESIZE : int = 480
+CELLSIZE : int = 48
+SIDEBAR : int = 120
+BOARD_WIDTH : int = 14
+WIDTH : int = CELLSIZE*BOARD_WIDTH+SIDEBAR
+HEIGHT : int = BOARD_WIDTH*CELLSIZE
+FPS : int = 60
+rand : Random
+click : boolean = false
+configPath : String
+playerIsWhite : String
+layout : String
+message : PFont
+playerSpeed : double
+maxMoveTime : int
+gameOver : boolean
+playerCheck : boolean = false
+enemyCheck : boolean = false
+playerCheckMate : boolean = false
+enemyCheckMate : boolean = false
+staleMate : boolean = false
+mustDefineKing : boolean = true
+flashCount : int = 0
+flashDuration : int = 30
+enemyCheckFlag : boolean
+playerRound : boolean = true
+enemySelect : boolean = true
+playerTime : int
+enemyTime : int
+playerCounter : int = 0
+enemyCounter : int = 0
+playerIncrement : int
+enemyIncrement : int
+rightCastle : boolean = false
+leftCastle : boolean = false
~bgmClip : Clip
~goClip : Clip
~loseClip : Clip
~eatClip : Clip
~selectClip : Clip
~winClip : Clip
~checkClip : Clip
+targetTile : Tile
+enemyTargetTile : Tile
+board : Board
+playerPiece : Piece
+enemyPiece : Piece
+currentMovingPiece : Piece
+currentMovingEnemy : Piece = new Pawn(0,0,1,true,new Tile(0,0))
+rookCastleTile : Tile
+App()
+settings() : void
+setup() : void
+keyPressed() : void
+mousePressed(e : MouseEvent) : void
+draw() : void
+main(args : String[]) : void

+boardMatrix
+allEnemyTile
+specialPlayerTile
+specialEnemyTile
+allPlayerTile

Tile
<<Property>> ~row : int
<<Property>> ~column : int
<<Property>> ~x : int
<<Property>> ~y : int
<<Property>> ~piece : Piece
+Tile(row : int, column : int)
-setPiece(Piece piece) : void
-getPiece() : Piece
-setRow(int row) : void
-getRow() : int
-getX() : int
-getY() : int
-setColumn(int column) : void
-getColumn() : int

+targetTile
+rookCastleTile
+enemyTargetTile

+possibleTiles

+playerKing
+enemyKing

King
+castleLeft : boolean = false
+castleRight : boolean = false
+possibleTiles : Tile = new ArrayList<>()
+King(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+addTargetTile(board : Board) : void
+contributingCheck(board : Board) : ArrayList<Piece>
+addPossibleTiles(board : Board) : void
+targetTile(board : Board) : void

#capturedTiles
#specialTiles
#historyTiles
#IsTile
#tile

#castlerookRight
+playerCollection
+enemyCollection

Piece
<<Property>> #x : int
<<Property>> #y : int
<<Property>> #value : double
<<Property>> #isPlayer : int
#isWhite : boolean
#isAlive : boolean
<<Property>> #speed : double
<<Property>> #maxMoveTime : int
#isPressed : boolean
#need_clear : boolean = false
#differenceX : int
#differenceY : int
#moved : boolean
#pin : boolean
#sprite : PImage
#tile : Tile
<<Property>> #IsTile : Tile
#capturedTiles : Tile
#historyTiles : Tile
#castlerookLeft : Piece
#castlerookRight : Piece
#specialTiles : Tile
+Piece(x : int, y : int, value : double, isPlayer : int, isWhite : boolean, tile : Tile)
+clone() : Piece
+setImage(app : PApplet) : void
+targetTile(board : Board) : void
+topBottom(board : Board) : void
+leftRight(board : Board) : void
+leftBottomTopRight(board : Board) : void
+leftTopbottomRight(board : Board) : void
+Pin(board : Board) : void
+checkPin(board : Board) : void
+tick(targetTile : Tile, board : Board) : void
+check(app : PApplet) : void
+draw(app : PApplet) : void
+getIsWhite() : boolean
+die(isAlive : boolean) : void
+getIsAlive() : boolean
+press() : void
+unpress() : void
+getPress() : boolean

#castlerookLeft

+playerPiece
+currentMovingPiece
+currentMovingEnemy
+enemyPiece
-piece

Pawn
+Pawn(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+addTargetTile(board : Board) : void
+targetTile(board : Board) : void

Rook
+Rook(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+targetTile(board : Board) : void
+topBottom(board : Board) : void
+leftRight(board : Board) : void

Chancellor
+Chancellor(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+targetTile(board : Board) : void

<<Interface>>
RookMove
+rookAddTiles(board : Board, tile : Tile, IsTile : ArrayList<Tile>, capturedTiles : ArrayList<Tile>, isWhite : boolean, specialTiles : ArrayList<Tile>) : void

Queen
+Queen(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+targetTile(board : Board) : void

Amazon
+Amazon(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+targetTile(board : Board) : void

Bishop
+Bishop(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+addTargetTile(board : Board) : void
+targetTile(board : Board) : void
+leftTopbottomRight(board : Board) : void
+leftBottomTopRight(board : Board) : void

General
+General(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+addTargetTile(board : Board) : void
+targetTile(board : Board) : void

<<Interface>>
KnightMove
+knightAddTiles(board : Board, tile : Tile, IsTile : ArrayList<Tile>, capturedTiles : ArrayList<Tile>, isWhite : boolean) : void

Archbishop
+Archbishop(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+targetTile(board : Board) : void

Knight
+Knight(x : int, y : int, isPlayer : int, isWhite : boolean, tile : Tile)
+setImage(app : PApplet) : void
+targetTile(board : Board) : void