CTT523 – Lập trình ứng dụng JAVA

LẬP TRÌNH JAVA

Báo cáo đồ án cuối kì

MỤC LỤC

1 Thông tin nhóm

MSSV:	**1412477**	**1412363**	**1412197**
Họ tên	Đoàn Hiếu Tâm	Trần Thị Nhã	Đoàn T. P. Huyền
Email:	nhoxbypass@gmail.com	tranthinha160296@gmail.com	dtphuyen2506@gmail.com
SĐT:	01684934109	01689040391	0969938215

2 Mô tả sản phẩm

Tên trò chơi: **Arthur Chess**

Ngôn ngữ lập trình: **JAVA.** Môi trường: **JetBrains IntelliJ IDEA**

Github repository: https://github.com/USAssignmentWarehouse/Chess/

Link google drive: https://drive.google.com/file/d/0BzFMOfS_Fr2GVVlSU3dMb01MeWM/

Link Youtube demo: https://youtu.be/KleUFJU1mj4

1. Tổng quan

Arthur Chess là một game cờ vua đặc sắc được viết bằng ngôn ngữ **Java**. Trò chơi cho phép người chơi ở nhiều chế độ khác nhau:

- Chơi một mình
- Chơi với bạn bè trên cùng một client
- Chơi với bạn bè trên 2 client thông qua mạng LAN
- Chơi với máy

Arthur Chess đêm đến cho bạn một trải nghiệm hoàn toàn mới lạ về bộ môn cờ vua. Với giao diện được thiết kế tỉ mỉ, bắt mắt, đậm chất **Material Design**. Vì đó người chơi sẽ không cảm thấy mệt mỏi hay chán nản ngay cả khi chơi nhiều giờ liền.

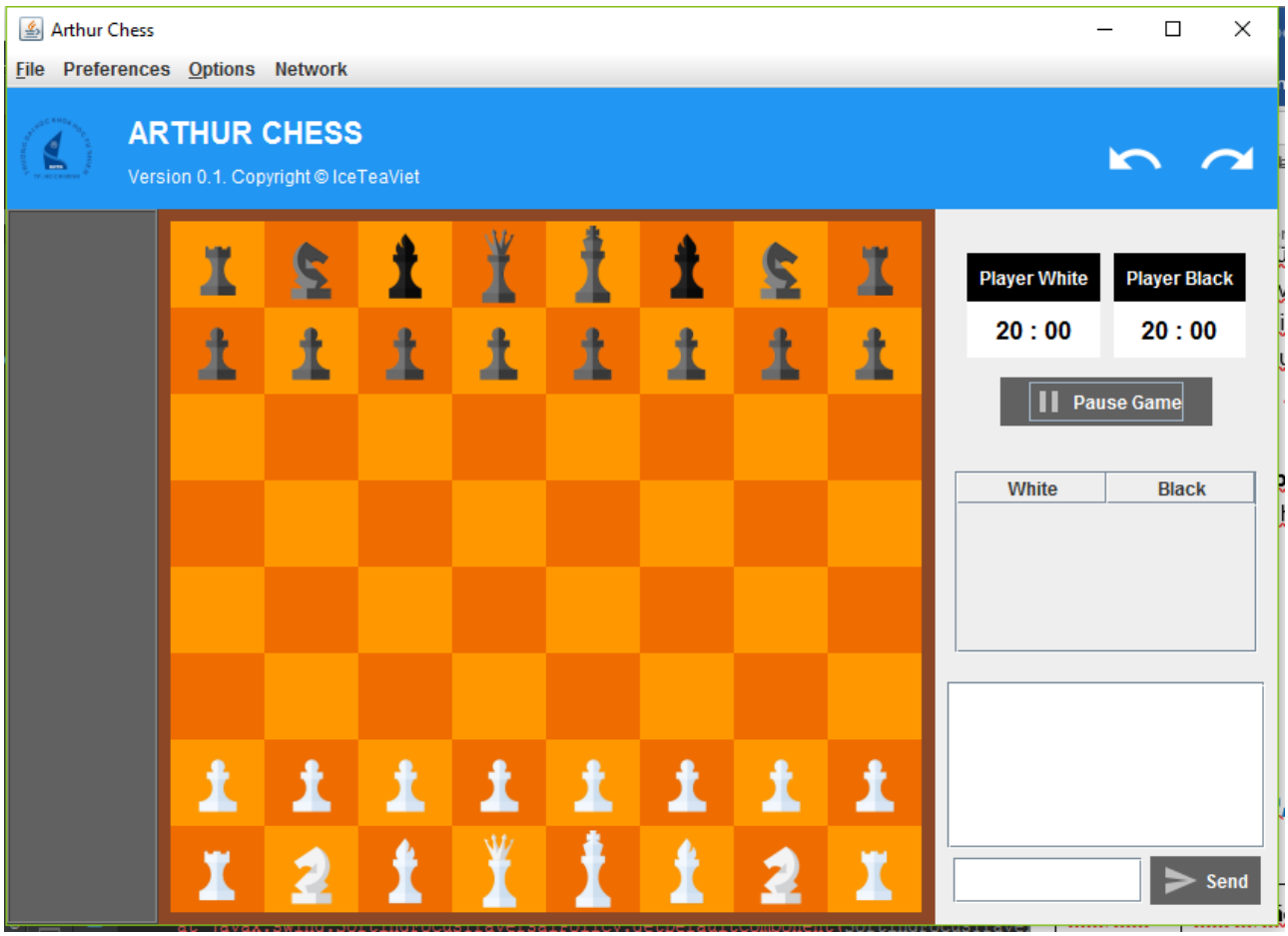
2. Các chức năng tiêu biểu

- Giao diện đồ họa đẹp mắt
- Chơi game nhiều chế độ.
- Luật chơi chuẩn quốc tế, đảm bảo đầy đủ các tính năng (EnPassant, King Castled,..)
- Hỗ trợ tối đa 02 người chơi qua mạng LAN.
- Hỗ trợ chat giữa các người chơi qua mạng LAN.

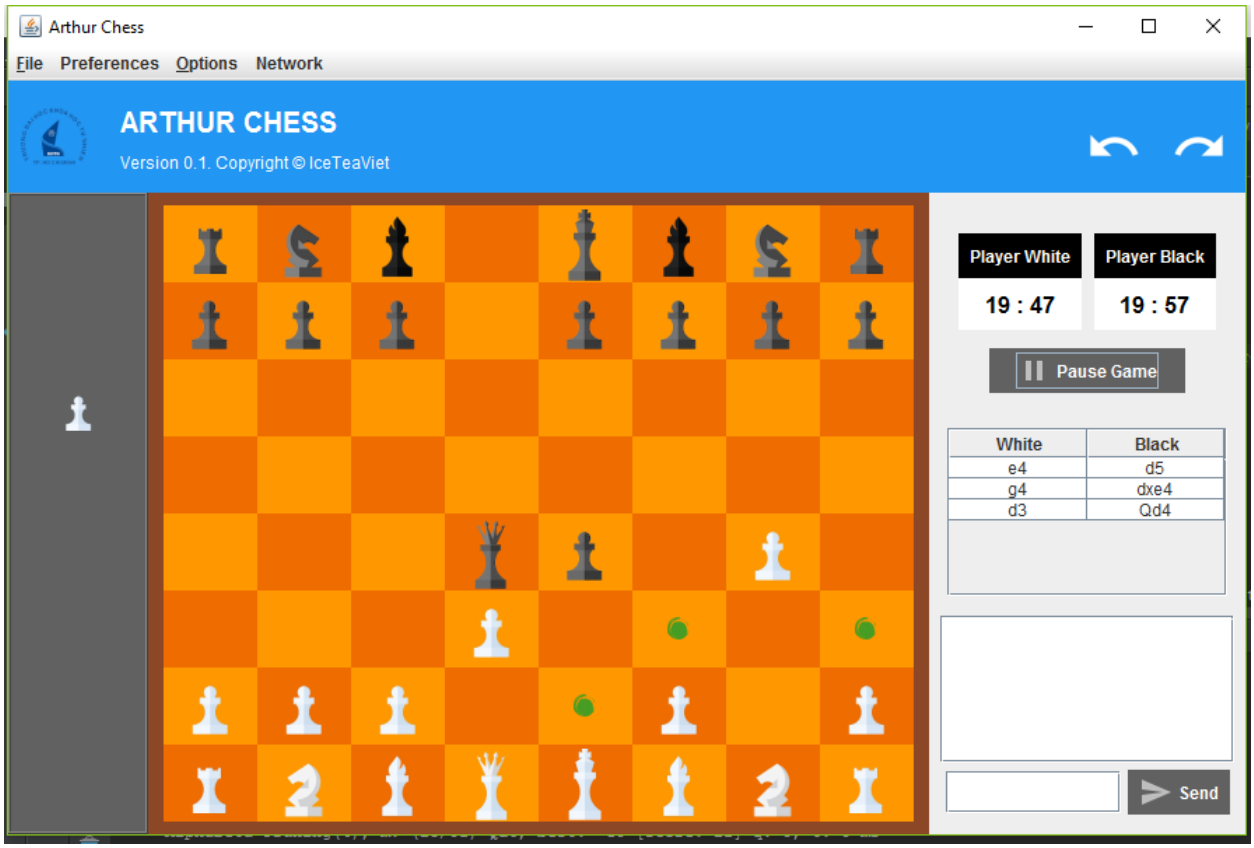
- Có máy đóng vai trò người chơi. (Sử dụng thuật toán MiniMax và AlphaBeta Pruning).
- Thay đổi tùy biên màu sắc của bàn cờ.
- Hỗ trợ undo nước đi, ghi lại lịch sử nước đi, gợi ý nước đi hợp lệ.
- Và nhiều chức năng nhỏ khác.,..

3. Mô tả cách hoạt động

- Giao diện chính của chương trình

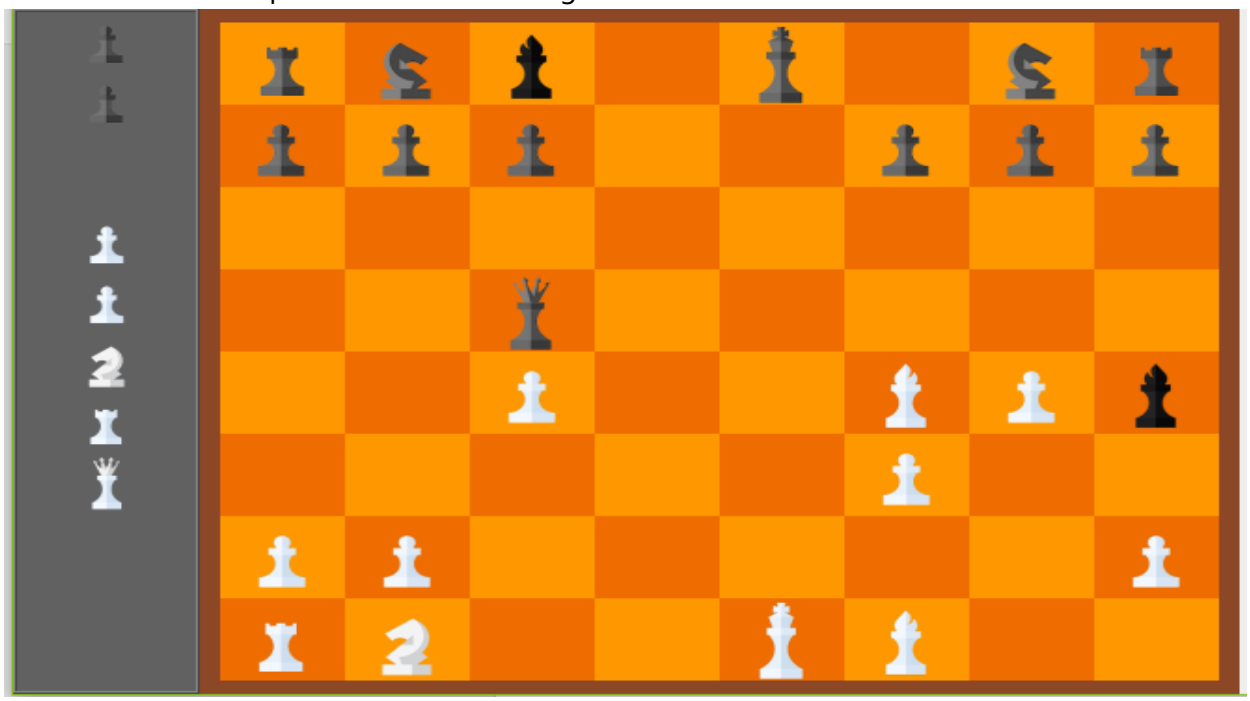


- Giao diện chơi game, mặc định khi khởi động game, chương trình sẽ cho bạn đấu với AI, chọn một quân cờ bất kì, bàn cờ sẽ hiện lên **gợi ý các nước đi có thể đi được** của quân cờ vừa chọn.

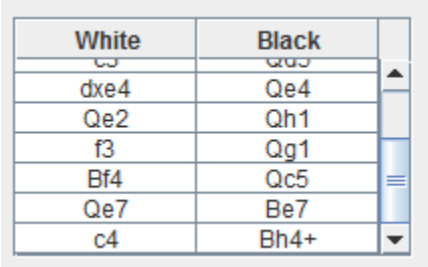


Chọn vào 1 trong các dấu chấm gợi ý nước đi hợp lệ để đi tiếp.

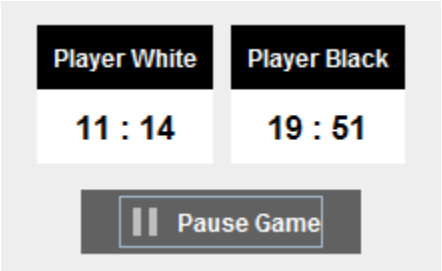
- Xem danh sách các quân cờ đã mất ở khung bên trái bàn cờ



- Xem danh sách các nước đã đi ở khung bên phải bàn cờ

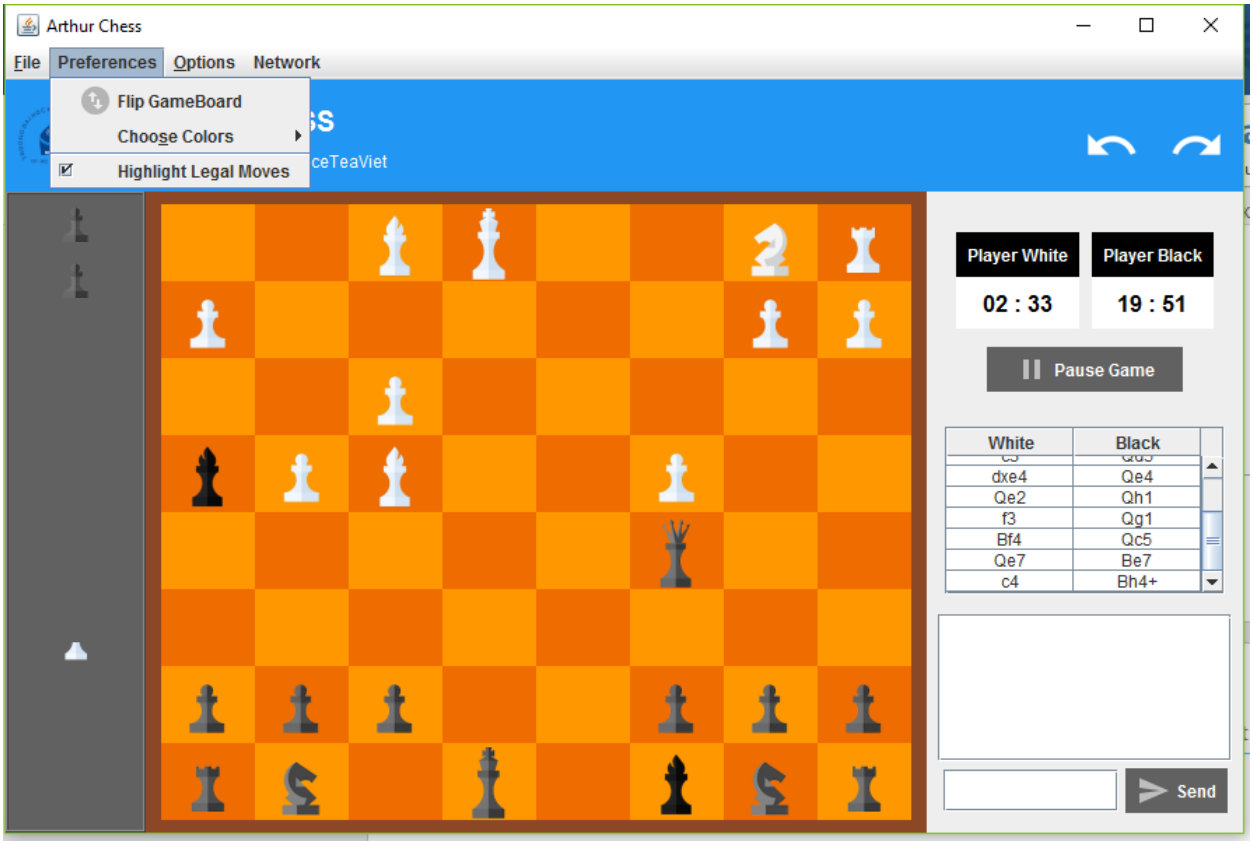


- Xem thời gian còn lại của từng người chơi ở khung bên phải bàn cờ.



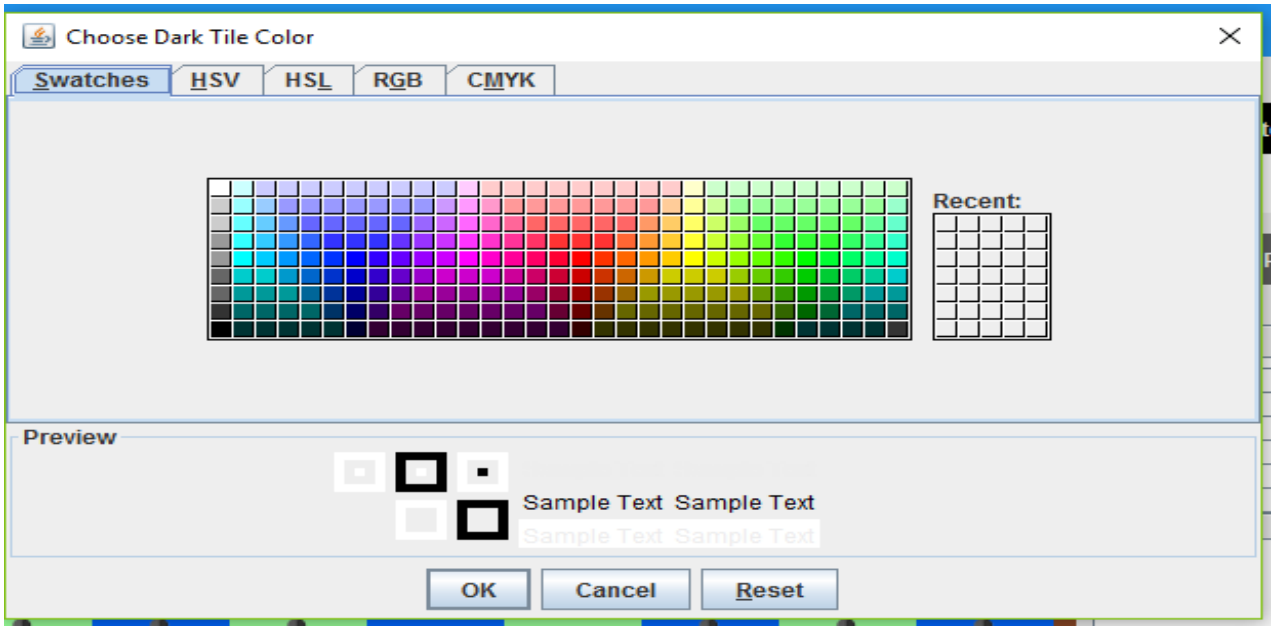
Đồng thời, người chơi có thể **pause game** bất cứ khi nào muốn.

- Chức năng **Flip board** cho phép đảo ngược vị trí, tọa độ và tất cả quân cờ đang có trên bàn cờ



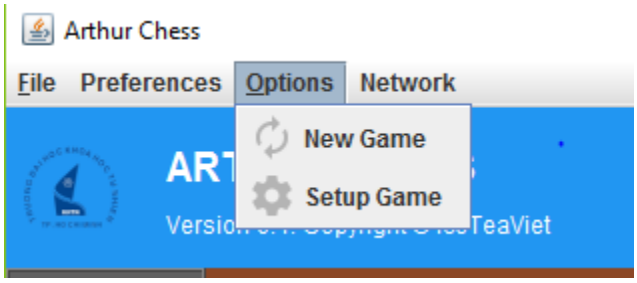
- Menu cho phép thay đổi màu nền các ô của bàn cờ

Dialog chọn màu

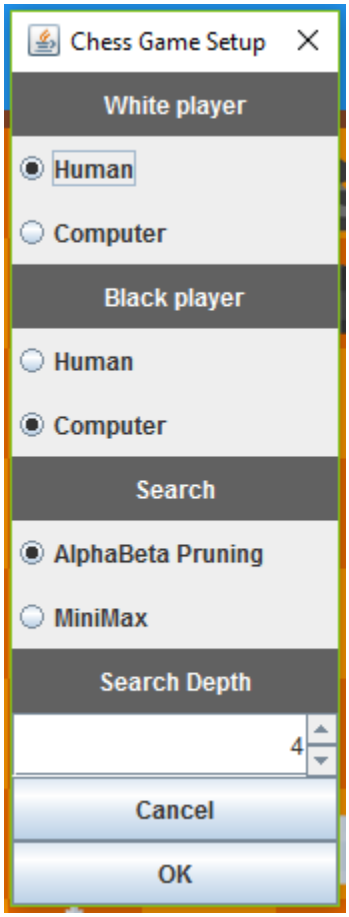


- Tùy chọn **Highlight legal moves** mặc định được bật để hiện gợi ý các nước đi hợp lệ cho người chơi dễ sử dụng. Có thể tắt nó bằng menu bar.

- Có thể **Re-new** lại game bằng cách chọn **New Game** trong menu bar



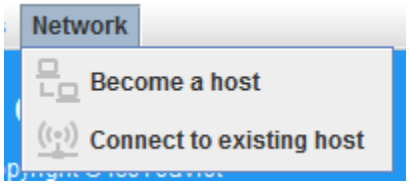
- Mở dialog game setup lên từ menu chính để **thiết lập lại các cài đặt của game**.



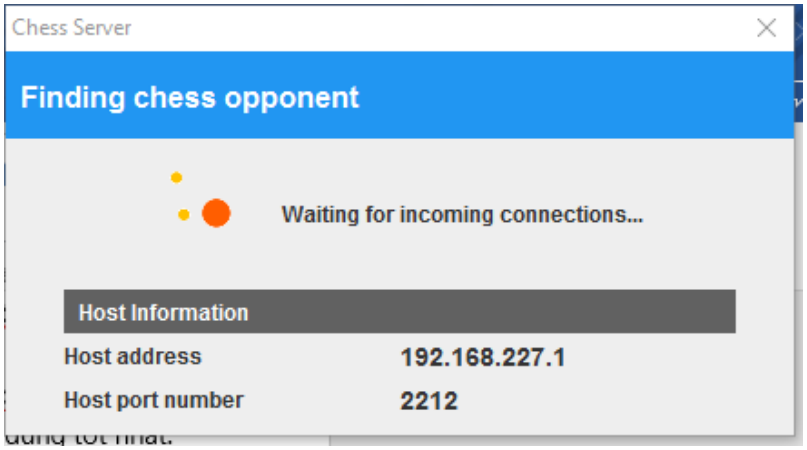
Trong dialog này, người chơi có thể chọn vai trò cho quân cờ trắng/đen là người điều khiển hay máy điều khiển.

Đối với khi được máy điều khiển, ta có thể chọn thuật toán mà nó dùng để tìm ra nước đi tốt nhất là **MiniMax** hay **AlphaBeta Pruning**. **Khuyến nghị nên chọn thuật toán cắt tỉa AlphaBeta** bởi vì nó sẽ nhanh hơn MiniMax trong nhiều trường hợp hơn.
Người chơi có thể tùy chỉnh độ sâu của phép tìm kiếm nước đi tốt nhất, độ sâu càng lớn thì AI chạy càng chậm. **Khuyến nghị nên chọn độ sâu <= 4** để có trải nghiệm người dùng tốt nhất.

- Menu **network** dùng để thực hiện các thao tác mạng cho game.

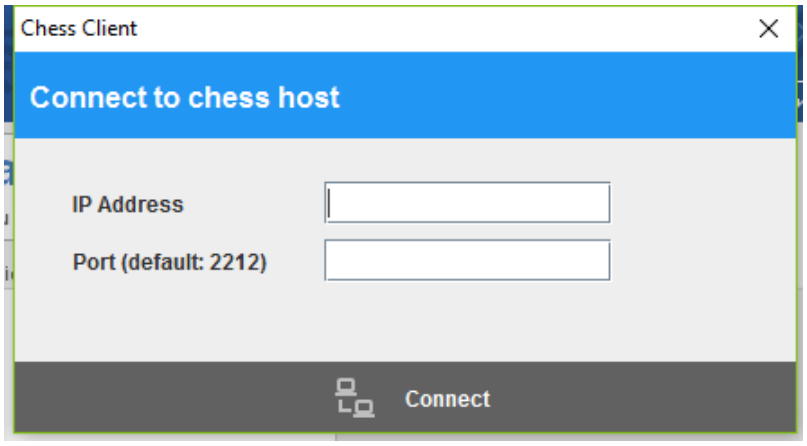


Chọn vào **Become a host** để biến mình trở thành một host game để chơi cùng với bạn bè.



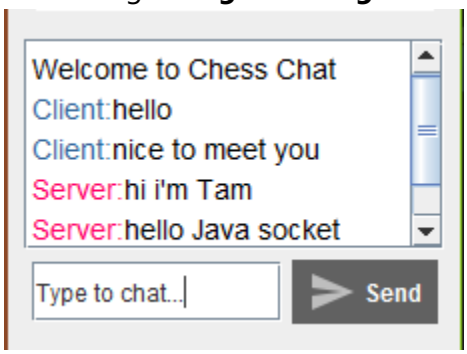
Khi chọn trở thành một host, một dialog **Chess Server** sẽ được bật để chạy **SocketServer** và chờ kết nối đến từ client. Dialog này cho người chơi thông tin về địa chỉ **IP** và **port** của host để gửi đến các người chơi muốn kết nối.

Khi chọn trở thành một client để kết nối đến một server, dialog của **Chess Client** sẽ xuất hiện

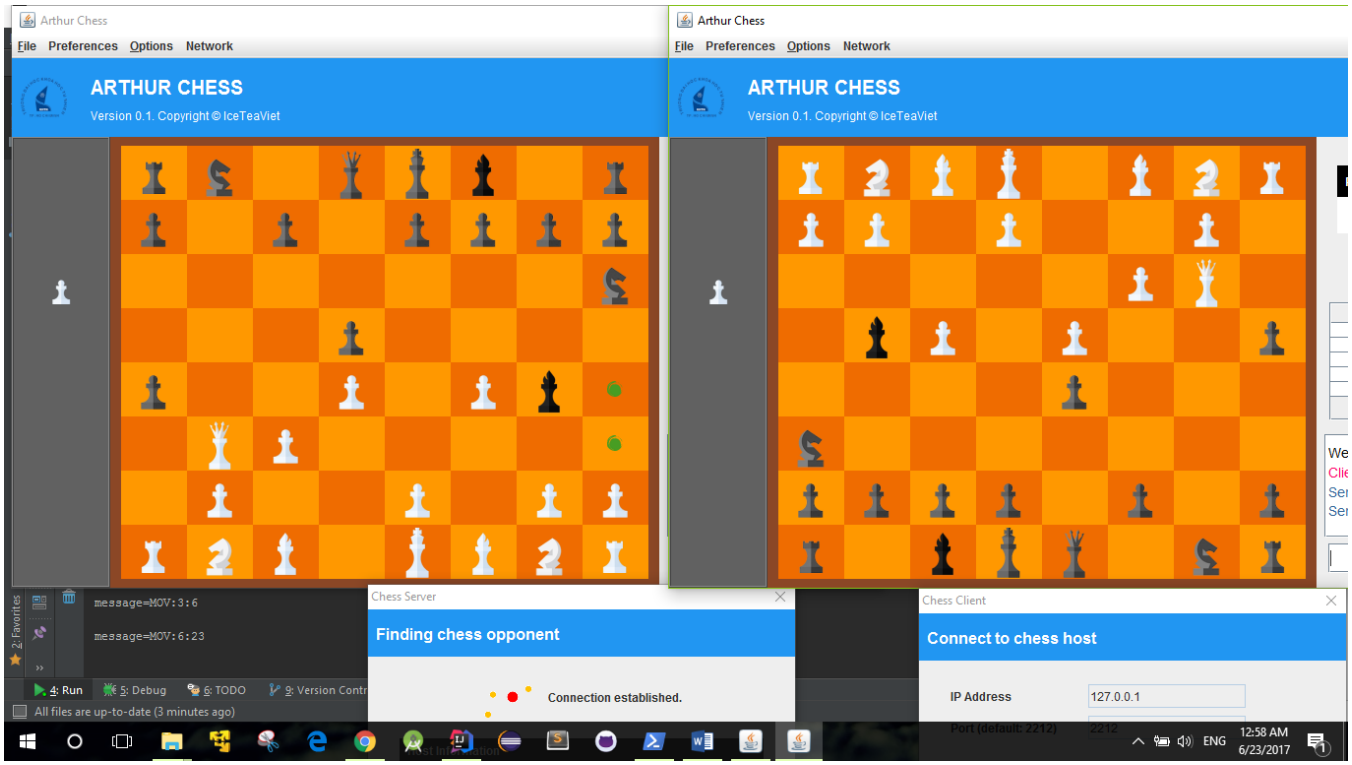


Người chơi có thể nhập vào địa chỉ IP và port của người host để kết nối đến host đó. Nếu để trống địa chỉ IP, chương trình sẽ lấy IP mặc định của **localhost** (**chạy 2 client trên cùng 1 máy**) để kết nối. Tương tự port mặc định là **2212**.

- Chức năng **chat giữa hai người chơi** nằm ở khung bên phải bàn cờ



Khi kết nối giữa client-server chess đã được thiết lập, khung chat được kích hoạt và người chơi có thể chat với nhau.

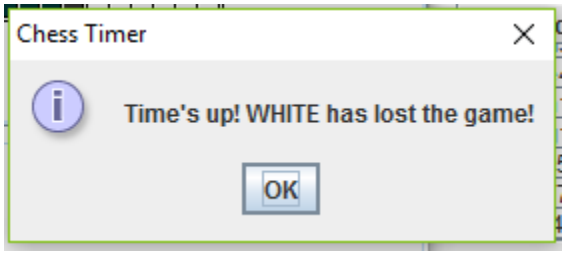


Mọi thứ trên bàn cờ đều được đồng bộ giữa hai người chơi, từ nước đi đến vị trí quân cờ, trạng thái, các quân cờ đã bị tiêu diệt và timer.

Lưu ý: Để nâng cao trải nghiệm người dùng, quân cờ của người chơi luôn nằm phía dưới màn hình. Nghĩa là Host-A được cấp cờ màu trắng, thì ở màn hình của Host-A quân trắng sẽ nằm dưới màn hình, Client-B kết nối đến được cấp cờ màu đen thì ở màn hình của Client-B quân đen sẽ nằm phía dưới màn hình.

- Người chơi có thể **Undo Redo** nước đi bằng hai nút trên Toolbar

- Khi một người chơi bất kì hết thời gian, dialog xuất hiện thông báo người thắng/thua cuộc.



- Tương lai sẽ hỗ trợ thêm các tính năng như: Lưu ván đấu lại thành file **PGN**, và load ngược lên lại để tiếp tục chơi, **cải thiện và tăng tốc độ xử lý cho AI**, cho phép nhập **html link** vào khung chat,..

3 Cách thực hiện chương trình

Trò chơi **Arthur Chess** được xây dựng bằng ngôn ngữ **Java**, gồm các bước chính sau:

1. Xây dựng bộ lõi cho game cờ vua (Chess Core Engine)

Là thành phần quan trọng nhất của game, nên quá trình xây dựng chess engine gồm nhiều giai đoạn:

- Đầu tiên là thu thập kiến thức về game cờ vua, bao gồm luật chơi, cách xử thắng thua, các loại bàn cờ, màu sắc,.. Nhóm đã sưu tầm tài liệu trên Wikipedia về luật cờ vua, kèm theo kiến thức cờ vua vốn có.
- Tiến thành khảo sát các phần mềm tương tự trên thị trường bằng cách chơi thử các game cờ vua trên **Zing Play**, **chess.com**,.. để phân tích những gì nên tái sử dụng và những gì nên cải thiện từ các game đó.

Từ hai thứ trên, thu được **SRS** sau đó tiến thành thiết kế kiến trúc của bộ lõi game engine. Bộ lõi game engine được thiết kế theo **phương pháp lập trình hướng đối tượng** (**OOP**).

- **Quân cờ:**

Những thứ thuộc về quân cờ được chứa trong package **core.piece.***
Mỗi loại quân cờ sẽ được tạo riêng một class ứng với tên của
quân cờ đó (**King**, **Pawn**, **Bishop**,..). Các class trên sẽ được kế thừa từ một base class là **Piece.java**. Class **Piece** sẽ chứa các thông tin cơ bản của một quân cờ như: loại cờ, màu cờ (loại đồng minh), vị trí của quân cờ trên bàn cờ,.. kèm theo các hàm get/set, hàm tính toán
các nước đi hợp lệ và hàm di chuyển quân cờ. Các class kế thừa lại sẽ implement các hàm này riêng biệt cho từng loại quân cờ, vì mỗi loại sẽ có nước đi, khả năng và chiến thuật riêng.
Trong class của từng loại quân cờ sẽ lưu một mảng **MOVE_VECTOR_CANDIDATE_COORD** để chứa các vector di chuyển
tương ứng với loại quân cờ đó. Từ những vector tọa độ đó mà hàm **calculateLegalMove()** tính toán và gợi ý được cho người chơi những nước đi nào là hợp lệ, và hàm **move()** có thể cài đặt để di chuyển tro ng các nước đi hợp lệ của quân cờ đó.
Đối với danh sách các loại quân cờ, nhóm taọ một class gọi là **PieceType** để lưu lại tên và giá trị của qu ân cờ (giá trị dùng để tính điểm khi sử dụng AI). Sau đó dùng list **enumerator** để lưu lại,
tương ứng với mỗi loại quân cờ sẽ có một biến enum riêng kiểu **PieceType**.
Đối với màu cờ, ta sử dụng một enumerator Alliance.java gồm hai loại là WHITE và BLACK để lưu màu của đồng minh, màu của kẻ thù,.

- **Người chơi:**

Những thứ thuộc về người chơi được chứa trong package **core.player.***
Bao gồm base class **Player.java** chứa các thông tin cơ bản của môt người chơi như: list các nước đi hợp lệ, màu cờ, đã đang bị chiếu tướng hay không,...
WhitePlayer và **BlackPlayer** là hay class kế thừa từ **Player**, đại diện cho người chơi của hai màu cờ trắng/đen, lớp này cài đặt lại các hàm của base class dùng để quản lý các quân cờ của người chơi đó.
Move và **MoveTransition** là các class đại diện cho một nước đi, gồm có các thông tin như: thứ tự của ô xuất phát, thứ tự của ô đích đến, biến chứa giá trị của quân cờ đang được di chuyển, trạng thái của việc di chuyển đó,...
MoveStatus thể hiện thông tin về việc nước đi đã thành công chưa, nước đi có thể thất bại do nhiều lý do như: nước đi không hợp lệ, nước đi vị phạm thread-safe,...
Alliance là một enumerator chứa 2 đối tượng tượng trưng cho 2 phe trắng và đen, kèm theo các hàm và các thuộc tính cần thiết của đối tượng đó.

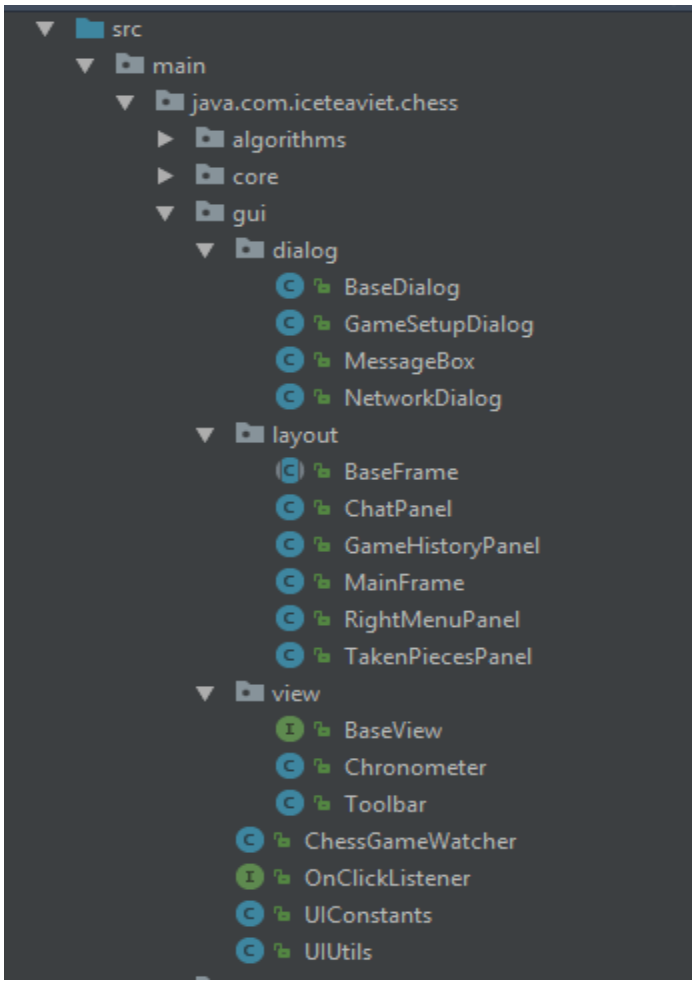
- **Bàn cờ:**

Những thứ thuộc về bàn cờ được chứa trong package **core.board.***
Đầu tiên là class **Tile.java** dùng để tượng trưng, chứa các thông tin của một ô trên bàn cờ kèm theo tọa độ của ô đó trên bàn cờ. Có hai loại **EmptyTile** và **OccupiedTile** kế thừa từ base class **Tile** tượng trưng cho ô chưa có hay đã có quân cờ.
Tiếp theo là một class cực kì quan trọng **GameBoard.java** giữ nhiệm vụ quản lý tất cả mọi thứ liên quan đến một bàn cờ vua. Chứa list của các ô (Tile) trên bàn cờ, list các quân cờ của mỗi bên, chứa dữ liệu của đối tượng người chơi kiểu Player. Đây là đối tượng tượng trưng cho bàn cờ, khi khởi tạo một

bàn cờ mới nó sẽ duyệt qua các Tile và vẽ ra màn hình thành đồ họa bàn cờ, đồng thời chứa các hàm tính toán vị trí của quân cờ,..
Cuối cùng là class helper tên là **BoardUtils.java**, dùng để chứa các hàm helper dành cho việc xử lý logic cho ChessBoard như hàm chuyển đổi giữa Chess Notation (tọa độ của quân cờ trên bàn cờ theo chuẩn **Algebraic Chess Notation**) sang chỉ số chỉ vị trí của quân cờ trên bàn cờ, khởi tạo dữ liệu cho hàng và cột của bàn cờ, lấy N nước đi gần đây nhất, tính toán vị trí và khoảng cách của quân cờ,...
Tuy nhiên đây chỉ là bộ lõi dùng để vẽ quân cờ, bàn cờ và xử lý logic cho game. Giao diện chương trình và xử lý yêu cầu của người dùng sẽ được xây dựng riêng ở giai đoạn xây dựng giao diện

2. Xây dựng giao diện

Giao diện đáp ứng đúng yêu cầu của **View** trong **MVC** và **Observer pattern**, chỉ tiếp nhận các yêu cầu từ người dùng và gọi các hàm từ các class manager, wrapper, và lõi Chess Engine để sử dụng.
Giao diện được xây dựng và thiết kế kiến trúc cẩn thận để có thể tái sử dụng các thành phần và source code. Để làm được điều đó ta sử dụng **OOP** vào việc xây dựng giao diện, ở đây chia thành ba loại: **Layout**, **View** và **Dialog**. Mỗi loại sẽ có một base class tương ứng, và sẽ được cài đặt những thông số, thành phần đặc trưng như: width, height, title, LayoutManager, Toolbar,..



Từ những base class đó mà xây dựng lên những thành phần của giao diện, cụ thể như sau:

- **Layout**

Layout bao gồm class **MainFrame.java** là JFrame chính chứa tất cả các thành phần còn lại. Trong MainFrame cũng cài đặt luôn các JMenu chính của chương trình, bắt sự kiện click của người dùng và gọi các hàm wrapper để thực thi.

Ở giữa màn hình là bàn cờ được vẽ bằng bộ lõi Chess Engine.

Bên trái là khung **TakenPiecesPanel.java** dùng cài đặt các hàm để hiển thị các quân cờ đã bị tiêu diệt ra màn hình. Class này duyệt qua danh sách các quân cờ, tìm ra các quân cờ đã bị tiêu diệt và vẽ lên màn hình.

Bên phải là khung **RightMenuPanel.java**, chứa hai **Chronometer** đại diện cho hai timer của hai người chơi, nút Pause Game, khung lịch sử các nước đã đi **GameHistoryPanel** và khung chat giữa hai người chơi **ChessChat**.

  - **GameHistoryPanel.java**: chứa một **JTable**, nhận dữ liệu tọa độ sau khi người dùng thực hiện một nước đi và sử dụng các hàm utils để convert nó qua Chess Algebraic Notation để cho vào bảng.
  - **ChessChat.java**: sử dụng **JEditorPane** để hiện nội dung chat, convert nội dung chat sang html và hiển thị để hỗ trợ highlight tên người chơi. Nhận vào dữ liệu từ **JTextField** và send message thông qua **Socket** object được truyền vào khi kết nối giữa hai người chơi được khởi tạo

- **View**

View bao gồm **Chronometer** đại diện cho timer của người chơi, được sử dụng trong khung **RightMenuPanel.java** ở trên. **Chronometer** chứa một **Timer** với interval là 1000ms, khi mỗi lần tick sẽ trừ thời gian con lại đi 1 giây, sau đó format thành dạng digital clock và hiển thị lên màn hình. **Toolbar** là một ActionBar được cài đặt có thể sử dụng cho cả Layout và Dialog, có thể tùy biến hiển thị và nội dụng.

- **Dialog**

Dialog bao gồm:

  - **GameSetupDialog** là một hộp thoại dùng để thiết lập các cài đặt, thông số cho game, hộp thoại này chứa thông tin về loại người chơi của hai người chơi (Human hay Computer), cho phép người dùng chọn 1 trong 2, đồng thời nếu là AI nó cũng cho người dùng chọn thuật toán muốn sử dụng cho AI đó (MiniMax hoặc AlphaBeta Pruning) và độ sâu search depth khi chạy thuật toán tìm kiếm nước đi tốt nhất.
  - **NetworkDialog** là một hộp thoại dùng để kết nối và chơi game giữa hai game client khác nhau. Hộp thoại này chứa một biến **Boolean** gọi là **isHost**, dùng để xác định client

vừa click vào "**Become a host**" hay "**Connect to chess server**". Nếu là host thì dialog sẽ khởi tạo giao diện của host, đồng thời cho chạy **ChessServer** lên và chờ kết nối, nếu là client thì sẽ đợi người dùng nhập thông tin, sau đó chạy **ChessClient** để connect tới server. **Dialog này phải được giữ trong suốt quá trình chơi game, nếu tắt kết nối sẽ bị chấm dứt.**

- **MessageBox**: Là nơi để cài đặt các message box thuộc nhiều dạng **error**, **information**, **warning**,... để hiển thị ra với mục đích xác định.

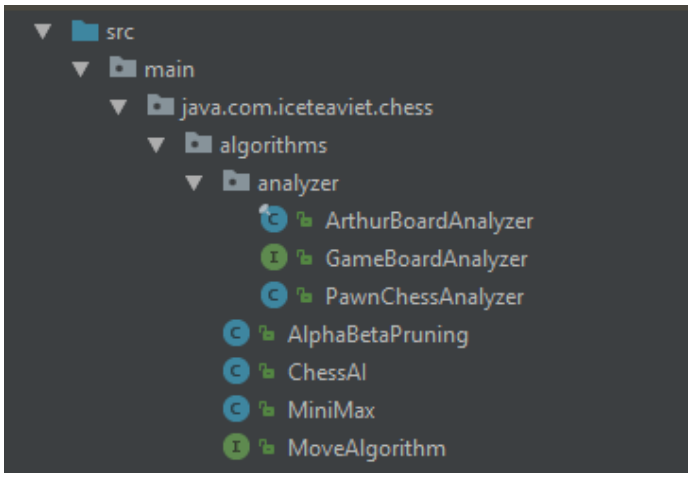
Ngoài ra còn có các class **UIConstant.java** chứa những thông tin về màu sắc và thuộc tính của giao diện, **UIUtils.java** chứa các hàm helper dùng để load hình ảnh, tạo button không viền,...phục vụ cho quá trình tạo UI, và **interface OnClickListener** dùng để xử lý sự kiện click, giao tiếp giữa các class với nhau.

3. Xây dựng các hàm và class wrapper cho bộ lõi chess game

- **ChessGameWatcher** là một **observable**, **manager** có nhiệm vụ giám sát trò chơi, giao tiếp với người dùng, lưu trữ thông tin của chương trình.
  Chứa instances của các layout có trên màn hình, chứa đối tượng GameBoard và các thông tin về chế độ chơi (PvP, PvC hay NetPlay).
  ChessGameWatcher cũng chứa các hàm wrapper của bộ lõi Chess Engine như hàm **movePiece()** để di chuyển quân cờ, **moveMadeUpdate()** và **drawBoardAfterMove()** dùng để update lại thông số bàn cờ và vẽ lại bàn cờ sau một nước đi **undoLastMove()** dùng để undo nước vừa đi, gọi ChessAI để tìm nước đi nếu đó là chế độ AI, gọi các hàm và lấy thông tin từ Chess Engine để tính toán nước đi hợp lệ,... Tất cả đều tổng hợp các tính năng từ các thành phần của game, và tổng hợp lại thành một class manager, quyết định tất cả mọi thứ về trò chơi.
- **ChessAI** là class manager cho chế độ AI của game, dùng để viết các hàm tổng hợp các chức năng của AI như: **move()** chạy một **Worker Thread** để tính toán nước đi tốt nhất và cập nhật lên bàn cờ, **getBestMove()** xét xem đang chọn thuật toán nào để chạy và lấy ra nước đi tốt nhất, **AIAsyncWorker** kế thừa từ **SwingWorker** là một background thread để chạy và xử lý cho AI, tách biệt với **UI Thread** nhằm không làm ảnh hưởng tới các chức năng còn lại của chương trình.
- **NetworkManager** là class manager cho các thao tác với network của chương trình, chứa các thông tin về trạng thái đã connect hay chưa, là host hay client, và các hàm **sendEndMessage()** gửi thông báo đóng kết nối, **closeConnection()** dùng để đóng các kết nối, in/out stream và socket, **sendMoveMessage()** dùng để gửi thông tin nước đi thông qua socket.

4. Xây dựng hệ thống AI

Hệ thông AI được xây dựng theo **OOP** cùng với các **thuật toán search và tính điểm**

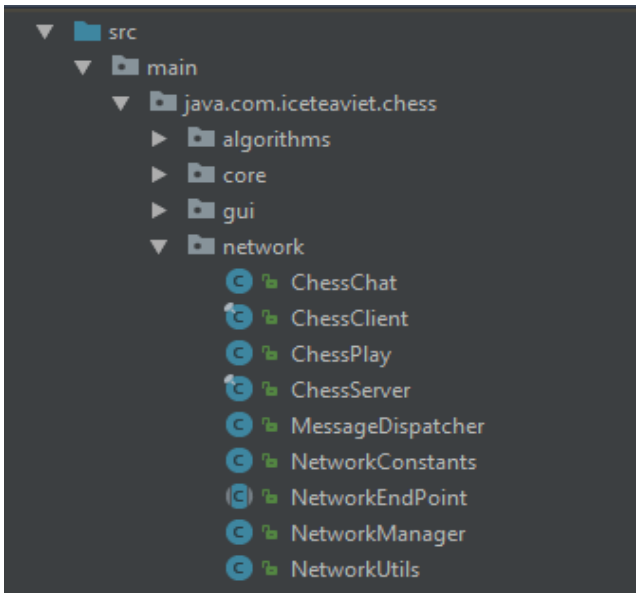


Đầu tiên là các bộ phân tích, ta có **ArthurBoardAnalyzer** kế thừa từ **GameBoardAnalyzer** dùng để duyệt qua thông tin vị trí của các quân cờ trên bàn cờ và tính điểm theo công thức.
Có các loại điểm như: **mobilityScore()** tính điểm linh động dựa vào khoảng cách giữa các quân cờ, **checkScore()** dựa vào khả năng một bên nào đó đang bị "chiếu tướng" mà tính điểm theo hằng số **CHECK_BONUS**, **checkmateScore()** tính khả năng bị "chiếu hết" của hai bên dựa vào độ sâu của phép search và hằng số **CHECK_MATE_BONUS**, **pieceValue()** tính số điểm của mỗi quân cờ dựa theo độ quan trọng và khả năng của quân cờ đó trong luật chơi, **attackScore()** tính số điểm tấn công dựa vào số quân cờ và giá trị của những quân cờ mà người chơi có thể tân công được, **pawnStructureScore()** tính điểm dựa vào vị trí của các quân tốt trên bàn cờ.
Từ những điểm số trên, ta nhân từng loại cho hệ số **MULTIPLIER** tùy theo độ quan trọng của loai điểm đó, sau đó cộng lại và ra quyết định nên tấn công hay phòng thủ.

Class **MoveAlgorithm.java** là một base class để cài đặt các thuật toán, ở đây ta có **MiniMax** và **AlphaBeta Pruning**.
MiniMax là một thuật toán đơn giản dựa vào các điểm số đã tính sau khi analyze bàn cờ mà tính được **min**, **max** và **đưa ra quyết định**, tuy nhiên thuật toán này phải duyệt nhiều lần và sẽ chậm đối với **search depth** lớn. Cắt tỉa AlphaBeta giải quyết được vấn đề đó, nó là sự cải tiến của MiniMax nhằm lọc bớt những nước đi thừa, giúp thuật toán search tìm ra nước đi tối ưu nhất chạy nhanh hơn mà vẫn đảm bảo kết quả đúng.

5. Xây dựng hệ thống network

Hệ thống network được xây dựng theo mô hình Client-Server với sự hỗ trợ của Java Socket kèm theo các thao tác với địa chỉ InetAddress và port number cơ bản.



Class **NetworkEndPoint** là một base class chứa những thành phần cơ bản của một kết nối đầu cuối như một biến **Socket** chứa kết nối đến người chơi còn lại, **BufferedWriter** và **BufferedReader** dùng để gửi/nhận message ra/vào OutputStream/InputStream, chứa biến **ChessPlay** dùng để giao tiếp network với game, và **ChessChat** dùng để giao tiếp với khung chat, chứa **OnNetworkUpdateListener** để cập nhật trạng thái của connection cho UI, cùng với các hằng số và giá trị mặc định.

ChessServer là một **thread** kế thừa từ **NetworkEndPoint** bao gồm một biến **ServerSocket** để giữ connection tới client, trong method **run()** của thread ta gọi hàm **waitingForConnection()** dùng để lắng nghe kết nối tới server, **getStreams()** dùng để lấy những Input/Output stream sau khi kết nối xong, cuối cùng sau khi khởi tạo thành công các stream thì khởi tạo **MessageDispatcher** để giao tiếp message giữa server và client.
ChessServer là một **thread** kế thừa từ **NetworkEndPoint** bao gồm một biến **Socket** để giữ connection tới server, trong method **run()** của thread ta gọi hàm **connectToServer()** dùng kết nối tới server theo thông tin **IP address** và **port** thu được từ giao diện NetworkDialog, **getStreams()** dùng để lấy những Input/Output stream sau khi đã kết nối xong, cuối cùng sau khi khởi tạo thành công các stream thì khởi tạo **MessageDispatcher** để giao tiếp message giữa server và client.

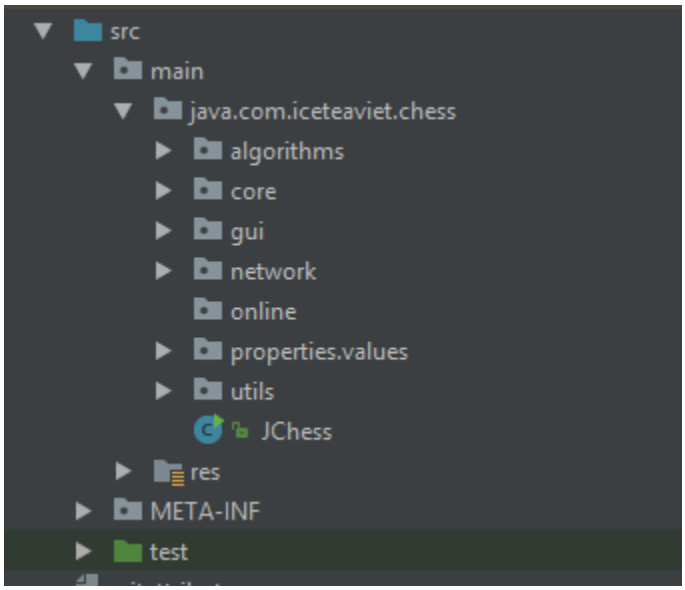
NetworkDispatcher là một thread với vai trò **trung tâm phân phối messages**, trong method **run()** của thread, chạy một vòng lặp **while(true)**, chờ đọc các message từ socket và dựa vào loại của message (phân biệt bằng prefix **MSG**: tượng trưng cho chat, **MOV**: tượng trưng cho move) để phân phối đến các thành phần cần thiết như **ChessChat**, **ChessGame** hay phát hiện các message **END_GAME** mà thoát vòng lặp, đóng kết nối.

ChessPlay và **ChessChat** dựa vào những message đã được phân phối đến, parse thông tin ra và gọi các hàm từ các class wrapup/manager của bộ phận đó mà xử lý.
NetworkUtils.java là một class dùng để giúp đỡ trong việc xác minh lại địa chỉ IP và port có hợp lệ hay không.

6. Refactoring code và tổ chức lại cấu trúc chương trình

Một chương trình tốt phải có được cấu trúc tốt và tổ chức vào các package hợp lý. Mặc dù cấu trúc chương trình đã được thiết kế sơ trước khi bắt tay vào làm, tuy nhiên trong quá trình làm game do thời gian giới hạn và nhiều người thực hiên nên cần refactor lại để sau này dễ fix bug và maintain.
Ở đây nhóm tuân thủ chặt chẽ pp **OOP** và chia source code chương trình ra thành từng phần riêng biệt như:

- **Core**: Bộ lõi Chess Engine
- **Gui**: Giao diện chương trình, xử lý các tương tác với người dùng
- **Algorithms**: Xử lý AI, thuật toán tìm nước đi tốt nhất.
- **Network**: Xử lý các thao tác chơi qua mạng
- **Online**: Xử lý server để deploy lên host online (Coming soon)
- **Properties**: Chứa các thông tin liên quan đến cài đặt và giá trị lưu trữ.
- **res**: chứa resource của chương trình



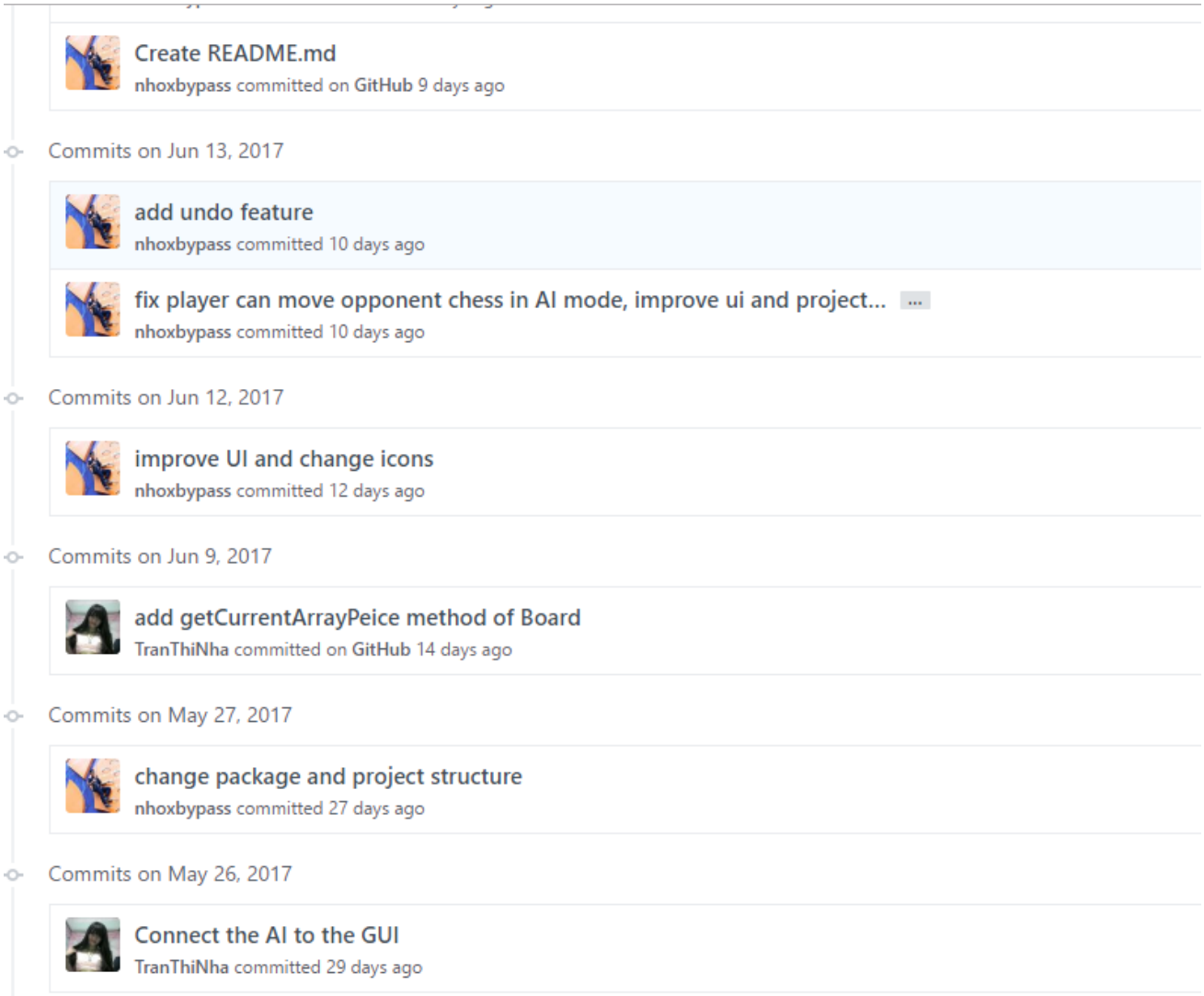
7. Testing

Trò chơi được viết unit test bằng **Junit** trong quá trình làm lõi chess engine.
Sau khi chương trình hoàn thành có những giai đoạn **Beta test** với một nhóm bạn chỉ là developer, và giai đoạn **Alpha test public** cho nhiều người chơi để lấy phản hồi lỗi và sửa chữa.

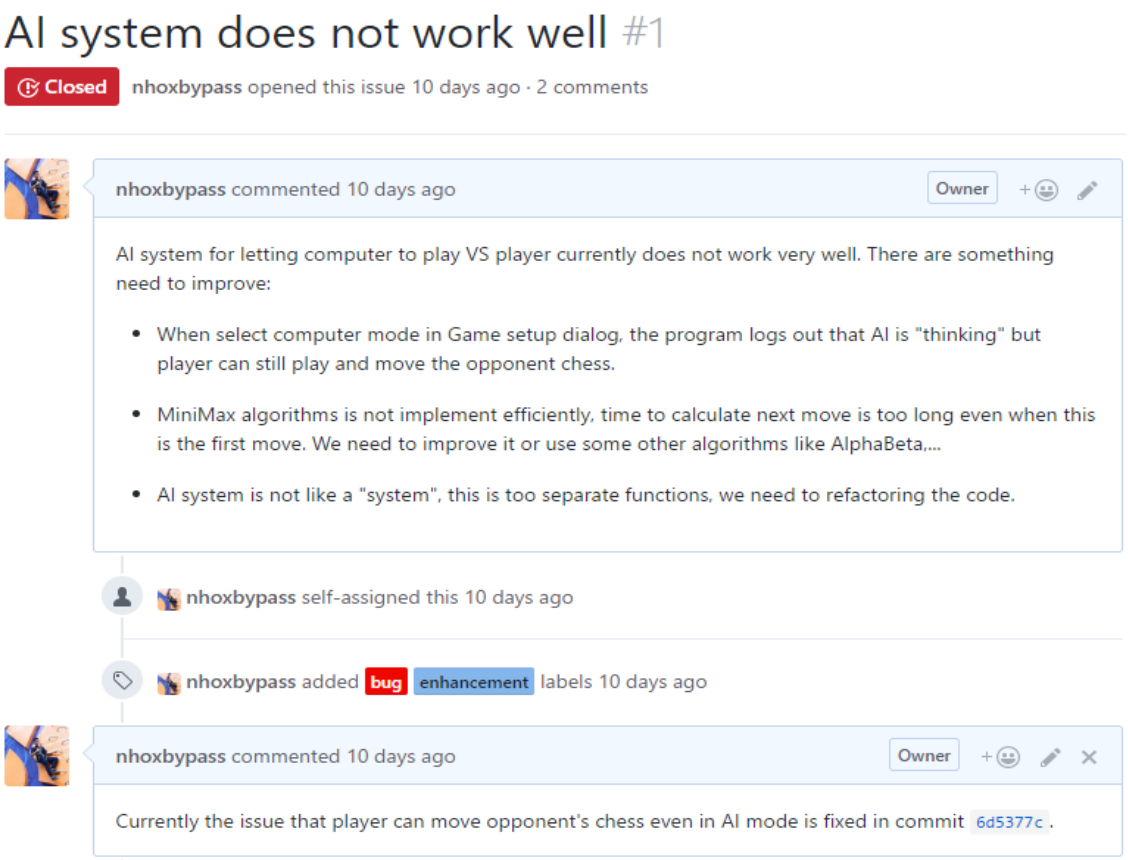
8. Công cụ hỗ trợ

Nhóm sử dụng Github để làm Source version control.

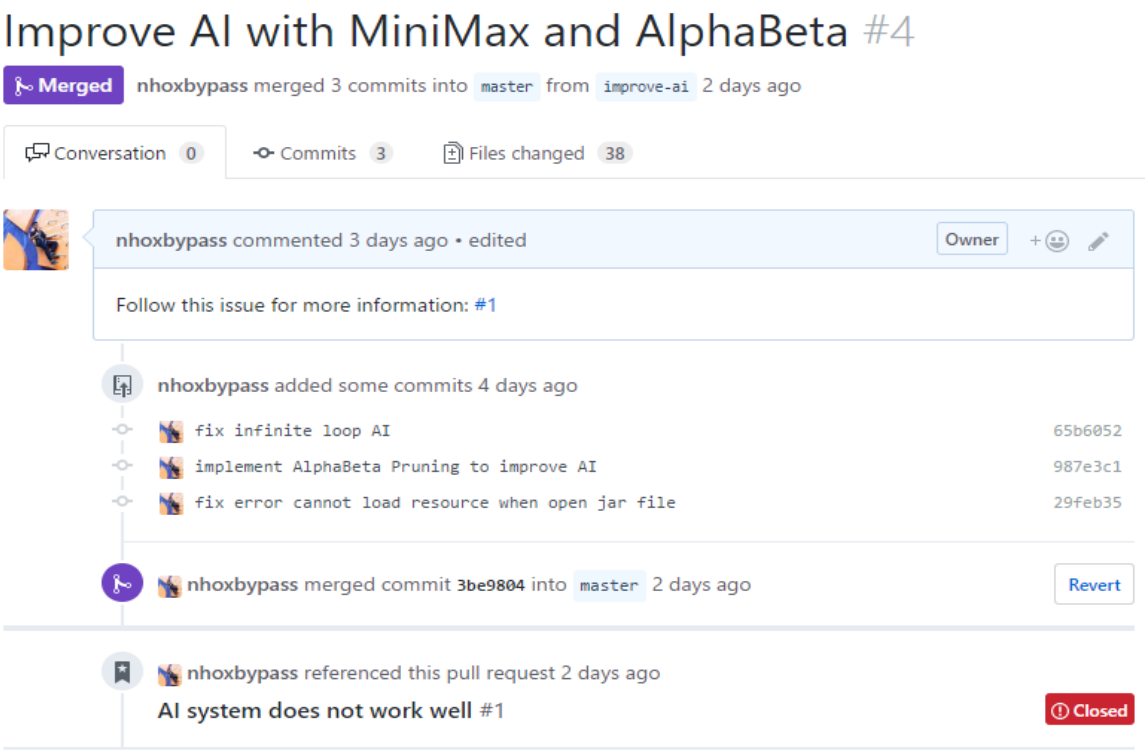
- **Thực hiện việc commit đều đặn**



- **Tạo issue khi gặp lỗi với đầy đủ requirement.**



- **Tạo thêm branch và pull request để cải thiện tính năng nào đó hoặc để fix lỗi.**



4 Báo cáo, đánh giá

Yêu cầu	Mô tả các kết quả đạt được	Bắt buộc	Tự đánh giá
Yêu cầu 1	Hỗ trợ đầy đủ tính năng và đúng luật của game cờ vua	✓	✓
Yêu cầu 2	Có giao diện đồ họa	✓	✓
Yêu cầu 3	Hỗ trợ tối đa 2 người chơi qua mạng LAN	✓	✓
Yêu cầu 4	Có chức năng cho máy đóng vai trò người chơi	✓	✓
Mở rộng 1	Giao diện material design		✓
Mở rộng 2	Có thể chat giữa hai người chơi		✓
Mở rộng 3	Gợi ý nước đi hợp lệ, cung cấp nhiều thuật toán cho AI		✓
	Tổng	10	10

5 Phân công công việc

MSSV	Họ tên	Công việc	Mức độ hoàn thành
1412197	Đoàn Thị Phương Huyền	Thiết kế giao diện + Tìm hiểu AI	100%
1412363	Trần Thị Nhã	Phát triển game + áp dụng AI	100%
1412477	Đoàn Hiếu Tâm	Phát triển chức năng nhiều người chơi qua mạng + Chỉnh sửa giao diện và tổng hợp	100%