The University of Hong Kong

Faculty of Engineering

Department of Computer Science

Final Year Project – Final Report

Chess Making Engine

A tool to quickly generate and modify a chess-game based Java source code with GUI implementation.

Supervisor

Dr. K. P. Chan

Group mates

Fung Wai Him (3035074971)

Li Chun Kit (3035074311)

Date

17th April, 2016

Simplified Table of Content:

Chapter	Content	Page
1	Summary	3
2	Background and Introduction	4
3	Objective and Target	10
4	Overview of Setup and System Requirement	12
5	Methodology and System Design	13
	Part 1: Chess Making Engine - Chreator	18
	Part 2: Chess Game Executable	52
6	Phase work, Result and Deliverables	62
7	Difficulties, Reflection and Improvement	64
8	Division of Labor	65
9	Conclusion	66

Chapter 1: Summery

This article is introducing a brand new tool designed for making chess gaming. This tool generates pre-written code of chess game executable GUI representation and interaction, general game settings and structure of the game system which controls the chess board and chess piece. These features are hopefully able to speed up the development of chess game as user can then focus on the implementation of AI algorithm. The game system structure is designed to be easily extended and modified.

The project is divided into 2 parts as mentioned above. The first part is the implementation of the chess game making tool called Chreator. Chreator is used to generate source code of chess game executable. The second part is the implementation of the chess game executable. All the details about the 2 software will be described in the article.

Chapter 2: Background and Introduction

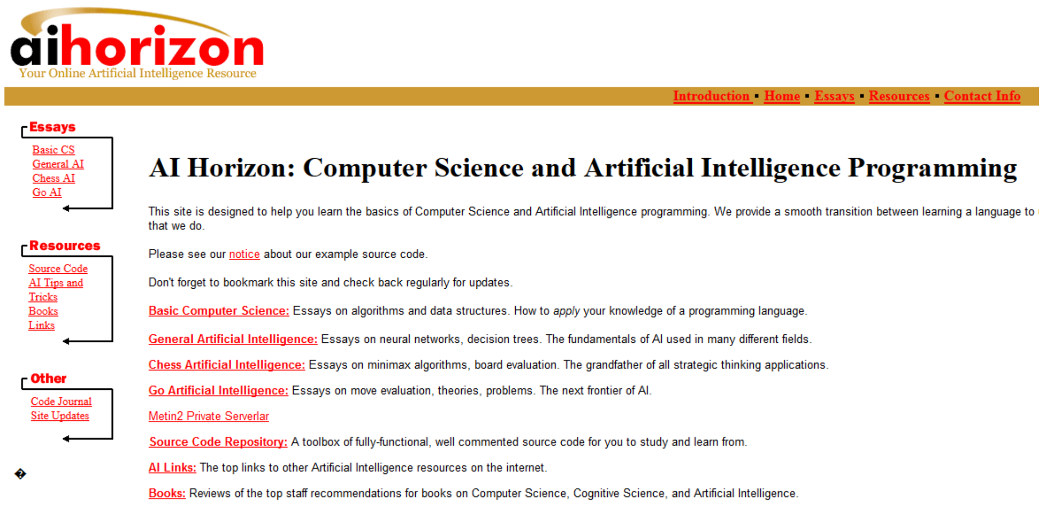
2.1 Background

Artificial intelligent (AI) is being more and more important and popular nowadays. Computer games acts as a very important character for entertainment and productivity enhancement in our daily life. AI in gaming can act as an enemy such that players can play against with it; it can also be used as utility tool for result prediction, network advertisement marketing, situation virtualization and decision making. Recently there is a very famous AI software developed by Google called AlphaGo that successfully beat the Korean 9-dan (九段) professional Go (圍棋) player with result 4:1[1]. Such a shocking result successfully raised lots of noise and discussion about the power of AI and its impacts in all different aspect of people, such as netizen, IT companies and computer scientists. We can see that studies on AI algorithm will develop much further more. As a result, learning and improving the implementation of AI algorithm becomes a topic being concerned.

2.2 Introduction

Since there are more beginners to start learning the AI implementation, there are many materials found on the internet. The most typical learning aspect to start with is chess game AI implementation. There exists simple chess game making tutorials, chess game software and game making software available online. The chess making tutorials provide step by step instructions of code implementation for building the whole chess game including main window, chess board, chess piece, AI, statistic information display frame and so on. Here is the example of aiHorizon[2].

[1] AlphaGo – Wikipedia: https://en.wikipedia.org/wiki/AlphaGo
[2] Aihorizon: http://www.aihorizon.com/


aihorizon
Your Online Artificial Intelligence Resource

Introduction ▪ Home ▪ Essays ▪ Resources ▪ Contact Info

Essays
Basic CS
General AI
Chess AI
Go AI

Resources
Source Code
AI Tips and Tricks
Books
Links

Other
Code Journal
Site Updates

AI Horizon: Computer Science and Artificial Intelligence Programming

This site is designed to help you learn the basics of Computer Science and Artificial Intelligence programming. We provide a smooth transition between learning a language to that we do.

Please see our notice about our example source code.

Don't forget to bookmark this site and check back regularly for updates.

Basic Computer Science: Essays on algorithms and data structures. How to *apply* your knowledge of a programming language.

General Artificial Intelligence: Essays on neural networks, decision trees. The fundamentals of AI used in many different fields.

Chess Artificial Intelligence: Essays on minimax algorithms, board evaluation. The grandfather of all strategic thinking applications.

Go Artificial Intelligence: Essays on move evaluation, theories, problems. The next frontier of AI.

Metin2 Private Serverlar

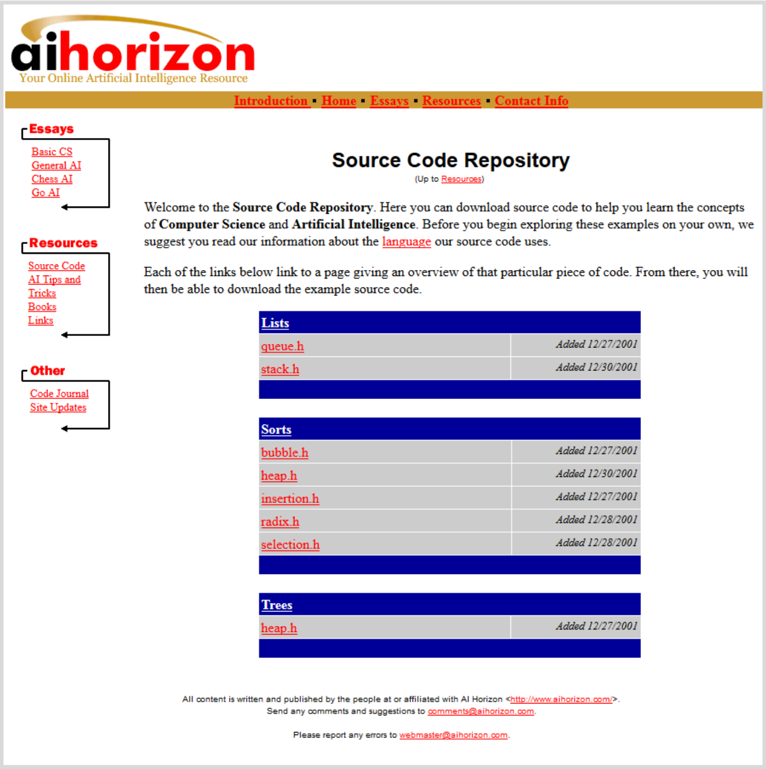
Source Code Repository: A toolbox of fully-functional, well commented source code for you to study and learn from.

AI Links: The top links to other Artificial Intelligence resources on the internet.

Books: Reviews of the top staff recommendations for books on Computer Science, Cognitive Science, and Artificial Intelligence.

Fig. 1a: Home page of aihorizon (http://www.aihorizon.com/)

Essays about some types of AI are found. For example the AI implementation methods and tips about western Chess, Go (圍棋) and so on. You may also find source codes of the topic, below is the source code page of the aihorzon:


aihorizon
Your Online Artificial Intelligence Resource

Introduction ▪ Home ▪ Essays ▪ Resources ▪ Contact Info

Essays
Basic CS
General AI
Chess AI
Go AI

Resources
Source Code
AI Tips and Tricks
Books
Links

Other
Code Journal
Site Updates

Source Code Repository
(Up to Resources)

Welcome to the **Source Code Repository**. Here you can download source code to help you learn the concepts of **Computer Science** and **Artificial Intelligence**. Before you begin exploring these examples on your own, we suggest you read our information about the language our source code uses.

Each of the links below link to a page giving an overview of that particular piece of code. From there, you will then be able to download the example source code.

Lists	
queue.h	*Added 12/27/2001*
stack.h	*Added 12/30/2001*

Sorts	
bubble.h	*Added 12/27/2001*
heap.h	*Added 12/30/2001*
insertion.h	*Added 12/27/2001*
radix.h	*Added 12/28/2001*
selection.h	*Added 12/28/2001*

Trees	
heap.h	*Added 12/27/2001*

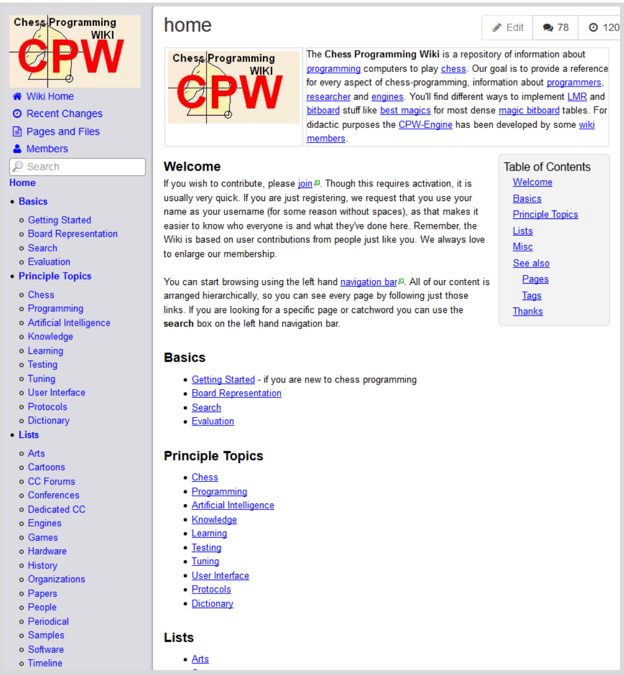
All content is written and published by the people at or affiliated with AI Horizon <http://www.aihorizon.com/>.
Send any comments and suggestions to comments@aihorizon.com.

Please report any errors to webmaster@aihorizon.com.

Fig. 1b: source code download page of aihorizion

We can see that the site provides source code header files of the AI implementation such as heap, bubble search, stack and so on. However, they are just part of the program. Users have to find other libraries to write the GUI and combine all the libraries downloaded from this site to from the complete program.

The other site about chess game making called “Chess Programming Wiki”[3].



Chess Programming WIKI CPW

Wiki Home
Recent Changes
Pages and Files
Members
Search

Home

- **Basics**
  - Getting Started
  - Board Representation
  - Search
  - Evaluation
- **Principle Topics**
  - Chess
  - Programming
  - Artificial Intelligence
  - Knowledge
  - Learning
  - Testing
  - Tuning
  - User Interface
  - Protocols
  - Dictionary
- **Lists**
  - Arts
  - Cartoons
  - CC Forums
  - Conferences
  - Dedicated CC
  - Engines
  - Games
  - Hardware
  - History
  - Organizations
  - Papers
  - People
  - Periodical
  - Samples
  - Software
  - Timeline

home

Edit 78 120

The **Chess Programming Wiki** is a repository of information about programming computers to play chess. Our goal is to provide a reference for every aspect of chess-programming, information about programmers, researcher and engines. You'll find different ways to implement LMR and bitboard stuff like best magics for most dense magic bitboard tables. For didactic purposes the CPW-Engine has been developed by some wiki members.

Welcome

If you wish to contribute, please join. Though this requires activation, it is usually very quick. If you are just registering, we request that you use your name as your username (for some reason without spaces), as that makes it easier to know who everyone is and what they've done here. Remember, the Wiki is based on user contributions from people just like you. We always love to enlarge our membership.

You can start browsing using the left hand navigation bar. All of our content is arranged hierarchically, so you can see every page by following just those links. If you are looking for a specific page or catchword you can use the **search** box on the left hand navigation bar.

Table of Contents
- Welcome
- Basics
- Principle Topics
- Lists
- Misc
- See also
  - Pages
  - Tags
- Thanks

Basics

- Getting Started - if you are new to chess programming
- Board Representation
- Search
- Evaluation

Principle Topics

- Chess
- Programming
- Artificial Intelligence
- Knowledge
- Learning
- Testing
- Tuning
- User Interface
- Protocols
- Dictionary

Lists

- Arts

Fig. 1c: Home page of Chess Programming Wiki

The site provides much abundant resources about different AI algorithms, implementation of different type of Chesses and their variants, methods of GUI implementation and method of implementation such that the chess game executable product can be run on different platform such as Mac OS, Linux, Windows or even Android.

However, users have to collect all the libraries they need from the site and write the extra code to combine the libraries to from the final workable game.

We can conclude that the chess game making tutorials and essays find on sites provide most of the core libraries to implement any type chess game that covering the AI, GUI and system structure aspects. Users have to implement lots of extra code to combine the libraries to form the final product. We can see that users may need to spend quite a lot of time on coding the GUI and the structure of the game system, but spend less focusing on implementing the AI algorithm and the game rules (including the chess piece moving rules and end game criteria).

To get rid of spending lots of time on implementing the GUI and system structure instead of AI and game rules, we may consider the other type of learning material – existing chess game software. XQwizard[4] is a typical example:

[3] Chess Programming Wiki: http://chessprogramming.wikispaces.com/
[4] XQWizard: https://www.xqbase.com/xqwizard/xqwizard.htm

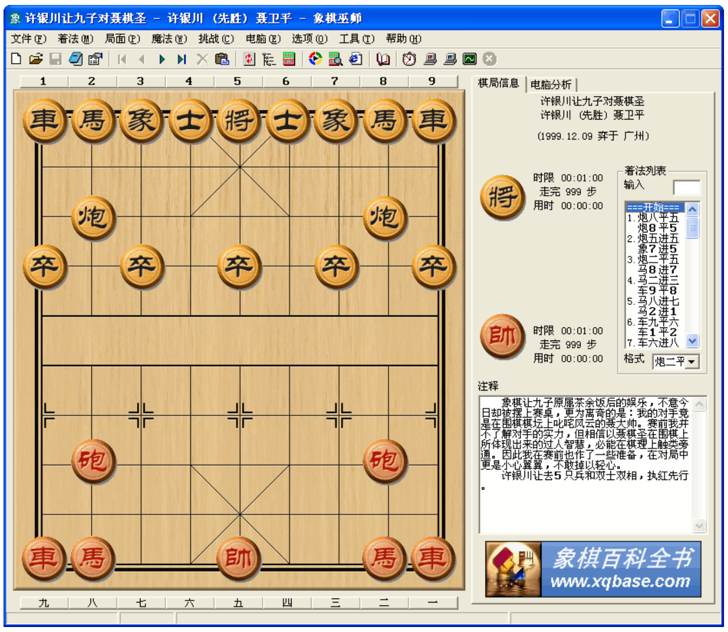


Fig. 1d, screen shot of XQwizard from the official site

This kind of chess game playing software contains the fully implemented GUI, game system structure, AI and all the game rules. The game can be executed without type one single line of code. However, the source code of the game may not be released. As a result the user may not be able to modify the game rules, the type of playing chess and AI algorithm. Thus it is not suitable for AI learning as we cannot view and modify the AI algorithm.

The third one is the specialized game making software – Unity[5]:

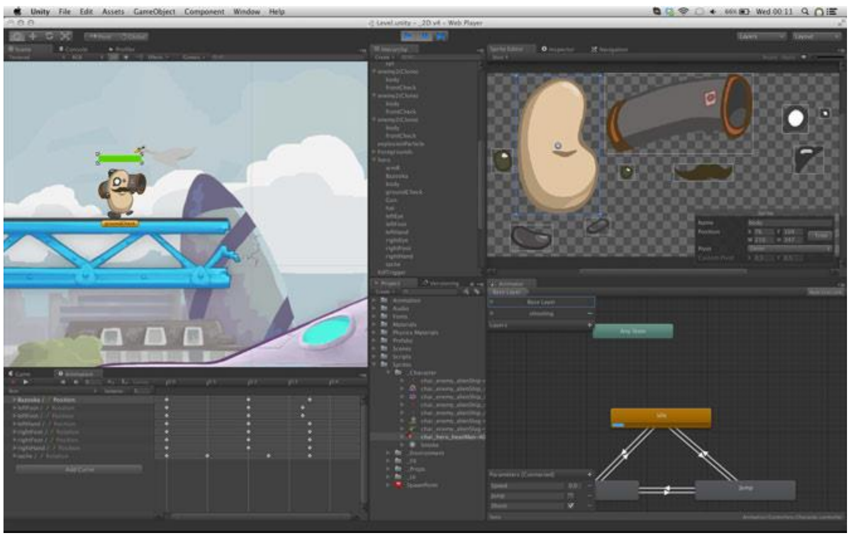


Fig 1e: screen shot of Unity (http://blogs.unity3d.com/2013/08/28/unity-native-2d-tools/)

[5] Unity: http://unity3d.com/cn/

Unity is a tool to make any 3D and 2D games. The tool is highly object-oriented and programmable. User can set most of the static features such as size, position and so on in the Inspector. Only the advanced logic and action of the game objects need to be coded in C# or JavaScript language. The user can speed up their chess game implementation process if they use this kind of tool. However Unity is not a tool specialized for making chess game, there may be still lots of features that need to be specified, such as the game structure that connects the entire chess piece and chess board all together. The other problem is that the game world built by Unity is a 3D space. The GUI rendering algorithm may be too advanced for a simple chess game. Thus, quite a part of the computation resource is used for rendering. This slows down the AI calculation process.

To sum up the chess making tools discussed above, chess making tutorials cover any kind of the chess game. It is also flexible on modifying the code. The complexity of the game is also high that there is no 3D world or 2D world rendering problem. However users need to spend great amount of time on combining the libraries by their own coding implementation. The chess game playing software supports single of just few types of chess gaming. The AI, game rules and the game system are usually not able to be modified unless the source code of the game is released. The advantage of chess game playing software is that it is directly downloaded and played. The game making tool like Unity is close to what we want. GUI and game system is implemented by the tool already. Only the game structure such as the chess board and chess piece, and the AI algorithm are needed to be implemented by the user himself. However the computation resource may be spent on rendering the 2D or 3D world. The game made by this tool may not be efficient enough to handle the highly complex AI algorithm.

In order to fulfill the topic on chess game making tool for AI learning, we may need to combine all the advantages of chess game making tutorials, chess game playing software and general game making software, we introduce our own chess making engine – Chreator and the chess game executable produced by Chreator. The objective and target of our project immediately introduced in the next chapter.

Chapter 3: Objective and Target

By the reference of the general game making tool Unity, our target solution called Chreator is a tool for creating an executable chess game that allows users to implement the features of the chess piece by GUI and coding for AI, game rules and chess piece behaviors. Chreator ideally has the following target and features:

1. It provides user-friendly UI for quick setup of chess game project or basic settings modification;
2. It is able to support multiple types of fully observable & deterministic chess game, such as Chess (西洋棋), XiangQi (中國象棋), Go (圍棋) and so on;
3. It is highly flexible and modifiable which allows users to set up custom data types on board, of pieces and in AI implementation; and
4. It is able to serve as a plain chess playing engine for testing or entertainment.

We hope to help users to either focus on implementing their AI algorithm by using our templates and API provided, or inventing their custom chess using the clean UI. At the same time, one could save, open, clone and compile a chess game project to a chess game executable easily which makes any kind of development simpler.

A chess game executable compiled by the Chreator is basically similar to a normal simple chess game existing in the market. Information will be provided for user to understand their implementation. The chess game executable is ideally expected to provide:

1. Human to AI chess gaming;
2. Game flow control, such as undo; export game stage to a file; import and resume a saved game stage;

3. Information frame showing system resources consumed for one round of calculation;
4. Debug mode: executing AI and game tree line by line to see how the decision is made.

The latter two implementations would help a lot on AI learning and implementation. However, it is reminded that the final products do not achieve all the target functions proposed here. Only the basic functions of both the software are implemented due to time limitation and technical difficulties. All the functions mentioned above can actually be implemented in the future develop. The more details will be reported in the later chapters.

Chapter 4: Overview of Setup and System Requirement

Both the Chreator and the source code of the chess game executable are written in Java language. Since Java is a cross platform language, our product can be run successfully on Windows, Mac OS and Linux platform. Despite rewriting the UI handling module of the product, the remaining libraries can even be able to be executed on Android as Java is the middle-man Android application developing language.

To run the Chreator and the chess game executable, is it recommended that a personal computer with computation power be higher than or equal to Intel core i5 processor, 4GB main memory or above is used.

For the software part, running the product requires a personal computer with 64-bit PC operating system such as Windows, Mac OS or Linux. Since the user has to compile the source code of the chess game executable and run the Chreator and the chess game executable, both the Java Development Kit (JDK) and the Java Runtime Environment (JRE) are needed to be installed. Again in order to let the operating system to allocate more main memory to the Java running programs, the latest 64 bit version of JDK and JRE are required. Normally, the JDK install package downloaded from the official Oracle Java website already contains the JRE inside.

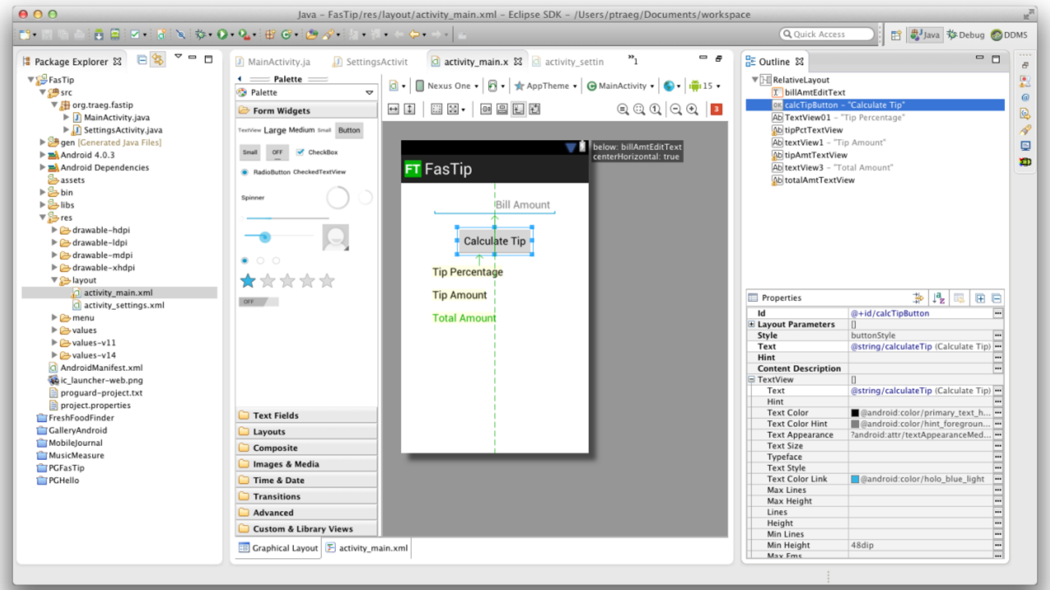
Though the Chess making engine Chreator provides the code editing function, but it is sure that the code editing function is not as professional as the Java Development IDE tools. Therefore, it is recommended that Eclipse IDE for Java is used to modify the source code of the chess game executable for advanced mean. To modify the Chreator and the chess game executable so that they can be run on the Android devices in optional mean, Android Studio is needed.

Chapter 5: Methodology and System Design

Comparison of methods to the solution

From the previous chapters, the solution of developing the chess game making engine Chreator is briefly introduced, however, there are several approaches for solving the problem that have been discussed, such as providing command set similar to bash shell script, for user to implement the whole chess game. However those methods still do not solve the fundamental problem – providing ways to let user quickly create chess piece, chess board and game rules with the help of the software such that AI implementation can be concerned. It is because the users have to learn a complete new programming language (the command invented by us) and have to code for part of the UI of the game, especially the main UI of the chess game, chess board and chess piece.

We also think of the other approach that developing a plug-in application for the Eclipse Java development IDE tool. The plugin does the very similar functions to the Chreator mentioned in the objective, that is – when the user start a new project in the Eclipse IDE for Java, the user can choose to build or open a chess game executable source code project. Besides the normal coding area in Eclipse IDE, the plug-in also provides a GUI for defining the GUI of chess board and chess piece, just like the Android plug-in in Eclipse, that enable drag and drop to edit the layout of an Android application and preview the layout of the Android application (What You See Is What You Get approach - WYSIWYG). Figure 5.0a below is an example of the Android plugin running in Eclipse IDE that editing the layout of an Android application without actually typing code for defining the layout:



Id	@+id/calcTipButton
Layout Parameters	[]
Style	buttonStyle
Text	@string/calculateTip (Calculate Tip)
Hint	
Content Description	
TextView	[]
Text	@string/calculateTip (Calculate Tip)
Hint	
Text Color	■ @android:color/primary_text_h...
Text Color Hint	■ @android:color/hint_foregroun...
Text Appearance	?android:attr/textAppearanceMed...
Text Size	
Typeface	
Text Style	
Text Color Link	🟦 @android:color/holo_blue_light
Max Lines	
Max Height	
Lines	
Height	
Min Lines	
Min Height	48dip

Fig 5.0a: Android plugin running in Eclipse IDE, providing a GUI layout for editing the layout of the Android application by using drag and drop WYSIWYG approach.[6]

Although this approach helps us to save the time for implementing the coding IDE and the mechanism of this approach is similar to what is doing in our final product - the Chreator, we have to study coding for a plug-in tool in the Eclipse IDE. Moreover, the user may be forced to use Eclipse IDE to code for Java program. Though Eclipse IDE is cross-platform and popular software, this method at the end is considered as not that user friendly enough. We finally proposed the solution mentioned in the objective – developing an independent Java Chess Making Engine called Chreator and the Chess Game Executable.

<u>Introduction of final method chosen to the solution</u>

Summering the approaches concerned, we finally come up with the solution that making a chess making tool called Chreator, it generates source code of a chess game executable with simple work done to the definition to the chess board and chess

[6] Peter Traeg – Four Ways To Build A Mobile Application, Part 2: Native Android. Site: https://www.smashingmagazine.com/2014/01/four-ways-to-build-a-mobile-app-part2-native-android/

pieces of a chess game. Below is the simplified use case model and brief introduction about the relationship and general logic flow of the Chreator and the chess game executable.

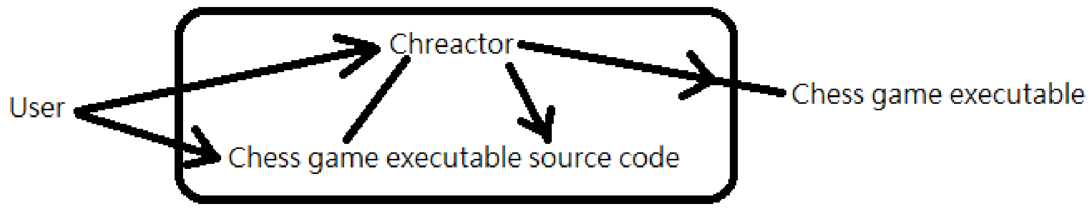


Fig 5.0b Arrowed links are the actions to be done to the target. Non-arrowed link is the dependency

The user first launches the Chreator – which is the middleman for managing the project files and the content of the source code files of the chess game executable. Then the user asks the middle man to generate the source code of the chess game executable. In the Chreator, the user has to setup the chess piece placing points in the chess board; size and image of the chess board; size, image and the initial placing position of the chess piece; Java code for the moving rule of the chess piece, game rules and AI implementation. After those data are provided and other appropriate modifications are done to the source code, the middleman Chreator submits the code to JDK and request for compilation of chess game executable. Any compilation errors to the source code and the execution output from the chess game executable will be received by the Chreator and report to the user. The running chess game executable in this stage shows the chess board UI and the initial layout of the chess piece placed on the chess board. Selecting a chess piece on the chess board shows its available moves.

<u>Special structure of the chess game executable</u>

The chess game executable is just a normal game. The chess pieces and chess board are the basic components of the game. The general game flow is as flow:

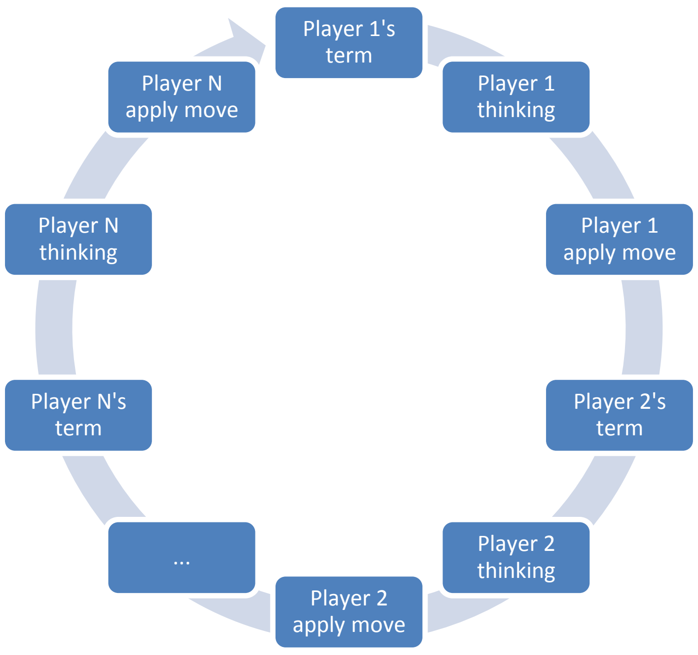


Fig. 5a general chess game game flow

The loop only breaks when some game rules are satisfied. For example, if one or all players have no next move.

There is a concern about the data structure of the chess board. It is known that most of the chess boards are rectangular, a 2D array is suitable for representing the structure of a chess board. However, there are exceptions. Consider the following triangular chess board of Chinese checkers (中國跳棋).

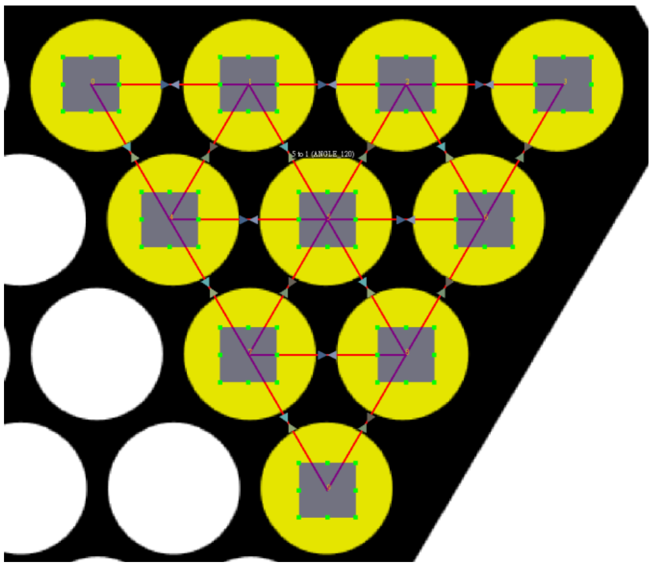


Fig. 5b: A corner of the chess board of Chinese checkers (中國跳棋)

Here, a 2D array does not hold the representation of the chess board of the Chinese checkers anymore. When need a new system to store the chess board. Here we introduce the concept of ***chess piece placing points, point edge*** and ***point orientation*** (or ***point direction*** you may say). This concept is similar to the linked-list.

The center chess piece placing point of a Chinese checkers chess board shown on the figure 5b has 6 neighbor points, there direction are ANGLE_0, ANGLE_60, ANGLE_120, ANGLE_180, ANGLE_240 and ANGLE_300 respectively.

In order to recognize each piece placing point on the chess board, a unique point ID is given to each point and each piece. A piece placing point can hold a chess piece. The chess piece should also know its own representing player side.

In the following sub-parts, the system architecture, design of each component, special methods or algorithms adopted and user using approach are introduced in detail.

Part 1: Chess Making Engine - Chreator

<u>System Architecture of the entire Chreator</u>

This part is introducing the system architecture of the Chreator. Please download and extract the Chreator deliverable of our project "Chreator.jar", the following structure of the product is shown.

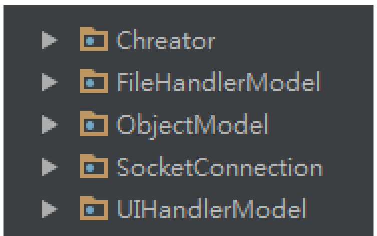

```
▶ Chreator
▶ FileHandlerModel
▶ ObjectModel
▶ SocketConnection
▶ UIHandlerModel
```

Fig. 5.1a Root hierarchy of the Chreator

Figure 5.1a shows the root hierarchy of the Chreator. There are 5 folders in the root hierarchy, only the Chreator folder contains the main program of the Chreator, the other 4 folders contains the static source file dependency for compiling the chess game executable. Missing the 4 folders though can still make the Chreator run successfully, but the generated chess game executable will fail to run.

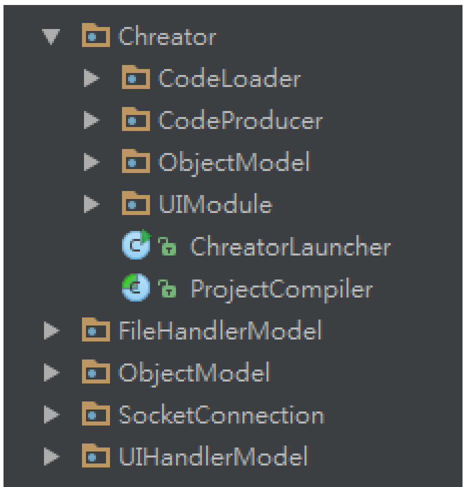

```
▼ Chreator
    ▶ CodeLoader
    ▶ CodeProducer
    ▶ ObjectModel
    ▶ UIModule
      ChreatorLauncher
      ProjectCompiler
▶ FileHandlerModel
▶ ObjectModel
▶ SocketConnection
▶ UIHandlerModel
```

Fig. 5.1b Chreator hierarchy – the main program of the Chreator.

The main program of the Chreator is divided into 6 parts, the "CodeLoader" module, "CodeProducer" module, "ObjectModel" moduel, "UIModuel" module, "ChreatorLauncher" Java class and the "ProjectCompiler" Java class. Functions of each module and Java class is explained as follow:

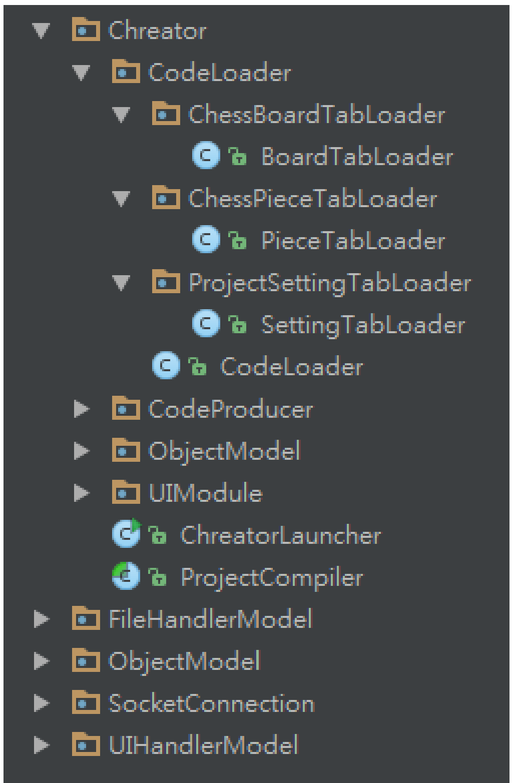
- The "CodeLoader" module loads the source code of the chess game executable, and then tries to grab the data in the source code and apply to the Chreator.
- The "CodeProducer" convert the project data in Chreator to the source code of

the chess game executable

- The “ObjectModel” is the definition for the template concept objects of the chess game, this will be explained latter.
- The “UIModule” controls, renders and handles the GUI of the Chreator.
- The “ChreatorLauncher” contains the main function of the Chreator, it starts the Chreator and connects all the modules together, you may consider it is a bridge of all the Modules.

The “ProjectCompiler” Java class is responsible for compiling the source code of the chess game executable, report compilation errors, launching the chess game executable and receiving messages from the chess game executable.

CodeLoader



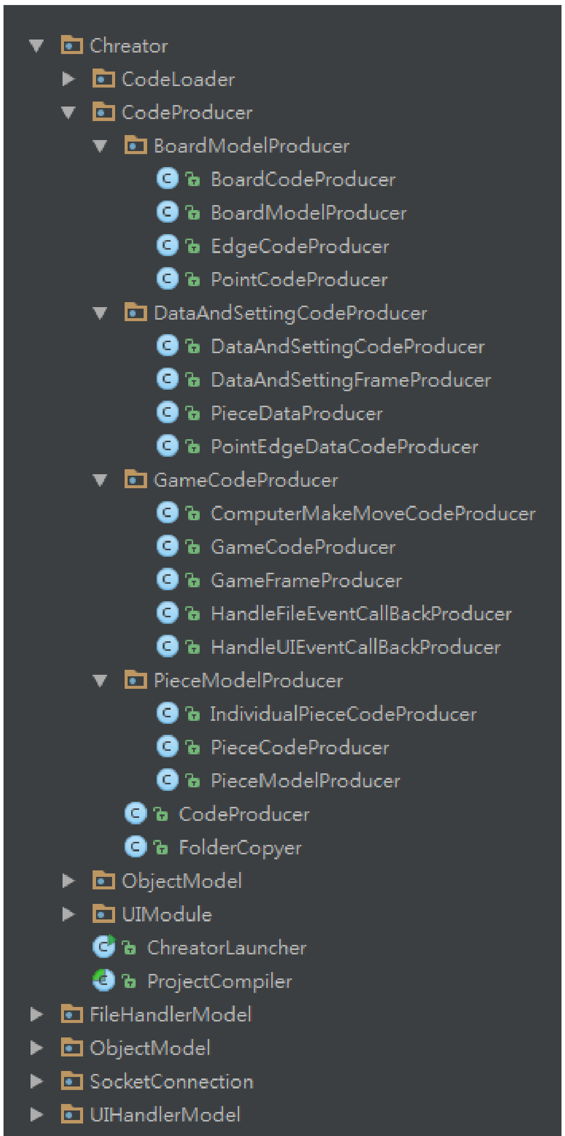
```
▼ Chreator
    ▼ CodeLoader
        ▼ ChessBoardTabLoader
            BoardTabLoader
        ▼ ChessPieceTabLoader
            PieceTabLoader
        ▼ ProjectSettingTabLoader
            SettingTabLoader
          CodeLoader
    ▶ CodeProducer
    ▶ ObjectModel
    ▶ UIModule
      ChreatorLauncher
      ProjectCompiler
▶ FileHandlerModel
▶ ObjectModel
▶ SocketConnection
▶ UIHandlerModel
```

Fig 5.1c CodeLoader Module of the Chreator

Expending the “CodeLoader”, there are 4 Java classes, the CodeLoader, BoardTabLoader, PieceTabLoader and the SettingTabLoader. The “Tab Loaders” are covered by a parent folder. By the meaning of the Java classes, the CodeLoader controls the 3 TabLoaders. BoardTabLoader, PieceTabLoader and the SettingTabLoader collects the data from the source code of the chess game executable and apply to the Board tab, Piece tab and General Game rule tab in the Chreator.

Please be reminded that this part is implemented by my partner Li Chun Kit. For more details about the implementation, please refer to Kit’s report.

CodeProducer



```
▼ Chreator
    ▶ CodeLoader
    ▼ CodeProducer
        ▼ BoardModelProducer
              BoardCodeProducer
              BoardModelProducer
              EdgeCodeProducer
              PointCodeProducer
        ▼ DataAndSettingCodeProducer
              DataAndSettingCodeProducer
              DataAndSettingFrameProducer
              PieceDataProducer
              PointEdgeDataCodeProducer
        ▼ GameCodeProducer
              ComputerMakeMoveCodeProducer
              GameCodeProducer
              GameFrameProducer
              HandleFileEventCallBackProducer
              HandleUIEventCallBackProducer
        ▼ PieceModelProducer
              IndividualPieceCodeProducer
              PieceCodeProducer
              PieceModelProducer
          CodeProducer
          FolderCopyer
    ▶ ObjectModel
    ▶ UIModule
      ChreatorLauncher
      ProjectCompiler
▶ FileHandlerModel
▶ ObjectModel
▶ SocketConnection
▶ UIHandlerModel
```

Fig 5.1d CodeProducer Module of the Chreator

The “CodeProducer” Module is responsible to covert the data about the chess board, chess piece and game settings in Chreator to the source code of the chess game executable. There are 4 sub-folder modules, a CodeProducer class and a FolderCopyer class. Similar to the CodeLoader, the CodeProducer class controls all the other classes under the 4 sub-folder modules and the FolderCopyer class. The FolderCopyer class is responsible to copy the static file dependency of the chess game executable: they are the FileHandlerModel, ObjectModel, SocketConnection and UIHandlerModel described in Figure 5.1a. The four sub-folder modules produce code for specific function of the chess game executable.

Classes under BoardModelProducer cooperate together to produce source code about the chess board image, chess board piece placing points, edges and their directions between the piece placing points and so on. Classes under the

DataAndSettingCodeProducer collect data in the Chreator and coopreate together to form the DataAndSetting class of the chess game Executable. This class defines the setting of the standard chess piece, including the size, piece image, initial location and other general settings to the game.

Classes under GameCodeProducer produce the code for connecting the GUI and the game system of the chess game executable, such as the chess piece moving mechanism, handler of the dynamic event callback from the GUI and so on.

Classes under the PieceModelProducer produce the detail codes about a chess piece, especially the moving rules of each kind of piece.

Please be reminded that the whole CodeProducer module is implemented by Li Chun Kit. For more details about the implementation, please refer to Kit's report.

ObjectModel

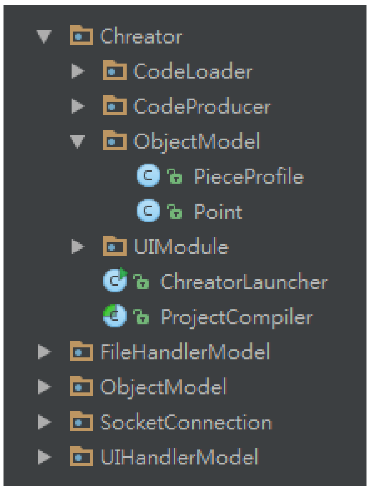
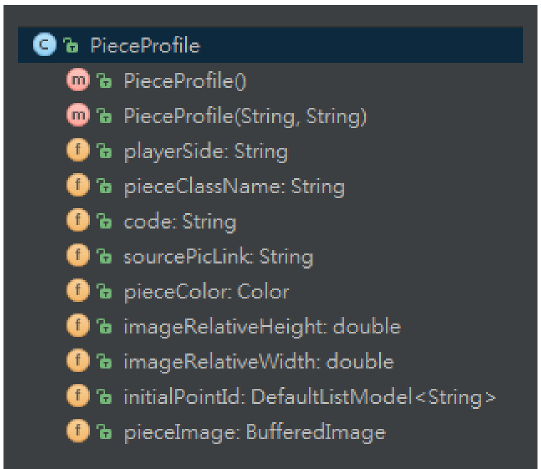


Fig. 5.1e.0 ObjectModel of Chreator

The whole ObjectModel is implemented by me. It aims to provide virtual concept about the chess game and the communication packet between classes such that data collected from different part of the Chreator are be synchronized.

ObjectModel / PieceProfile



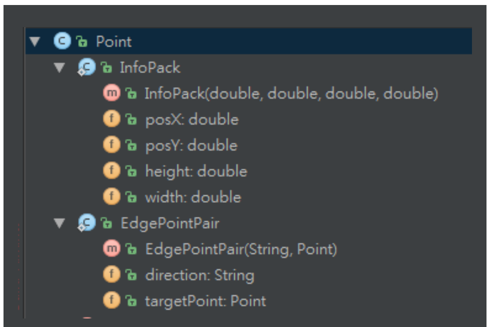
```
PieceProfile
    PieceProfile()
    PieceProfile(String, String)
    playerSide: String
    pieceClassName: String
    code: String
    sourcePicLink: String
    pieceColor: Color
    imageRelativeHeight: double
    imageRelativeWidth: double
    initialPointId: DefaultListModel<String>
    pieceImage: BufferedImage
```

Fig. 5.1e.1 of Chreator

Figure 5.1e.1 shows the structure of the PieceProfile class under the ObjectModel. PieceProfile in the Chreator is used to pack up all the information describing a chess piece to this piece profile. The PieceProfile stores the information about:

- playerSide: the player side
- pieceClassName: piece name
- code: code of the piece class in the source code of the chess game executable
- sourcePicLink: picture link of the chess piece
- pieceColor: the piece color if the user does not provide the picture of the chess piece.
- imageRelativeHeight: the relative height of the chess piece. The range is between 0.0 to 1.0, where 1.0 means that the height of the chess piece is as same to the height of the chess board.
- imageRelativeWidth: the relative width of the chess piece. The range is between 0.0 to 1.0, where 1.0 means that the width of the chess piece is as same to the width of the chess board.
- initialPointId: a list stores all the ID of the piece placing point that the chess piece should be placed to when the game starts.
- pieceImage: a buffered image of the chess piece.

ObjectModel / Point



```
▼ Point
  ▼ InfoPack
      InfoPack(double, double, double, double)
      posX: double
      posY: double
      height: double
      width: double
  ▼ EdgePointPair
      EdgePointPair(String, Point)
      direction: String
      targetPoint: Point
```

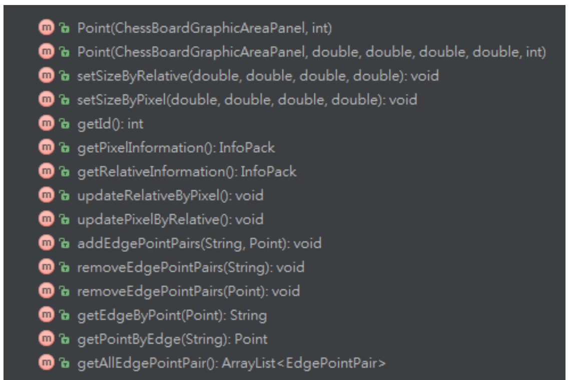
Fig. 5.1e.2.1 Sub-class inside the PointClass

The point class is used to represent the piece placing point and the chess piece in the Chreator. Since the window of the chess game may resize from time to time, but the ratio of the chess board and the chess piece should the same.

Under this concept, if all the chess pieces are represented by relative size that 0.0 x 0.0 is the size of 0 x 0 pixel, 1.0 x 1.0 is the size same to the chess board. However, since the application of editing chess piece size in the Chreator relies on pixel size, the InfoPack class shown in figure 5.1e.2.1 supports both the representation of pixel and relative of size and coordinates. The attributes of the InfoPack class is as follow:

- posX: the pixel or relative x coordinate of the center of the point
- posY: the pixel or relative y coordinate of the center of the point
- height: the pixel or relative height of the point
- width: the pixel or relative width of the point

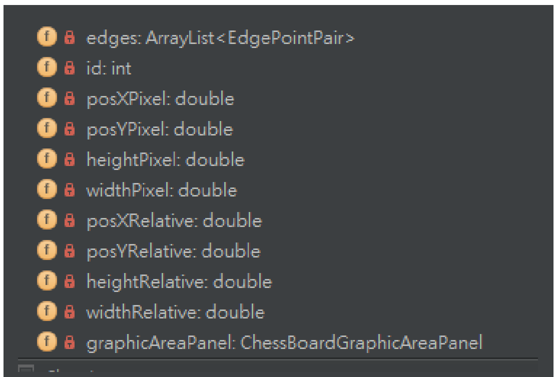
The EdgePointPair class describes the neighbor point of the point with the direction. The “direction” attribute stores the direction pointing to the target neighbor point; the ‘targetPoint” attribute stores the actual destination neighbor point.



```
Point(ChessBoardGraphicAreaPanel, int)
Point(ChessBoardGraphicAreaPanel, double, double, double, double, int)
setSizeByRelative(double, double, double, double): void
setSizeByPixel(double, double, double, double): void
getId(): int
getPixelInformation(): InfoPack
getRelativeInformation(): InfoPack
updateRelativeByPixel(): void
updatePixelByRelative(): void
addEdgePointPairs(String, Point): void
removeEdgePointPairs(String): void
removeEdgePointPairs(Point): void
getEdgeByPoint(Point): String
getPointByEdge(String): Point
getAllEdgePointPair(): ArrayList<EdgePointPair>
```

Fig. 5.1e.2.2 methods inside the PointClass

Figure 5.1e.2.2 shows the class methods of the Point class. In order to protect the data stored in the class. Utility methods to modify and access the data stored in the class are provided, such as the constructors, setSizeByRelative(), set sizeByPixel(), getID(), getPixelInformation(), getRelativeInformation(), addEdgePointPairs(), removeEdgePointPairs(), getEdgeByPoint(), getPointByEdge() and getAllEdgePointPair(). The function updateRelativeByPixel() will update the relative information of the point by the reference of the pixel information, the function updatePixelByRelative() just do the same way.

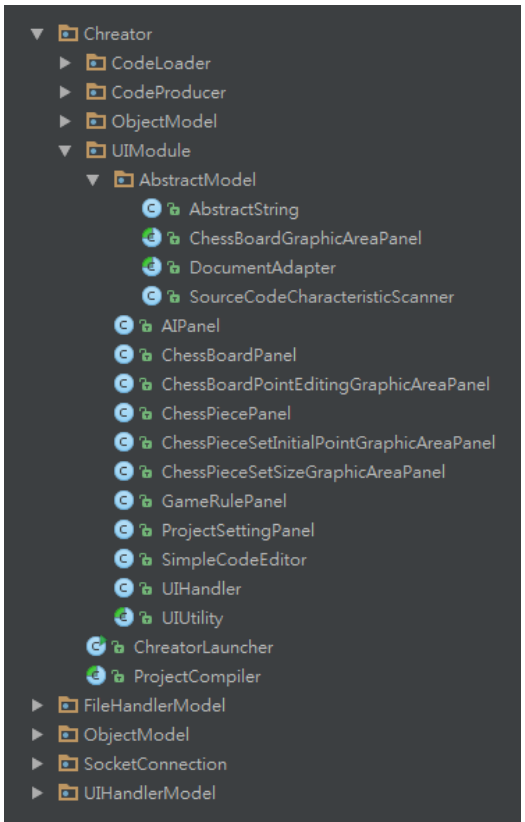


```
edges: ArrayList<EdgePointPair>
id: int
posXPixel: double
posYPixel: double
heightPixel: double
widthPixel: double
posXRelative: double
posYRelative: double
heightRelative: double
widthRelative: double
graphicAreaPanel: ChessBoardGraphicAreaPanel
```

Fig. 5.1e.2.3 methods inside the PointClass

Figure 5.1e.2.3 shows the class fields of the Point class. "edges" stores the list to all the neighbor points; "id" stores the ID of the point. It is useful for representing the chess piece placing point; "posXPixel" stores the pixel value of the x coordinate of the point; "posYPixel" stores the pixel value of the y coordinate of the point; "heightPixel" stores the pixel value of the height; "widthPixel" stores the pixel value of the width; "posXRelative", "posYRelative", "heightRelative" and "widthRelative" does the same of the corresponding attribute fields mentioned above, instead these values store the relative information. "graphicAreaPanel" indicates the layout that the point is referencing.

UIModule



```
▼ Chreator
  ▶ CodeLoader
  ▶ CodeProducer
  ▶ ObjectModel
  ▼ UIModule
    ▼ AbstractModel
        AbstractString
        ChessBoardGraphicAreaPanel
        DocumentAdapter
        SourceCodeCharacteristicScanner
      AIPanel
      ChessBoardPanel
      ChessBoardPointEditingGraphicAreaPanel
      ChessPiecePanel
      ChessPieceSetInitialPointGraphicAreaPanel
      ChessPieceSetSizeGraphicAreaPanel
      GameRulePanel
      ProjectSettingPanel
      SimpleCodeEditor
      UIHandler
      UIUtility
    ChreatorLauncher
    ProjectCompiler
▶ FileHandlerModel
▶ ObjectModel
▶ SocketConnection
▶ UIHandlerModel
```

Fig. 5.1f.1 UIModule of the Chreator

Figure 5.1f.1 shows the entire UIModule of the Chreator. The entire module is implemented by me. Li Chun Kit did some enhancement and minor change to part of the system for adapting the usage of CodeLoader and CodeProducer.

The AbstractModel contains the big utility classes and abstract model class that are extended by some other classes. The [name]Panel classes are the GUI of the Chreator such as AIPanel, ChessBoardPanel and so on. The SimpleCodeEditor class provides the semi-finished code editor for entire Chreator. The UIHandler is the main controller and the bridge of all the GUI panels. It creates the instance of the AIPanel, ChessBoardPanel, ChessPiecePanel, GameRulePanel and ProjectSettingPanel. GameRulePanel provides a SimpleCodeEditor for user to implementing the game rule coding; ChessBoardPanel provides a GraphicAreaPanel for users to set up the layout of the chess board, including the board image and chess piece placing points; ChessPiecePanel loads the chess piece profiles, setting up the appurtenance of the chess piece and its moving rule in code; ProjectSettingPanel setup the project location, project type, JDK location, project compilation function and chess game launching function. All the messages and information flow between panels and communication between the code producer and the code loader relies on this class. The UIUtility class is an abstract class that provides static utility methods for all the panel classes. This class does not have an actual instance object. The below graph is shown to explain the hierarchy relationship more clearly.

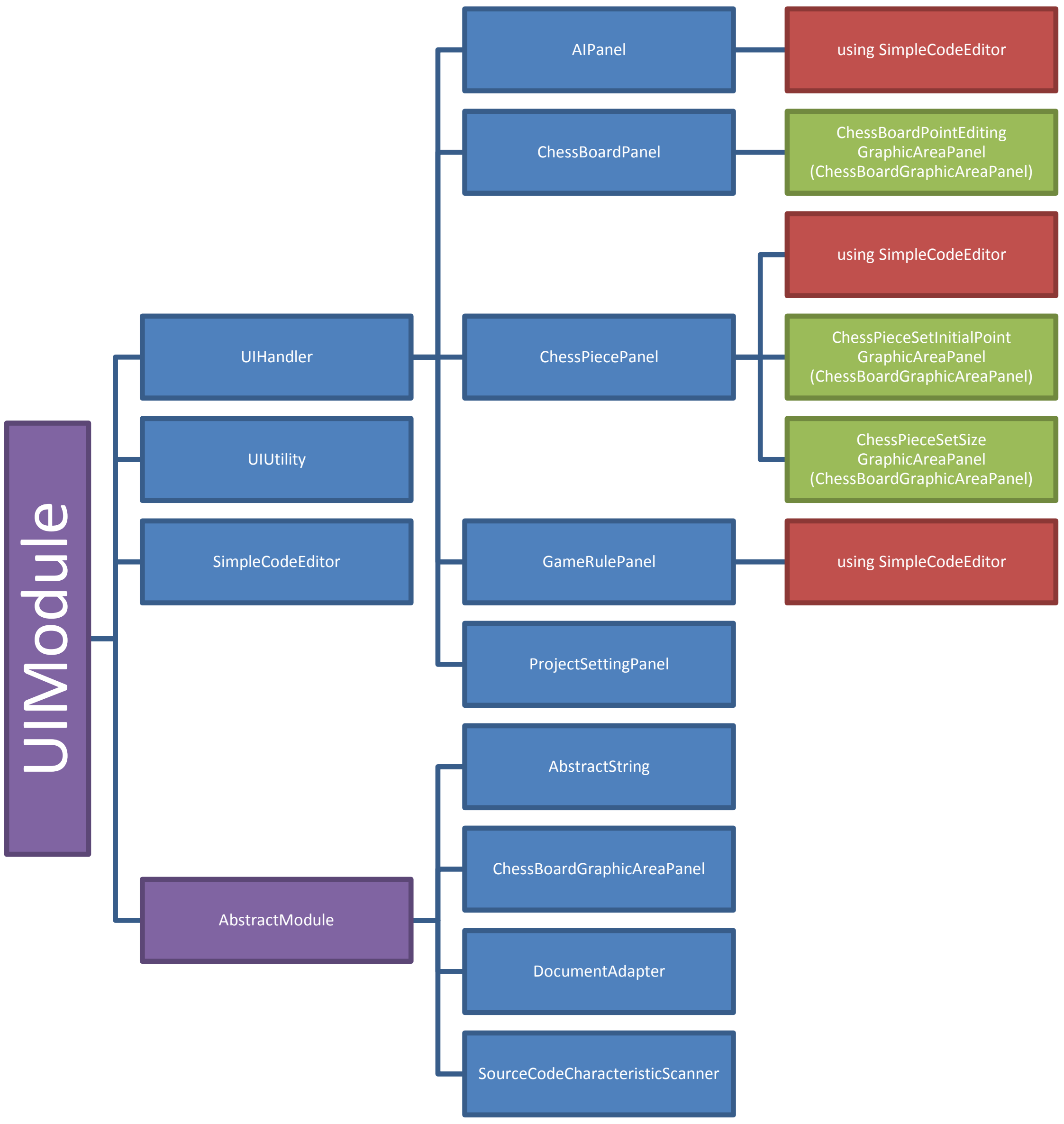
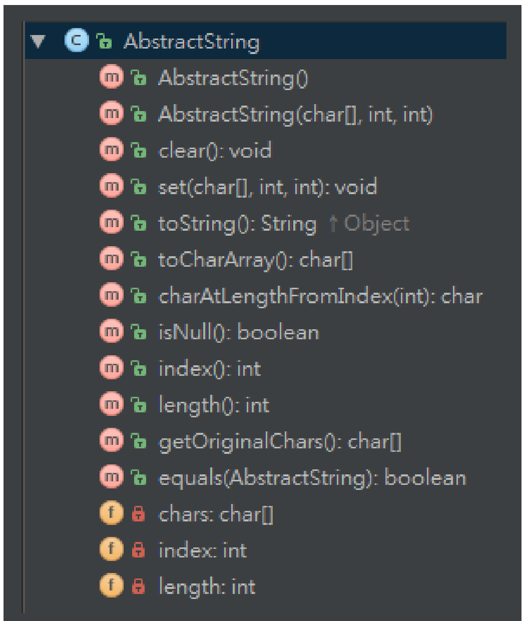

UIModule
UIHandler
UIUtility
SimpleCodeEditor
AbstractModule
AIPanel
ChessBoardPanel
ChessPiecePanel
GameRulePanel
ProjectSettingPanel
AbstractString
ChessBoardGraphicAreaPanel
DocumentAdapter
SourceCodeCharacteristicScanner
using SimpleCodeEditor
ChessBoardPointEditing GraphicAreaPanel (ChessBoardGraphicAreaPanel)
using SimpleCodeEditor
ChessPieceSetInitialPoint GraphicAreaPanel (ChessBoardGraphicAreaPanel)
ChessPieceSetSize GraphicAreaPanel (ChessBoardGraphicAreaPanel)
using SimpleCodeEditor

Fig. 5.1f.2 Relationship between classes inside the UIModule

The following part will discuss about the important classes of the Chreator for highlighting the important technical coding methods applied.

UIModule / AbstractModel / AbstractString

This class is implemented for the Java keyword scanning in the SimpleCodeEditor. The original method for finding Java keywords in the SimpleCodeEditor first reads all the text in the JTextPane, and then chops the full code string into substrings for analysis. However, this algorithm involves lots of String.subString(), new String() and other methods that lead to creating lots of substring. The computer spends over 100 ms to finish the analysis if the code contains several ten thousands characters. The AbstractString helps to solve this problem.



```
▼ AbstractString
    AbstractString()
    AbstractString(char[], int, int)
    clear(): void
    set(char[], int, int): void
    toString(): String ↑Object
    toCharArray(): char[]
    charAtLengthFromIndex(int): char
    isNull(): boolean
    index(): int
    length(): int
    getOriginalChars(): char[]
    equals(AbstractString): boolean
    chars: char[]
    index: int
    length: int
```

Fig. 5.1g Structure of the AbstractString class

The class only accepts character arrays and stores start index and length of the abstract string. Only a new string or character array storing the real representation of the abstract string is created when toString() and toCharArray() is called respectively. Accessing or even changing the index, length and the provided character array do not trigger the AbstractString to generate a new String object describing the AbstractString.

UIModule / AbstractModel / ChessBoardGraphicPanel

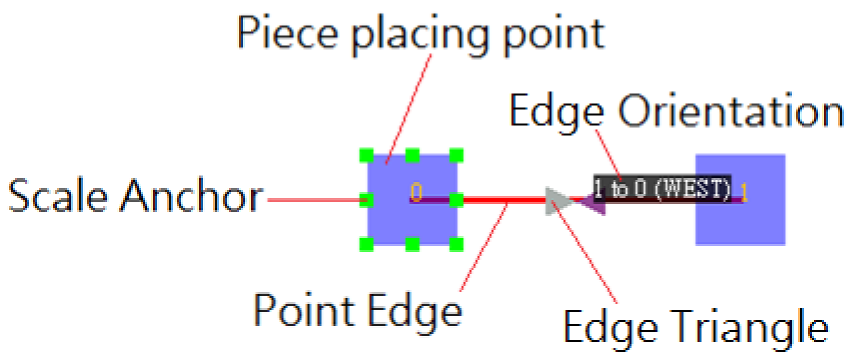


Fig.5.1i concept of a chess board

Let us review the structure of a chess board before talking about the ChessBoardGraphicAreaPanel. There are piece placing points on the chess board. There are point edges between points. There must be an orientation to the neighbor points. In order to show the orientation between points, the EdgeOrientationTriangle is introduced. The points can be scaled and translated, to do so, we introduce a scale anchor, each piece placing point must be assigned to an unique ID:

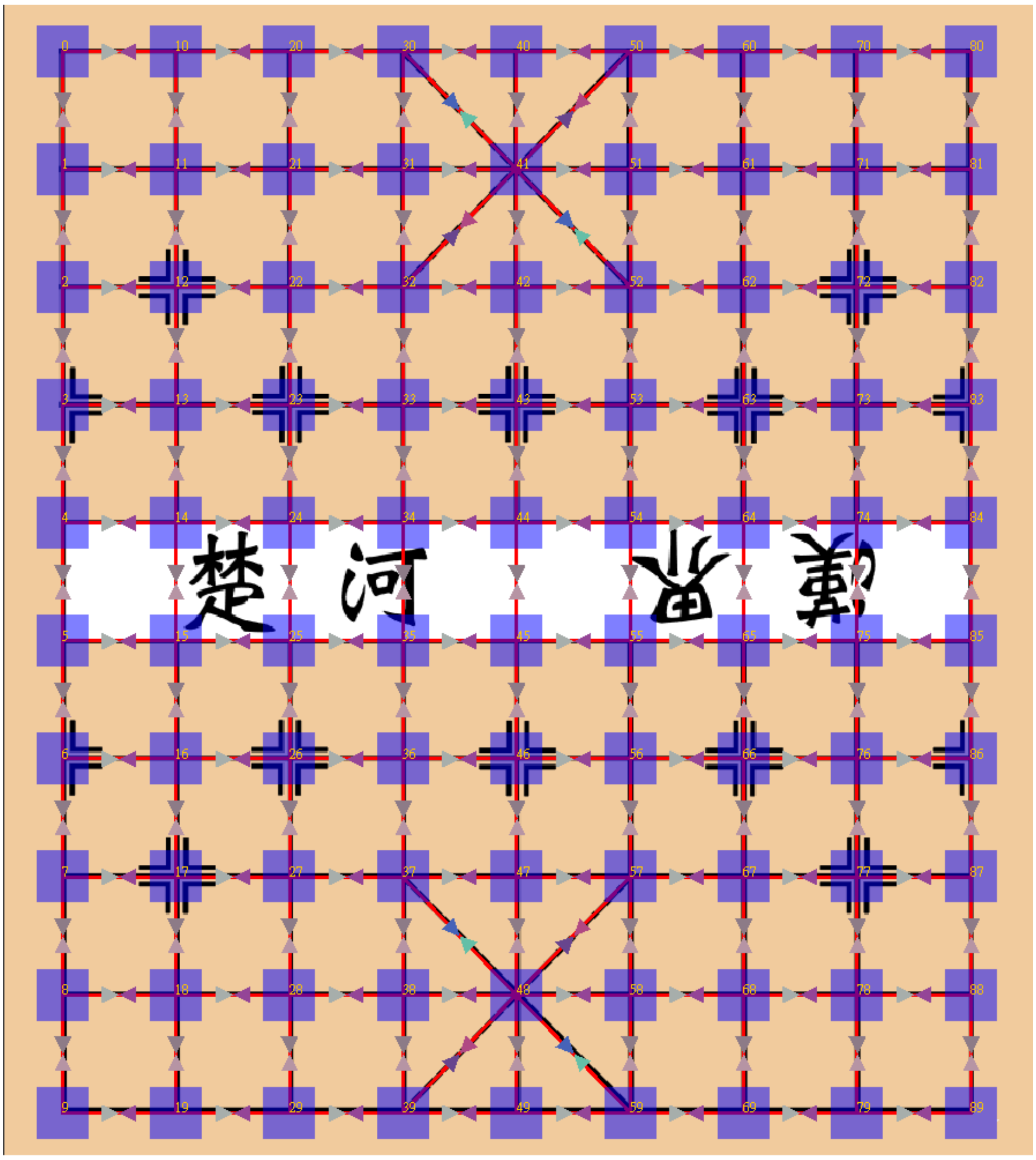
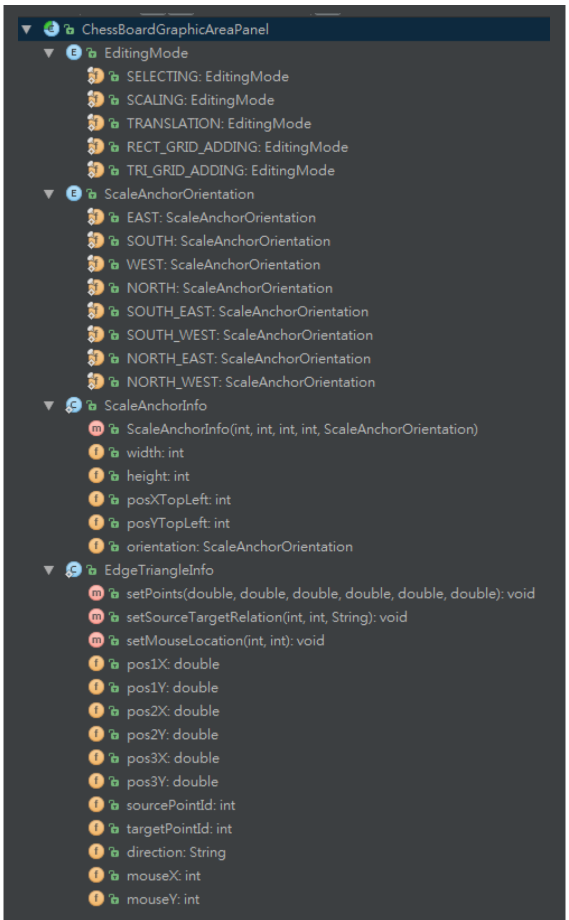


Fig.5.1j A XiangQi chess board with chess piece placing points, edge and directions.

Figure 5.1j shows the complete layout after adding the chess piece placing points, point edge and appropriate orientations. We can introduce the ChessBoardGraphicAreaPanel and how it groups the repeated code for all the other types of ChessBoardGraphicAreaPanels. ChessBoardGraphicAreaPanel is responsible to provide the basic chess board rendering functions. It draws the chess board image, chess piece placing points, edges between points, edge triangles, scale anchors and edge orientation information.



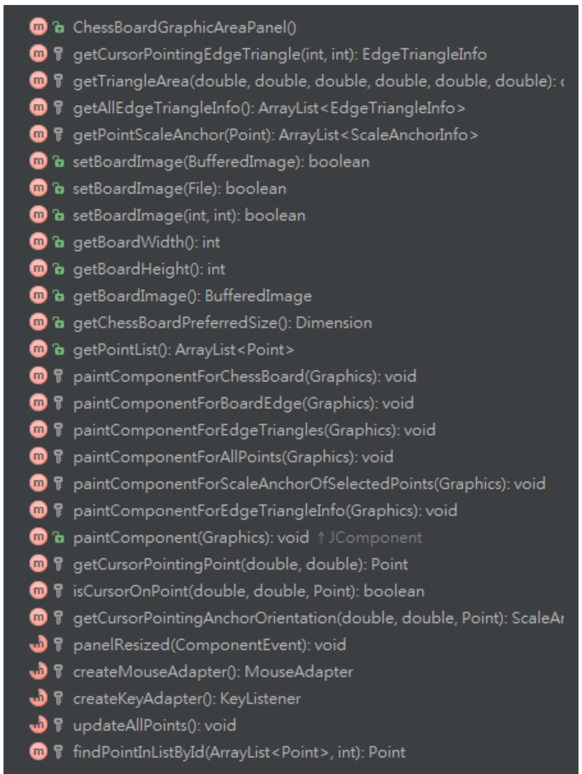
```
ChessBoardGraphicAreaPanel
    EditingMode
        SELECTING: EditingMode
        SCALING: EditingMode
        TRANSLATION: EditingMode
        RECT_GRID_ADDING: EditingMode
        TRI_GRID_ADDING: EditingMode
    ScaleAnchorOrientation
        EAST: ScaleAnchorOrientation
        SOUTH: ScaleAnchorOrientation
        WEST: ScaleAnchorOrientation
        NORTH: ScaleAnchorOrientation
        SOUTH_EAST: ScaleAnchorOrientation
        SOUTH_WEST: ScaleAnchorOrientation
        NORTH_EAST: ScaleAnchorOrientation
        NORTH_WEST: ScaleAnchorOrientation
    ScaleAnchorInfo
        ScaleAnchorInfo(int, int, int, int, ScaleAnchorOrientation)
        width: int
        height: int
        posXTopLeft: int
        posYTopLeft: int
        orientation: ScaleAnchorOrientation
    EdgeTriangleInfo
        setPoints(double, double, double, double, double, double): void
        setSourceTargetRelation(int, int, String): void
        setMouseLocation(int, int): void
        pos1X: double
        pos1Y: double
        pos2X: double
        pos2Y: double
        pos3X: double
        pos3Y: double
        sourcePointId: int
        targetPointId: int
        direction: String
        mouseX: int
        mouseY: int
```

Fig. 5.1k.1 inner classes of ChessBoardGraphicAreaPanel

There are 4 inner classes in the ChessBoardGraphicAreaPanel. They are used to define some stage of the panel. Some classes are also used as information exchange just like the Point class in ObjectModel.

- EditingMode defines how the system should interact with the next mouse click for example selecting, scaling or translating the chess piece placing points and so on.
- ScaleAnchorOrientation defines the orientation types of the scale anchor
- ScaleAnchorInfo just like the Point class in ObjectModel. It is only much simple and only stores size and coordinates in pixel size of a scale anchor.
- EdgeTriangleInfo collects the information of a edge triangle. pos1X, pos2X, pos3X, pos1Y, pos2Y and pos3Y define the 3 vertices of the edge triangle. Shis

classes also stores the source point ID, target point ID, mouse cursor coordinates and so.



```
ChessBoardGraphicAreaPanel()
getCursorPointingEdgeTriangle(int, int): EdgeTriangleInfo
getTriangleArea(double, double, double, double, double, double):
getAllEdgeTriangleInfo(): ArrayList<EdgeTriangleInfo>
getPointScaleAnchor(Point): ArrayList<ScaleAnchorInfo>
setBoardImage(BufferedImage): boolean
setBoardImage(File): boolean
setBoardImage(int, int): boolean
getBoardWidth(): int
getBoardHeight(): int
getBoardImage(): BufferedImage
getChessBoardPreferredSize(): Dimension
getPointList(): ArrayList<Point>
paintComponentForChessBoard(Graphics): void
paintComponentForBoardEdge(Graphics): void
paintComponentForEdgeTriangles(Graphics): void
paintComponentForAllPoints(Graphics): void
paintComponentForScaleAnchorOfSelectedPoints(Graphics): void
paintComponentForEdgeTriangleInfo(Graphics): void
paintComponent(Graphics): void ↑JComponent
getCursorPointingPoint(double, double): Point
isCursorOnPoint(double, double, Point): boolean
getCursorPointingAnchorOrientation(double, double, Point): ScaleAr
panelResized(ComponentEvent): void
createMouseAdapter(): MouseAdapter
createKeyAdapter(): KeyListener
updateAllPoints(): void
findPointInListById(ArrayList<Point>, int): Point
```

Fig. 5.1k.2 methods of ChessBoardGraphicAreaPanel

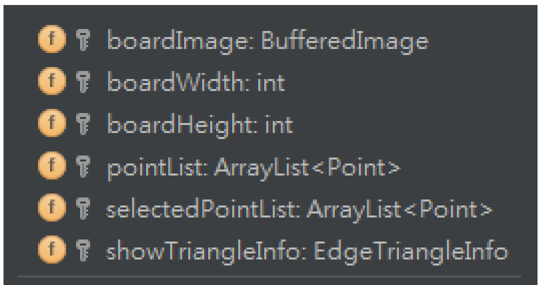
ChessBoardGraphicAreaPanel helps to reduce the repeated code when there are several different type of chess board graphic area panels to be implemented because it group the repeated part here.

- The get[Name]() and set[Name]() functions provide the exchange of class field data between different classes. Such that the data can be keep synchronized among classes. This can reduce the chance of error.

The child class can skip implementing those methods must can still use them.

- paintComponentFor[Name]() methods define the default painting behavior of chess board, chess piece placing point, the scale anchors, edge triangles and so on. Such that other chess board graphic area panel can simply call the function that the super class will draw for the child class.
- getCursorPointingPoint(), isCursorOnPoint() and getCursorPointingAnchorOrientation() are utility methods to check is the cursor on some piece placing points or is on some scale anchor.
- panelResized(), createMouseAdapter(), createKeyAdapter() and updateAllPoints() are abstract methods that force the child classes to handle the mouse and keyboard action of the panel and also update all the piece placing

points when there is a update to the graphic of the chess board graphic area panel.

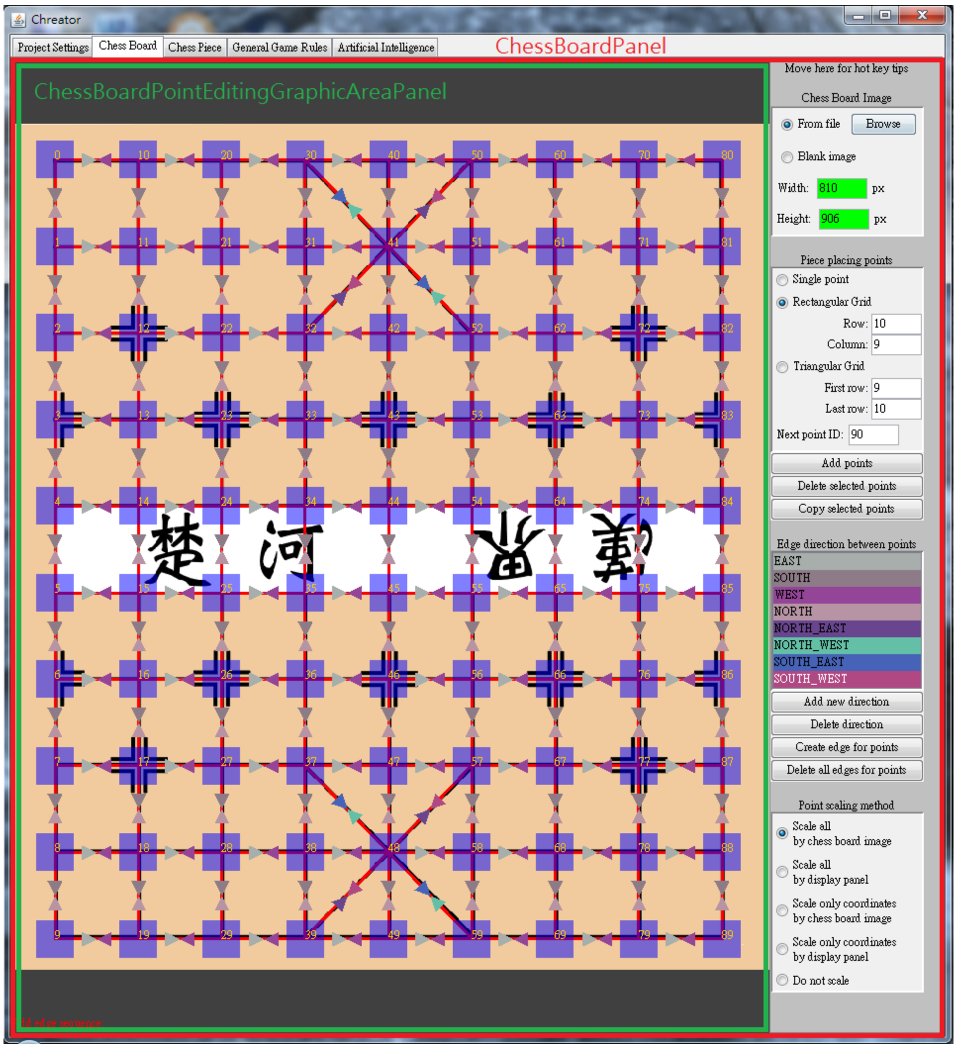


```
boardImage: BufferedImage
boardWidth: int
boardHeight: int
pointList: ArrayList<Point>
selectedPointList: ArrayList<Point>
showTriangleInfo: EdgeTriangleInfo
```

Fig. 5.1k.3 fields of ChessBoardGraphicAreaPanel

There are only few fields in the class. They just store the chess board image, its size, all the piece placing points, the selected points and the current displaying edge triangle information.

UIModule / ChessBoardPanel and UIModule / ChessBoardPointEditingGraphicAreaPanel



Chess Board Image
[✓ From file] Blank image
Width: 810 px
Height: 906 px

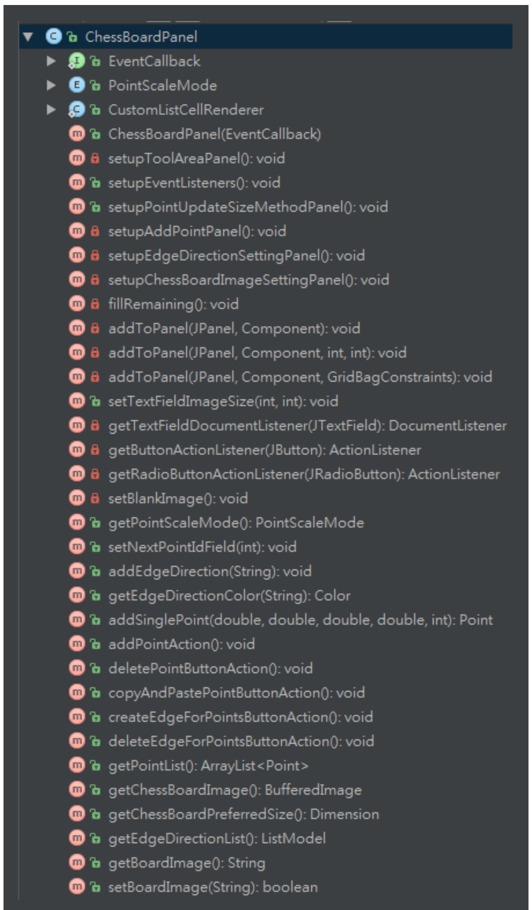
Piece placing points
Single point [✓ Rectangular Grid]
Row: 10
Column: 9
Triangular Grid
First row: 9
Last row: 10
Next point ID: 90

Edge direction between points
EAST
SOUTH
WEST
NORTH
NORTH_EAST
NORTH_WEST
SOUTH_EAST
SOUTH_WEST

Point scaling method
[✓ Scale all by chess board image]
Scale all by display panel
Scale only coordinates by chess board image
Scale only coordinates by display panel
Do not scale

Fig. 5.1l.1 The chess board tab contains a ChessBoardPanel (red frame highlighted). The ChessBoardPanel contains a ChessBoardPointEditingGraphicAreaPanel (green frame highlighted)

The ChessBoardPanel extends from JPanel, and it mainly controls the vertical tool area panel on the left hand side and communicates with the ChessBoardPointEditingGraphicAreaPanel.

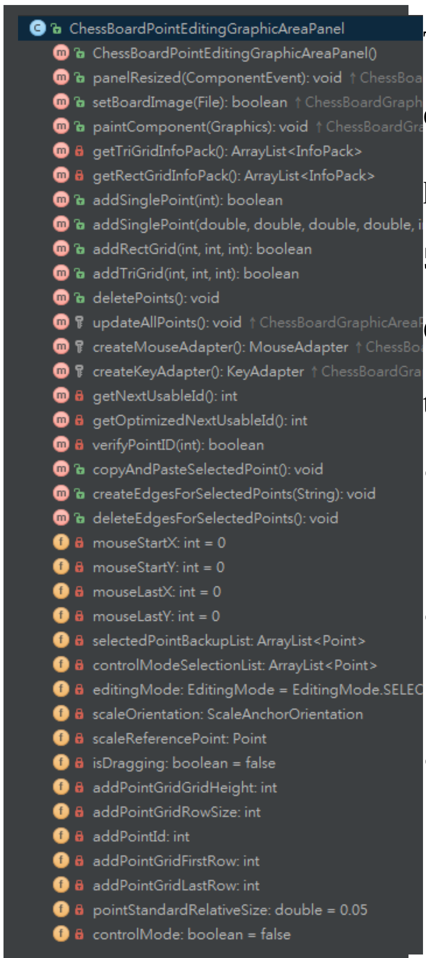


```
ChessBoardPanel
    EventCallback
    PointScaleMode
    CustomListCellRenderer
    ChessBoardPanel(EventCallback)
    setupToolAreaPanel(): void
    setupEventListeners(): void
    setupPointUpdateSizeMethodPanel(): void
    setupAddPointPanel(): void
    setupEdgeDirectionSettingPanel(): void
    setupChessBoardImageSettingPanel(): void
    fillRemaining(): void
    addToPanel(JPanel, Component): void
    addToPanel(JPanel, Component, int, int): void
    addToPanel(JPanel, Component, GridBagConstraints): void
    setTextFieldImageSize(int, int): void
    getTextFieldDocumentListener(JTextField): DocumentListener
    getButtonActionListener(JButton): ActionListener
    getRadioButtonActionListener(JRadioButton): ActionListener
    setBlankImage(): void
    getPointScaleMode(): PointScaleMode
    setNextPointIdField(int): void
    addEdgeDirection(String): void
    getEdgeDirectionColor(String): Color
    addSinglePoint(double, double, double, double, int): Point
    addPointAction(): void
    deletePointButtonAction(): void
    copyAndPastePointButtonAction(): void
    createEdgeForPointsButtonAction(): void
    deleteEdgeForPointsButtonAction(): void
    getPointList(): ArrayList<Point>
    getChessBoardImage(): BufferedImage
    getChessBoardPreferredSize(): Dimension
    getEdgeDirectionList(): ListModel
    getBoardImage(): String
    setBoardImage(String): boolean
```

Fig. 5.1l.2 inner classese and methods ChessBoardPanel

- The inner class PointScaleMode defines the status of the point scaling method on the bottom right of the tool area panel
- The inner class CustomListRenderer is responsible for rendering the colored edge orientation list under the edge direction between points area
- Methods setupToolAreaPanel(), setupPointUpdateSizeMethodPanel(), setupAddPointPanel(), setupEdgeDirectionSettingPanel(), setupChessBoardImageSettingPanel(), fillRemaining() and addToPanel() setup the layout of the tool area panel.
- Methods getTextFieldDocumentListener(), getButtonActionListener() and getRadioButtonActionListener() define the response action of the GUI components such as buttons, text fields and radio buttons.
- The remaining methods are used to operate the functional practices such as adding and removing piece placing points, edge direction between points and so on.

- The fields of the ChessBoardPanel are not shown here, as they are mainly the GUI components of the panel, such as buttons, radio buttons, panels, radio button groups, text fields, labels and so on.



```
ChessBoardPointEditingGraphicAreaPanel
    ChessBoardPointEditingGraphicAreaPanel()
    panelResized(ComponentEvent): void ↑ChessBoa
    setBoardImage(File): boolean ↑ChessBoardGraph
    paintComponent(Graphics): void ↑ChessBoardGra
    getTriGridInfoPack(): ArrayList<InfoPack>
    getRectGridInfoPack(): ArrayList<InfoPack>
    addSinglePoint(int): boolean
    addSinglePoint(double, double, double, double, i
    addRectGrid(int, int, int): boolean
    addTriGrid(int, int, int): boolean
    deletePoints(): void
    updateAllPoints(): void ↑ChessBoardGraphicAreaI
    createMouseAdapter(): MouseAdapter ↑ChessBo
    createKeyAdapter(): KeyAdapter ↑ChessBoardGra
    getNextUsableId(): int
    getOptimizedNextUsableId(): int
    verifyPointID(int): boolean
    copyAndPasteSelectedPoint(): void
    createEdgesForSelectedPoints(String): void
    deleteEdgesForSelectedPoints(): void
    mouseStartX: int = 0
    mouseStartY: int = 0
    mouseLastX: int = 0
    mouseLastY: int = 0
    selectedPointBackupList: ArrayList<Point>
    controlModeSelectionList: ArrayList<Point>
    editingMode: EditingMode = EditingMode.SELEC
    scaleOrientation: ScaleAnchorOrientation
    scaleReferencePoint: Point
    isDragging: boolean = false
    addPointGridGridHeight: int
    addPointGridRowSize: int
    addPointId: int
    addPointGridFirstRow: int
    addPointGridLastRow: int
    pointStandardRelativeSize: double = 0.05
    controlMode: boolean = false
```

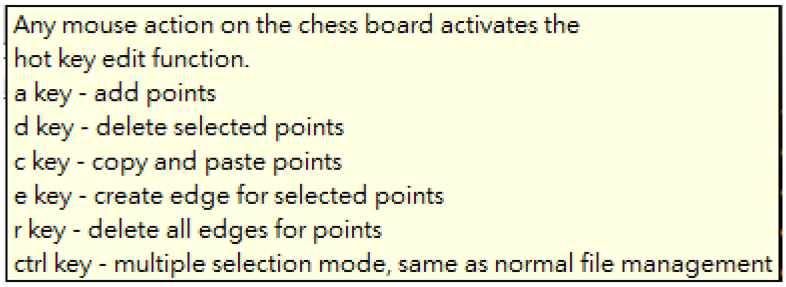
Fig. 5.1l.3 entire structure of the ChessBoardPointEditingGraphicAreaPanel

The ChessBoardPointEditingGraphicAreaPanel is an extension of ChessBoardGraphicAreaPanel. Besides painting the chess board design as shown in figure 5.1l.1 (this will be done by ChessBoardGraphicAreaPanel as mentioned above), this class mainly focus on:

- handling the mouse and keyboard interaction; (createMouseAdapter(), createKeyAdapter())
- applying the add and removing piece placing point action; (addSinglePoint(), deletePoints())
- grid points adding action; copy and parse points; (getTriGridInfoPack(), getRectGridInfoPack(), addRectGrid(), addTriGrid(), copyAndPasteSelectedPoint())
- Point edges managements. (createEdgesForSelectedPoints(),deleteEdgesForSelectedPoints())

The numbers of methods and fields in this class is not high, but the createMouseAdapter() method which provide the definition to the handling action of the mouse event is about 400 line out of total about 800 lines in this class. The mouse event mechanism is too complicated. It needs to determine whether the next action is clicked on a point, empty space or a scale anchor; whether the mouse is dragging, clicking, moving or others. The combinations of events finally form the function of single point or multiple points translation, scaling, selection and copying.

The createKeyAdapter() handles the key input.

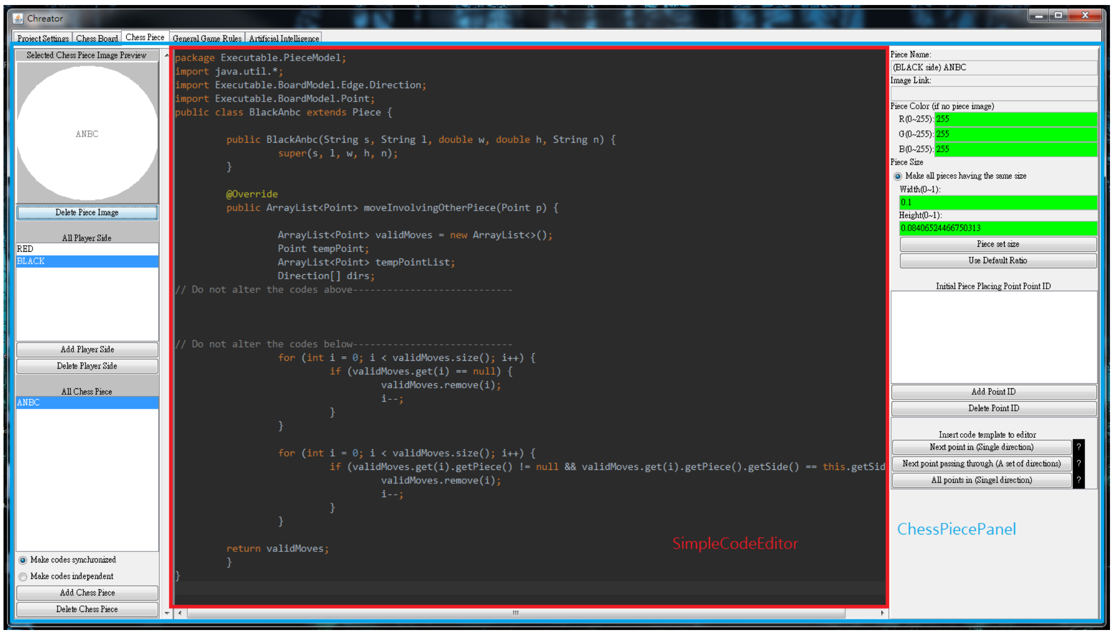


Any mouse action on the chess board activates the
hot key edit function.
a key - add points
d key - delete selected points
c key - copy and paste points
e key - create edge for selected points
r key - delete all edges for points
ctrl key - multiple selection mode, same as normal file management

Fig. 5.1l.4 Tooltip prompted up by moving the cursor to the label "Move here for hot key tips"

Hot keys are supported to quickly operate the chess board point editing panel.

UIModule / ChessPiecePanel

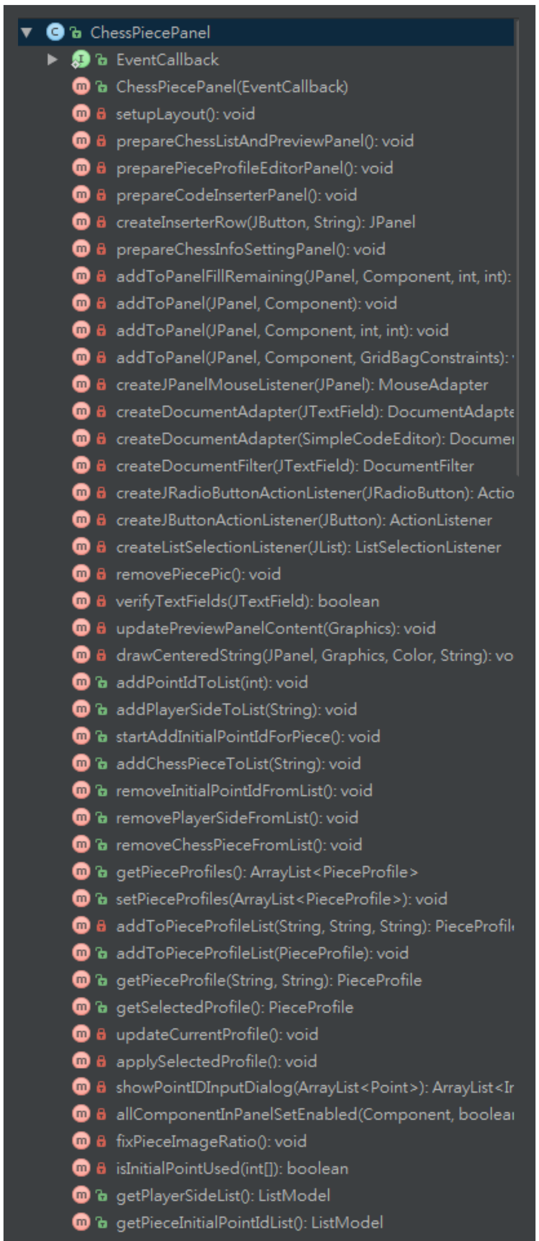


```
package Executable.PieceModel;
import java.util.*;
import Executable.BoardModel.Edge.Direction;
import Executable.BoardModel.Point;
public class BlackAnbc extends Piece {

        public BlackAnbc(String s, String l, double w, double h, String n) {
                super(s, l, w, h, n);
        }

        @Override
        public ArrayList<Point> moveInvolvingOtherPiece(Point p) {

                ArrayList<Point> validMoves = new ArrayList<>();
                Point tempPoint;
                ArrayList<Point> tempPointList;
                Direction[] dirs;
// Do not alter the codes above---------------------------



// Do not alter the codes below---------------------------
                for (int i = 0; i < validMoves.size(); i++) {
                        if (validMoves.get(i) == null) {
                                validMoves.remove(i);
                                i--;
                        }
                }

                for (int i = 0; i < validMoves.size(); i++) {
                        if (validMoves.get(i).getPiece() != null && validMoves.get(i).getPiece().getSide() == this.getSid
                                validMoves.remove(i);
                                i--;
                        }
                }

        return validMoves;
        }
}
```

Piece Name:
(BLACK side) ANBC
Image Link:
Piece Color (if no piece image)
R(0~255): 255
G(0~255): 255
B(0~255): 255
Piece Size
Make all pieces having the same size
Width(0~1):
0.1
Height(0~1):
0.08406524466750313
Initial Piece Placing Point Point ID

Fig.5.1m.1 The GUI of the chess piece editing tab. It contains the ChessPiecePanel and the SimpleCodeEditor.

The above figure shows the ChessPiecePanel highlighted in blue in the chess piece tab. Similar to the chess board tab, the ChessPiecePanel contains a SimpleCodeEditor highlighted in red, but it aims on managing the tool area of both sides, such that users can quickly edit the appearance of the chess piece, player sides, name, placing location when the game starts, convenient coding for the .the chess piece.



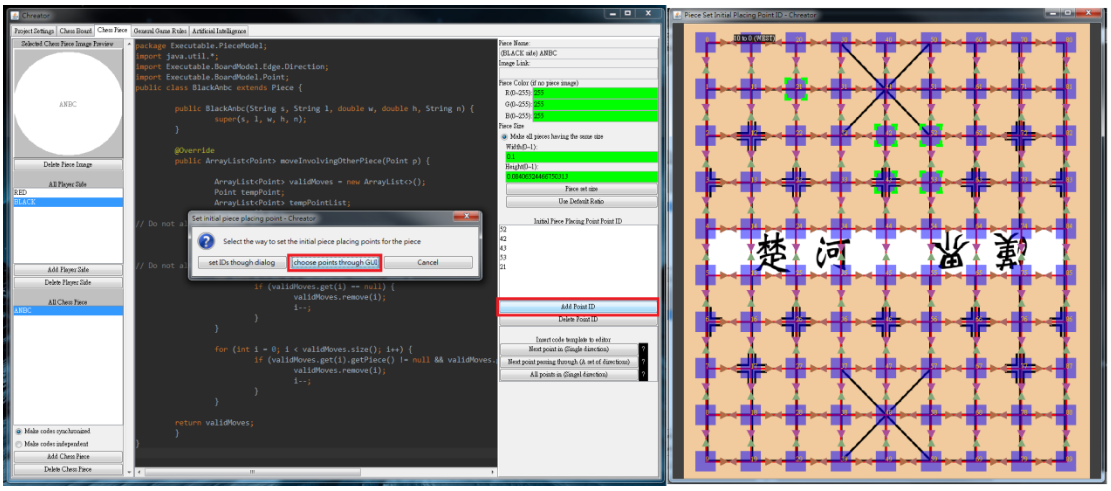
```
ChessPiecePanel
    EventCallback
    ChessPiecePanel(EventCallback)
    setupLayout(): void
    prepareChessListAndPreviewPanel(): void
    preparePieceProfileEditorPanel(): void
    prepareCodeInserterPanel(): void
    createInserterRow(JButton, String): JPanel
    prepareChessInfoSettingPanel(): void
    addToPanelFillRemaining(JPanel, Component, int, int):
    addToPanel(JPanel, Component): void
    addToPanel(JPanel, Component, int, int): void
    addToPanel(JPanel, Component, GridBagConstraints):
    createJPanelMouseListener(JPanel): MouseAdapter
    createDocumentAdapter(JTextField): DocumentAdapte
    createDocumentAdapter(SimpleCodeEditor): Documei
    createDocumentFilter(JTextField): DocumentFilter
    createJRadioButtonActionListener(JRadioButton): Actio
    createJButtonActionListener(JButton): ActionListener
    createListSelectionListener(JList): ListSelectionListener
    removePiecePic(): void
    verifyTextFields(JTextField): boolean
    updatePreviewPanelContent(Graphics): void
    drawCenteredString(JPanel, Graphics, Color, String): vo
    addPointIdToList(int): void
    addPlayerSideToList(String): void
    startAddInitialPointIdForPiece(): void
    addChessPieceToList(String): void
    removeInitialPointIdFromList(): void
    removePlayerSideFromList(): void
    removeChessPieceFromList(): void
    getPieceProfiles(): ArrayList<PieceProfile>
    setPieceProfiles(ArrayList<PieceProfile>): void
    addToPieceProfileList(String, String, String): PieceProfil
    addToPieceProfileList(PieceProfile): void
    getPieceProfile(String, String): PieceProfile
    getSelectedProfile(): PieceProfile
    updateCurrentProfile(): void
    applySelectedProfile(): void
    showPointIDInputDialog(ArrayList<Point>): ArrayList<Ir
    allComponentInPanelSetEnabled(Component, booleai
    fixPieceImageRatio(): void
    isInitialPointUsed(int[]): boolean
    getPlayerSideList(): ListModel
    getPieceInitialPointIdList(): ListModel
```

Fig. 5.1m.2 Inner class and methods of the ChessPiecePanel

- Similar to the ChessBoarddPanel, there are lots of methods that are used to setup the layout of the panel, they are setupLayout(), prepareChessListAndPreviewPanel(), preparePieceProfileEditorPanel(), prepareCodeInserterPanel(), createInserterRow(), prepareChessInfoSettingPanel(), addToPanelFillRemaining() and addToPanel();
- The create[Name]Listener() methods and create[Name]Adapter() methods defines the event handlers for the button, text fields, radio buttons and list.
- All the remaining methods handle the chess piece profiles and their data profile, for example the chess piece image, color, size, initial placing points of the chess piece and so on.

The fields of the ChessPiecePanel is similar to the ChessBoardPanel. Most of them are the GUI components such as buttons, text fields, radio buttons and list.

UIModule / ChessPieceSetInitialPointGraphicAreaPanel



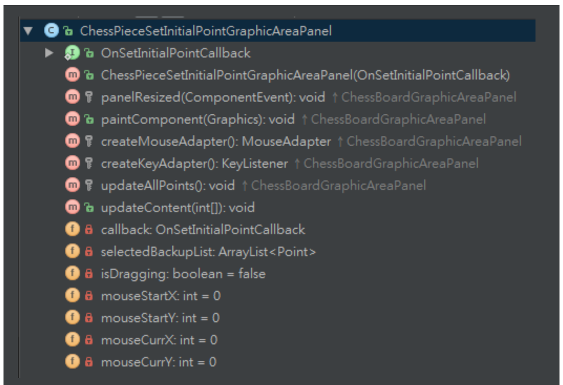
```
package Executable.PieceModel;
import java.util.*;
import Executable.BoardModel.Edge.Direction;
import Executable.BoardModel.Point;
public class BlackAnbc extends Piece {

        public BlackAnbc(String s, String l, double w, double h, String n) {
                super(s, l, w, h, n);
        }

        @Override
        public ArrayList<Point> moveInvolvingOtherPiece(Point p) {

                ArrayList<Point> validMoves = new ArrayList<>();
                Point tempPoint;
                ArrayList<Point> tempPointList;

// Do not al

// Do not al
                        if (validMoves.get(i) == null) {
                                validMoves.remove(i);
                                i--;
                        }
                }

                for (int i = 0; i < validMoves.size(); i++) {
                        if (validMoves.get(i).getPiece() != null && validMoves.
                                validMoves.remove(i);
                                i--;
                        }
                }

        return validMoves;
        }
}
```

Set initial piece placing point - Chreator

Select the way to set the initial piece placing points for the piece

Fig.5.1m.3 The GUI is shown after Add Point ID button is pressed then choose points through GUI is pressed. The window frame contains a ChessBoardSetInitialPointGraphicAreaPanel.

Figure 5.1m.3 shows how to open the window frame with the ChessPieceSetInitialPointGraphicAreaPanel. We can achieve this by clicking on the "Add Point ID" button then select the "choose points through GUI". This class aims to use a GUI format to let the user to set the set point that chess piece would be placed to when the game starts.



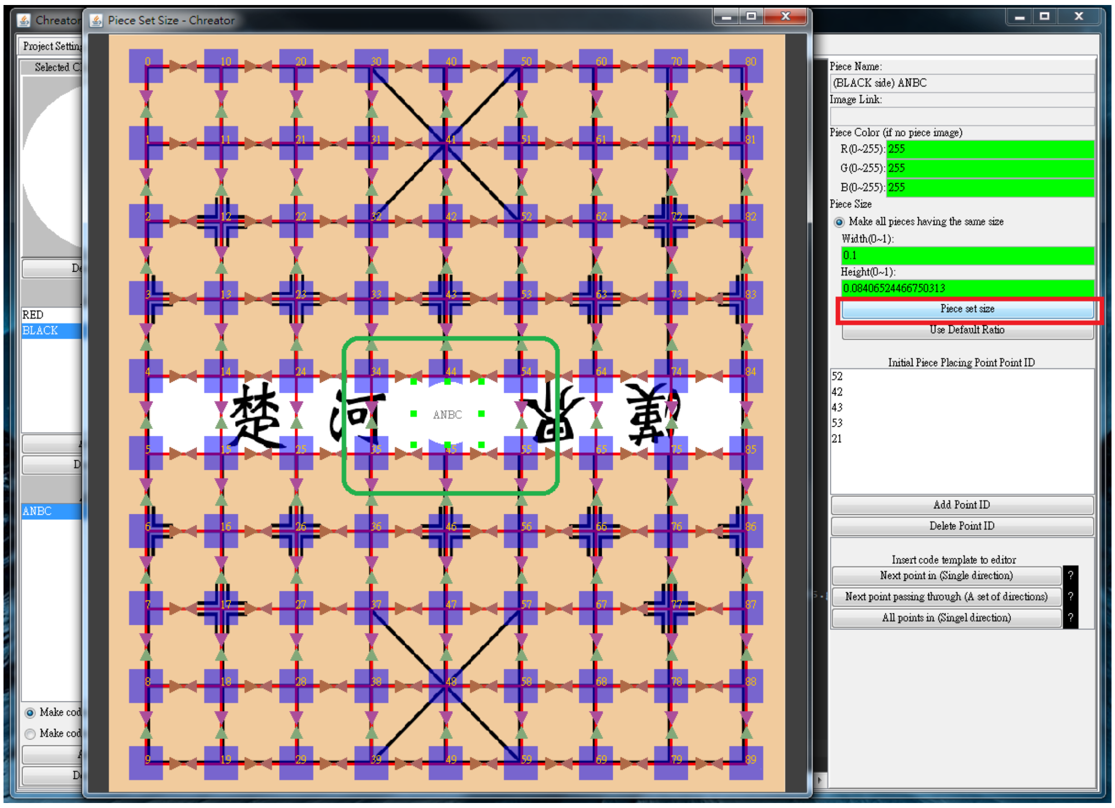
```
ChessPieceSetInitialPointGraphicAreaPanel
    OnSetInitialPointCallback
    ChessPieceSetInitialPointGraphicAreaPanel(OnSetInitialPointCallback)
    panelResized(ComponentEvent): void ↑ChessBoardGraphicAreaPanel
    paintComponent(Graphics): void ↑ChessBoardGraphicAreaPanel
    createMouseAdapter(): MouseAdapter ↑ChessBoardGraphicAreaPanel
    createKeyAdapter(): KeyListener ↑ChessBoardGraphicAreaPanel
    updateAllPoints(): void ↑ChessBoardGraphicAreaPanel
    updateContent(int[]): void
    callback: OnSetInitialPointCallback
    selectedBackupList: ArrayList<Point>
    isDragging: boolean = false
    mouseStartX: int = 0
    mouseStartY: int = 0
    mouseCurrX: int = 0
    mouseCurrY: int = 0
```

Fig. 5.1m.4 the entire class of ChessPieceSetInitialPointGraphicAreaPanel

Since the class extends from the ChessBoardGraphicAreaPanel, most of the component painting can be done by just calling the methods in the parent class. This class just need to write the own mouse event handling scheme, update all points method and the update for the window resize method.

The mouse event handler only accepts clicking on a chess piece point as selecting or unselecting.

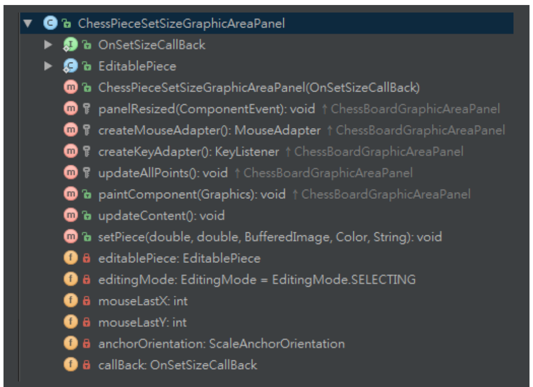
UIModule / ChessPieceSetSizeGraphicAreaPanel


Piece Set Size - Chreator
Piece set size
Use Default Ratio
Add Point ID
Delete Point ID
Insert code template to editor
Next point in (Single direction)
Next point passing through (A set of directions)
All points in (Singel direction)

Piece Name: (BLACK side) ANBC
Image Link:
Piece Color (if no piece image)
R(0~255): 255
G(0~255): 255
B(0~255): 255
Piece Size
[✓ Make all pieces having the same size]
Width(0~1): 0.1
Height(0~1): 0.08406524466750313
Initial Piece Placing Point Point ID
52
42
43
53
21

Fig.5.1m.5 The GUI of the shown after Piece set size button is pressed. The new window contains a ChessPieceSetSizeGraphicAreaPanel

The ChessPieceSetSizeGraphicAreaPanel class extends from ChessBoardGraphicAreaPanel. It aims to provide a GUI for users to adjust the size and the ratio of an individual chess piece. It can be called by pressing the “Piece set size” button.

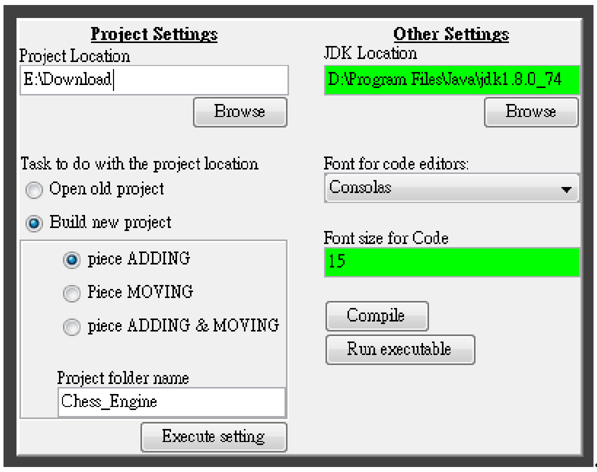


```
ChessPieceSetSizeGraphicAreaPanel
    OnSetSizeCallBack
    EditablePiece
    ChessPieceSetSizeGraphicAreaPanel(OnSetSizeCallBack)
    panelResized(ComponentEvent): void ↑ChessBoardGraphicAreaPanel
    createMouseAdapter(): MouseAdapter ↑ChessBoardGraphicAreaPanel
    createKeyAdapter(): KeyListener ↑ChessBoardGraphicAreaPanel
    updateAllPoints(): void ↑ChessBoardGraphicAreaPanel
    paintComponent(Graphics): void ↑ChessBoardGraphicAreaPanel
    updateContent(): void
    setPiece(double, double, BufferedImage, Color, String): void
    editablePiece: EditablePiece
    editingMode: EditingMode = EditingMode.SELECTING
    mouseLastX: int
    mouseLastY: int
    anchorOrientation: ScaleAnchorOrientation
    callBack: OnSetSizeCallBack
```

Fig. 5.1m.6 the entire class of ChessPieceSetSizeGraphicAreaPanel

Similar to the ChessPieceSetInitialPointGraphicAreaPanel, the parent class handles the drawing functions. This class just implements the mouse event handler for mouse click, drag drop and so on; update all points method and window resize method.

UIModule / ProjectSettingPanel

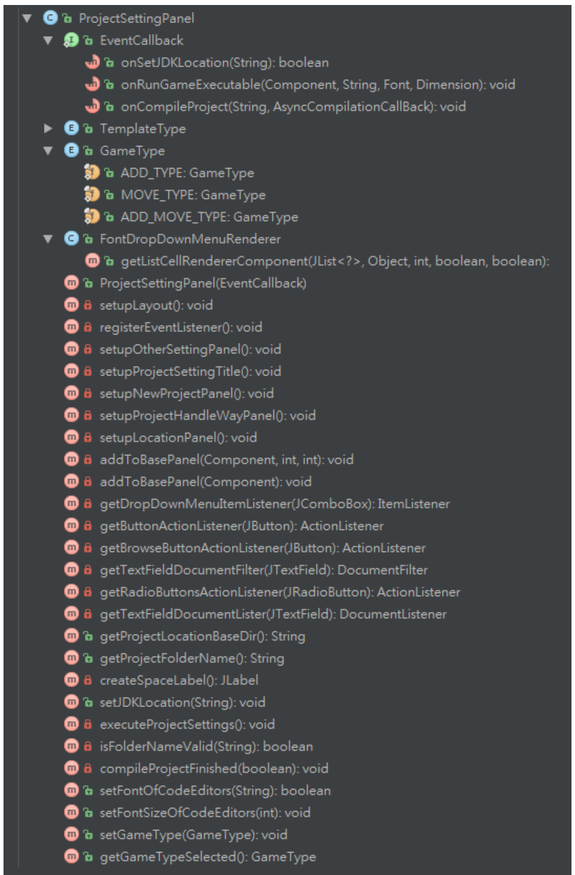


```
Project Settings                          Other Settings
Project Location                          JDK Location
E:\Download                               D:\Program Files\Java\jdk1.8.0_74
                          [Browse]                                  [Browse]

Task to do with the project location      Font for code editors:
( ) Open old project                      Consolas
(•) Build new project
    (•) piece ADDING                      Font size for Code
    ( ) Piece MOVING                      15
    ( ) piece ADDING & MOVING
                                          [Compile]
    Project folder name                   [Run executable]
    Chess_Engine
                   [Execute setting]
```

.

Fig.5.1n.1 Layout of the ProjectSettingPanel

This class provides a GUI for user to setup the project location, project type, project name, compilation and execution of project, setting the location of the JDK and font of the code editor and the console.



```
▼ ProjectSettingPanel
    ▼ EventCallback
        onSetJDKLocation(String): boolean
        onRunGameExecutable(Component, String, Font, Dimension): void
        onCompileProject(String, AsyncCompilationCallBack): void
    ▶ TemplateType
    ▼ GameType
        ADD_TYPE: GameType
        MOVE_TYPE: GameType
        ADD_MOVE_TYPE: GameType
    ▼ FontDropDownMenuRenderer
        getListCellRendererComponent(JList<?>, Object, int, boolean, boolean):
    ProjectSettingPanel(EventCallback)
    setupLayout(): void
    registerEventListener(): void
    setupOtherSettingPanel(): void
    setupProjectSettingTitle(): void
    setupNewProjectPanel(): void
    setupProjectHandleWayPanel(): void
    setupLocationPanel(): void
    addToBasePanel(Component, int, int): void
    addToBasePanel(Component): void
    getDropDownMenuItemListener(JComboBox): ItemListener
    getButtonActionListener(JButton): ActionListener
    getBrowseButtonActionListener(JButton): ActionListener
    getTextFieldDocumentFilter(JTextField): DocumentFilter
    getRadioButtonsActionListener(JRadioButton): ActionListener
    getTextFieldDocumentLister(JTextField): DocumentListener
    getProjectLocationBaseDir(): String
    getProjectFolderName(): String
    createSpaceLabel(): JLabel
    setJDKLocation(String): void
    executeProjectSettings(): void
    isFolderNameValid(String): boolean
    compileProjectFinished(boolean): void
    setFontOfCodeEditors(String): boolean
    setFontSizeOfCodeEditors(int): void
    setGameType(GameType): void
    getGameTypeSelected(): GameType
```

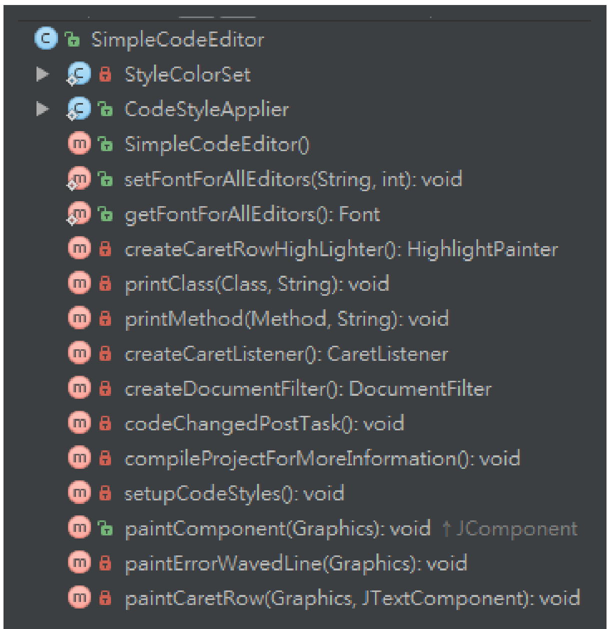
Fig. 5.1n.2 inner class and methods of ProjectSettingPanel

- Eventcall back class as the UIHandler to send the compile, execute and JDK location setting request to the Project compiler.
- setup[Name]() methods, addToBasePanel() is responsible to setup the layout
- get[GUI component]Listener() and get[GUI component]Filter() setup the event handlers for the radio buttons, text fields, drop down menu and buttons.
- The remaining methods do the system functions that stated in the

introduction about the class.

UIModule / SimpleCodeEditor

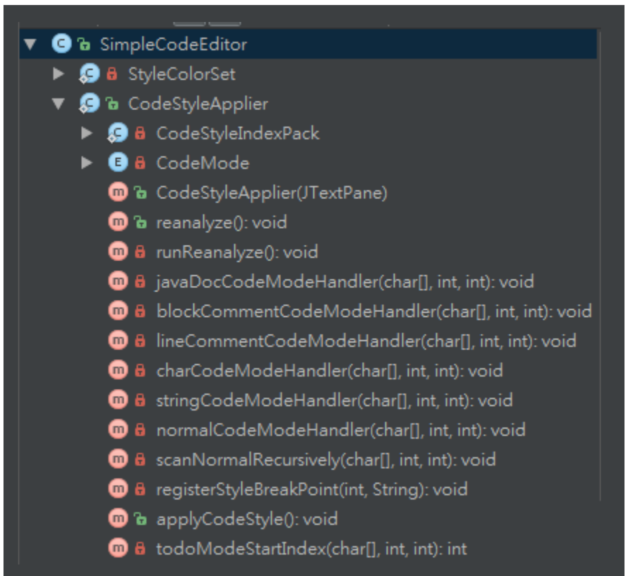
The layout the simple code editor is shown in figure 5.1m.1. This class aims to provide part of the functions of an IDE; they are Java keyword highlighting, code formatting, coding error highlighting, coding assistant and coding recommendation. Due to technical difficulties and time limitation, only the Java reserved keyword, numbers, Java Doc, block comment, line comment, char assignment and string assignment.



```
SimpleCodeEditor
▶ StyleColorSet
▶ CodeStyleApplier
  SimpleCodeEditor()
  setFontForAllEditors(String, int): void
  getFontForAllEditors(): Font
  createCaretRowHighLighter(): HighlightPainter
  printClass(Class, String): void
  printMethod(Method, String): void
  createCaretListener(): CaretListener
  createDocumentFilter(): DocumentFilter
  codeChangedPostTask(): void
  compileProjectForMoreInformation(): void
  setupCodeStyles(): void
  paintComponent(Graphics): void ↑JComponent
  paintErrorWavedLine(Graphics): void
  paintCaretRow(Graphics, JTextComponent): void
```

Fig.5.1o.1 inner class and methods of the SimpleCodeEditor

- There is a static list of SimpleCodeEditor, static methods that change the font and size for all the SimpleCodeEditor existing in the Chreator.
- CreateCaretListener(), createDocumentFilter() and setupCodeStyles() setup the handler monitoring the change of caret and content of the code editor and the styles of the code.
- createCaretRowHighLighter(), paintComponent(), paintErrorWavedLine() and paintCaretRow() are responsible for the painting of the caret row and coding area that containing error.
- codeChangedPostTask() asks for the CodeStyleApplier to rescan the code and highlight the characteristic of the code

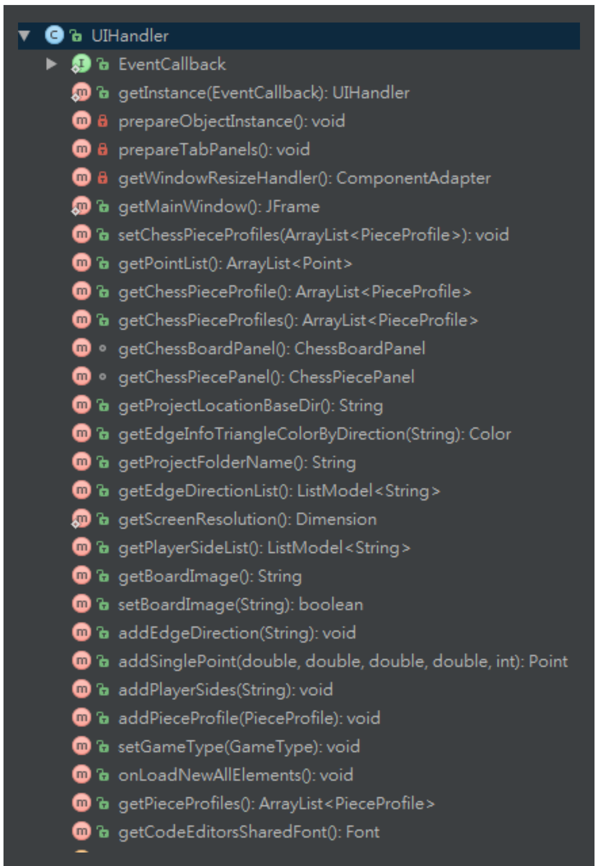


```
SimpleCodeEditor
  StyleColorSet
  CodeStyleApplier
    CodeStyleIndexPack
    CodeMode
    CodeStyleApplier(JTextPane)
    reanalyze(): void
    runReanalyze(): void
    javaDocCodeModeHandler(char[], int, int): void
    blockCommentCodeModeHandler(char[], int, int): void
    lineCommentCodeModeHandler(char[], int, int): void
    charCodeModeHandler(char[], int, int): void
    stringCodeModeHandler(char[], int, int): void
    normalCodeModeHandler(char[], int, int): void
    scanNormalRecursively(char[], int, int): void
    registerStyleBreakPoint(int, String): void
    applyCodeStyle(): void
    todoModeStartIndex(char[], int, int): int
```

Fig.5.1o.2 methods of the CodeStyleApplier inside the SimpleCodeEditor

The CodeStyleApplier start a new thread for analyzing the code asynchronously. It swtiches through different Code modes such as normal, string, char, block comment, line comment Java Doc and so on. It uses the Java keyword, Abstract String scanning methods provided in the UIUtility to find out all the Java keywords.

UIModule / UIHandler



```
UIHandler
  EventCallback
  getInstance(EventCallback): UIHandler
  prepareObjectInstance(): void
  prepareTabPanels(): void
  getWindowResizeHandler(): ComponentAdapter
  getMainWindow(): JFrame
  setChessPieceProfiles(ArrayList<PieceProfile>): void
  getPointList(): ArrayList<Point>
  getChessPieceProfile(): ArrayList<PieceProfile>
  getChessPieceProfiles(): ArrayList<PieceProfile>
  getChessBoardPanel(): ChessBoardPanel
  getChessPiecePanel(): ChessPiecePanel
  getProjectLocationBaseDir(): String
  getEdgeInfoTriangleColorByDirection(String): Color
  getProjectFolderName(): String
  getEdgeDirectionList(): ListModel<String>
  getScreenResolution(): Dimension
  getPlayerSideList(): ListModel<String>
  getBoardImage(): String
  setBoardImage(String): boolean
  addEdgeDirection(String): void
  addSinglePoint(double, double, double, double, int): Point
  addPlayerSides(String): void
  addPieceProfile(PieceProfile): void
  setGameType(GameType): void
  onLoadNewAllElements(): void
  getPieceProfiles(): ArrayList<PieceProfile>
  getCodeEditorsSharedFont(): Font
```

Fig.5.1p methods of the UIHandler

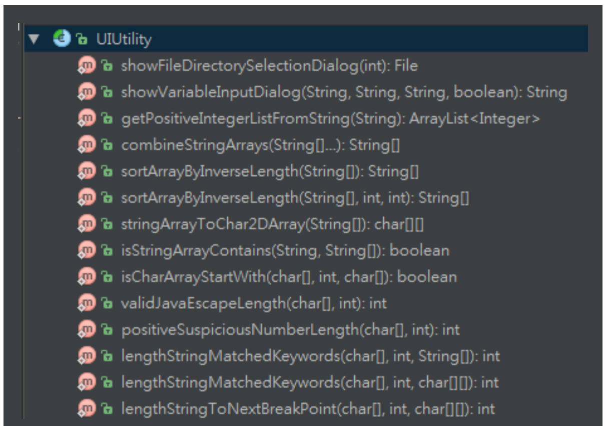
The UIHandler is designed to be the bridge of all the Panels to the outside world and the inter-communication between the panels.

- the prepare[Name]() functions setup the main windows and prepare the panels of the Chreator. They finally add 5 tab pages in the Chreator, the Project Setting page, Chess Board page, Chess Piece page, Game Rule page and AI page.
- the get[Name]() functions return

the panels or data from the corresponding panels for panels or other classes outside, such as the ProjectCompiler, CodeProducer and so on.

- The add[Name]() and set[Name]() functions are used to apply the data to the Chreator from the outside of the UIModule or the other panels within the UIModule.

UIModule / UIUtility



```
UIUtility
    showFileDirectorySelectionDialog(int): File
    showVariableInputDialog(String, String, String, boolean): String
    getPositiveIntegerListFromString(String): ArrayList<Integer>
    combineStringArrays(String[]...): String[]
    sortArrayByInverseLength(String[]): String[]
    sortArrayByInverseLength(String[], int, int): String[]
    stringArrayToChar2DArray(String[]): char[][]
    isStringArrayContains(String, String[]): boolean
    isCharArrayStartWith(char[], int, char[]): boolean
    validJavaEscapeLength(char[], int): int
    positiveSuspiciousNumberLength(char[], int): int
    lengthStringMatchedKeywords(char[], int, String[]): int
    lengthStringMatchedKeywords(char[], int, char[][]): int
    lengthStringToNextBreakPoint(char[], int, char[][]): int
```

Fig.5.1q methods of the UIUtility

The UIUtlity is an abstract class that does not suppose anybody to create an instance of the class.

This class provides the method to call the prompt up input dialogs, message dialogs, option dialogs and so on. Besides, this class provide the char array, String, char double array and String array sorting, combining and content matching functions that are used by finding and filtering out the Java keywords and Java numbers in the SimpleCodeEditor.

ChreatorLauncher

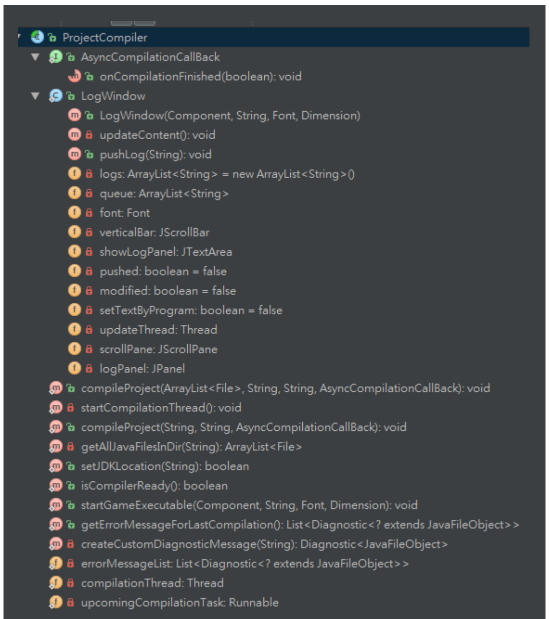
ChreatorLauncher is the main function of the program, it contains an public static void main(String[] args) function for the JVM to start the program. It is supposed to be the bridge of all the Modules inside the Chreator, where Module-to-Module communication should be done through it. Below is part of the coding captured.

```
public class ChreatorLauncher {

    private UIHandler ui;

    private String sourceFolder = "src", binaryFolder = "bin";

    public static void main(String[] args) {new ChreatorLauncher();}

    public ChreatorLauncher() {ui = UIHandler.getInstance(getUIEventCallback());

    private UIHandler.EventCallback getUIEventCallback() {

        return new UIHandler.EventCallback() {

            public boolean onSetJDKLocation(String JDKLocation) {

                boolean result = ProjectCompiler.setJDKLocation(JDKLocation);

                …

            }

            public void onRunGameExecutable(…) {

                ProjectCompiler.startGameExecutable(…);

            }

            public void onCompileProject(…) {

                ProjectCompiler.compileProject(…);

            }

        };

    }

}
```

Fig. 5.1r Part of the source code of ChreatorLauncher

We can see that the ChreatorLauncher creates the UIHandler, it passes the request from the UIHandler to the other classes, and return the result to the UIHandler. Figure 5.1r shows the mechanism that UIHandler communicating with the ProjectCompiler. The 3 functional communication – setting JDK location of the JVM; compile the chess game project and try to execute the ches game, between the UIModule and ProjectCompile rely on the ChreatorLauncher.

<u>ProjectCompiler</u>



```
ProjectCompiler
  AsyncCompilationCallBack
      onCompilationFinished(boolean): void
  LogWindow
      LogWindow(Component, String, Font, Dimension)
      updateContent(): void
      pushLog(String): void
      logs: ArrayList<String> = new ArrayList<String>()
      queue: ArrayList<String>
      font: Font
      verticalBar: JScrollBar
      showLogPanel: JTextArea
      pushed: boolean = false
      modified: boolean = false
      setTextByProgram: boolean = false
      updateThread: Thread
      scrollPane: JScrollPane
      logPanel: JPanel
  compileProject(ArrayList<File>, String, String, AsyncCompilationCallBack): void
  startCompilationThread(): void
  compileProject(String, String, AsyncCompilationCallBack): void
  getAllJavaFilesInDir(String): ArrayList<File>
  setJDKLocation(String): boolean
  isCompilerReady(): boolean
  startGameExecutable(Component, String, Font, Dimension): void
  getErrorMessageForLastCompilation(): List<Diagnostic<? extends JavaFileObject>>
  createCustomDiagnosticMessage(String): Diagnostic<JavaFileObject>
  errorMessageList: List<Diagnostic<? extends JavaFileObject>>
  compilationThread: Thread
  upcomingCompilationTask: Runnable
```

Fig.5.1s full hierarchy of the ProjectCompiler

- AsyncCompilationCallBack is an interface class with a callback function that allows the program to go back to the original call-in function and execute the remaining code after the compilation is finished. It is because compiling a big Java source project may take seconds to finish. If the compilation is done in the same thread of the UI, the UI may freeze for seconds. This is a bad user experience.

Therefore, a new thread must be used to compile the project independently.

- The LogWindow class is a class showing a GUI window for showing messages just like a terminal in Linux or command prompt in Windows.
- List<Diagnostic<? extends JavaFileObject>> errorMessageList: the JavaCompiler return an error message list of Diagnostic of JavaFileObject after the compilation. The Diagnostic tells you which position at which line of which java file contains what kind of error. Specific information about the compilation

error can be retrieved by the API of Diagnostic. To be reminded that the JavaCompiler, Diagnostic and JavaFileObject are built in classes or interfaces of Java in the javax.tools package.

- Thread compilationThread and Runnable upcomingCompilationTask are the single running compilation task in the ProjectCompiler class.
- void compileProject(ArrayList<File> files, final String inDirPath, final String outDirPath, final AsyncCompilationCallBack callBack): the main compilation starter method. It makes a new compilation task and starts the compilation thread if the thread is not alive. If the compilation thread find that there is a new compilation task after the current compilation task, the compilation thread will continue to compile the latest compilation task until there is no compilation task anymore. Then the compilation thread finishes.
- To compile java source code using java language, the following code are applied:

```
ArrayList<File> filesNeedToBeCompiled;
String outDirPath;
JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
// obtain the java compiler
DiagnosticCollector<JavaFileObject> diagnostics = new
DiagnosticCollector<JavaFileObject>();
// build the diagnostic object for the debug use
StandardJavaFileManager fileManager =
compiler.getStandardFileManager(diagnostics, null, null);
// obtain the java file manager to that general file can be treated as java source
code file
Iterable<? extends JavaFileObject> fileObjects =
fileManager.getJavaFileObjectsFromFiles(filesNeedToBeCompiled);
// use the java file manager to convert the general file to the java source code
file
Iterable<String> options = Arrays.asList("-d", outDirPath);
// specify the compilation argument, here the output path is specified
boolean result = compiler.getTask(null, fileManager, diagnostics, options, null,
fileObjects).call();
```

```
// call the compiler to compile the source code
errorMessageList = diagnostics.getDiagnostics();
// backup the compilation error information, that is the diagnostics
```

- Method getAllJavaFilesInDir() collect all the files with extension *.java recursively under a folder.
- To set the JDK location in a Java application by language, the following command can be used:

```
System.setProperty("java.home", someLocation + "/jre");
// someLocation can be “C:\Program Files\Java\jdk1.8.0_74”
```

- The method startGameExecutable() start the compile chess game executable. To start a process in Java language, the code is:

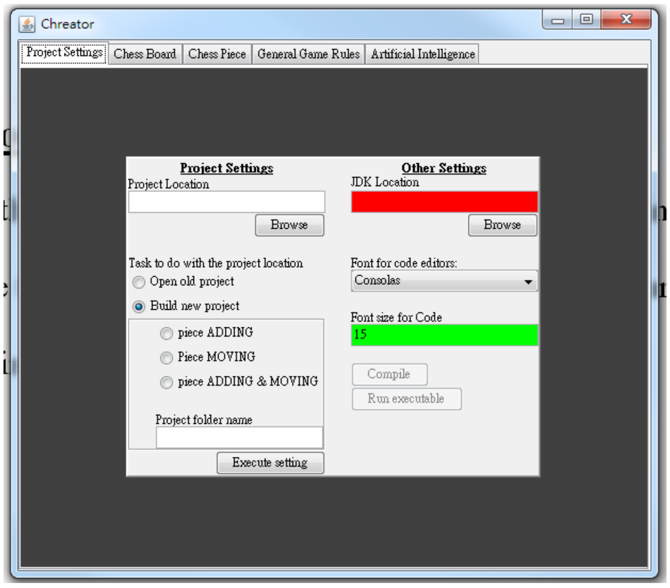
```
Process process = new ProcessBuilder("java", "-cp", gameBinDirectory,
"Executable.Game").start();
```

- After the process is start, the parent process must start 2 threads to consume the input stream and error stream messages pushed from the child process, otherwise the child process may hang up:

```
BufferedReader in =
    new BufferedReader(new InputStreamReader(process.getInputStream()));
try {
    while ((String line = in.readLine()) != null){…}
} catch (Exception e) {}
```

System design and usage of the entire Chreator

1 When the Chreator starts, there are 5 tabs shown: They are project setting tab, chess board tabs chess piece tab, general game rules tab, and AI tab. The currently showing page is project settings:


Chreator

Project Settings | Chess Board | Chess Piece | General Game Rules | Artificial Intelligence

Project Settings

Project Location:

Task to do with the project location: ( ) Open old project [✓ Build new project]
- ( ) piece ADDING
- ( ) Piece MOVING
- ( ) piece ADDING & MOVING

Project folder name:

Other Settings

JDK Location: 🟥

Font for code editors: Consolas

Font size for Code: 15

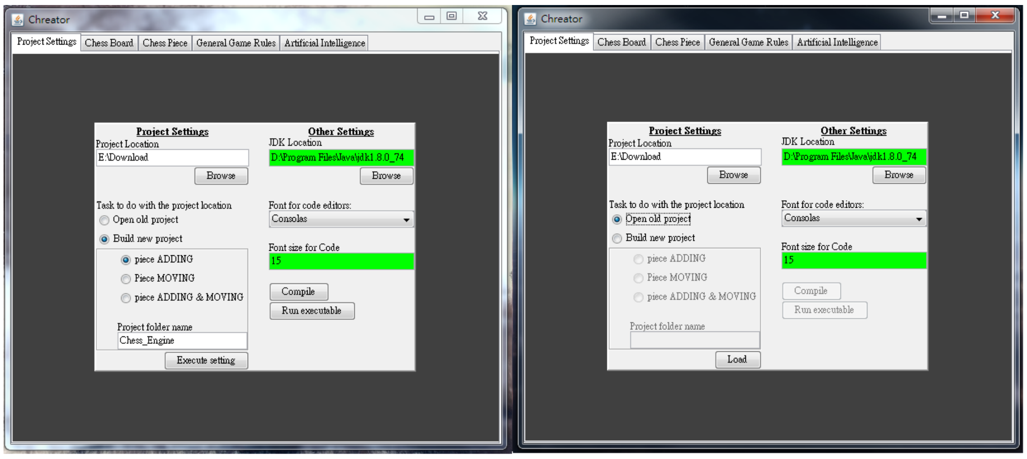
Fig. 5.1t.1 The project settings page

JDK location fields with red color means that the Chreator cannot find the Java compiler; green color means that the Java compiler is Ready. Similarly, font size for Code field with green color means that the input value is valid.

At first, the user has to provide the directory path. If the user chooses the open the old project, the path must contain a chess game project; if the user chooses to save a project, the project will be saved to the provided path. If it is optional to fill in a project folder name for saving a project / building a new project.

The JDK location is optional to be provided. It is only necessary if the user would like to use the Chreator to compile and run the chess game project.

The following figure provide a clearer concept.


Chreator

Project Settings | Chess Board | Chess Piece | General Game Rules | Artificial Intelligence

Project Settings
Project Location: E:\Download
Task to do with the project location: Open old project [✓ Build new project]
[✓ piece ADDING] Piece MOVING piece ADDING & MOVING
Project folder name: Chess_Engine

Other Settings
JDK Location: D:\Program Files\Java\jdk1.8.0_74
Font for code editors: Consolas
Font size for Code: 15

Chreator

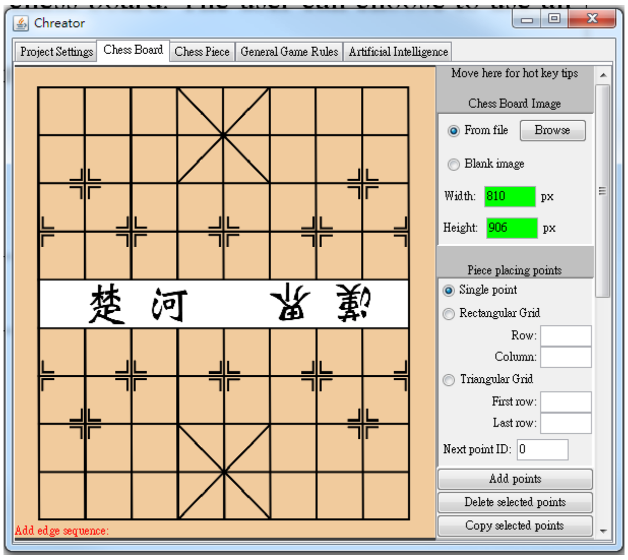
Project Settings
Project Location: E:\Download
Task to do with the project location: [✓ Open old project] Build new project
piece ADDING Piece MOVING piece ADDING & MOVING
Project folder name:

Other Settings
JDK Location: D:\Program Files\Java\jdk1.8.0_74
Font for code editors: Consolas
Font size for Code: 15

Fig. 5.1t.2 Both the Chreator programs provided a correct project location, a valid path to the JDK. The left Chreator program would like to save the project to the destination, while the right Chreator would like to load the project from the destination folder.

Execute setting or Load button is pressed, the action will be done. Now it is possible to press the compile button to compile the project. Pressing Run executable button to play the game.

2 Now the Chreator is ready to make the chess board. The user can choose to use an image or left the chess board blank. The size of the chess board is free to adjust Here is the example of using a chess board of XiangQi.


Chreator

Project Settings | Chess Board | Chess Piece | General Game Rules | Artificial Intelligence

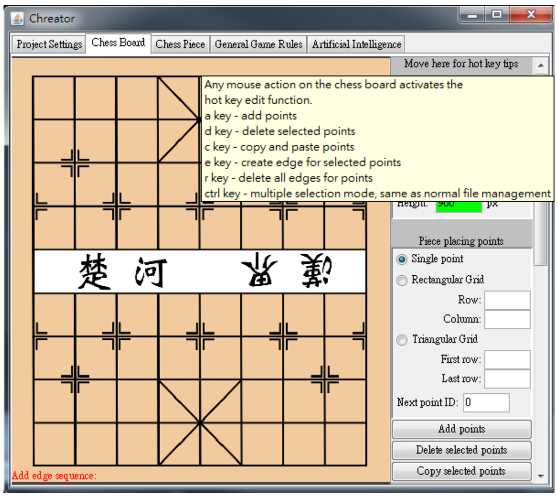
Chess Board Image: [✓ From file] Blank image
Width: 810 px
Height: 906 px

Piece placing points: [✓ Single point] Rectangular Grid Triangular Grid
Row:
Column:
First row:
Last row:
Next point ID: 0

Add edge sequence:

Fig. 5.1t.3 design with XiangQi Chess Board.

The user can now add the piece placing points to the chess board. User can choose to add single point, a rectangular grid of points or a triangular grid of points. Users can use the buttons in the “piece placing points” area to add the points, delete selected points and copy selected points. Or the user can click once on the chess board and use the hot key on the key board.



Any mouse action on the chess board activates the hot key edit function.
a key - add points
d key - delete selected points
c key - copy and paste points
e key - create edge for selected points
r key - delete all edges for points
ctrl key - multiple selection mode, same as normal file management

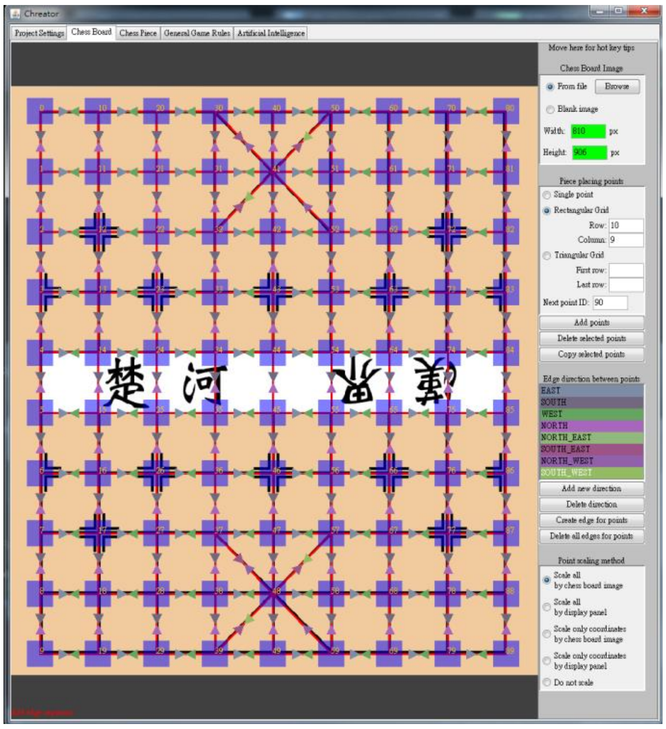
Height: 900 px

Piece placing points
(●) Single point
( ) Rectangular Grid
Row:
Column:
( ) Triangular Grid
First row:
Last row:
Next point ID: 0

Add edge sequence:

Fig. 5.1t.4 Hot key tips appear with the cursor is moved to the “move her for hot key tips”

To build the relationships between the points, there must be edge directions. After the edge directions between points are registered, the points can add edges, below is the finished chess board of XiangQi:



Chess Board Image
(●) From file
( ) Blank image
Width: 810 px
Height: 906 px

Piece placing points
( ) Single point
(●) Rectangular Grid
Row: 10
Column: 9
( ) Triangular Grid
First row:
Last row:
Next point ID: 90

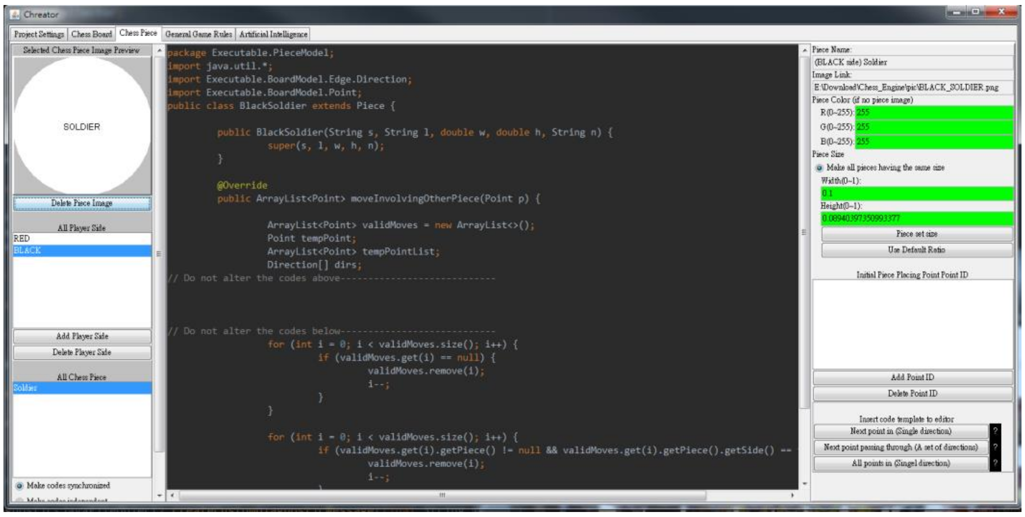
Edge direction between points
EAST
SOUTH
WEST
NORTH
NORTH_EAST
SOUTH_EAST
NORTH_WEST
SOUTH_WEST

Point scaling method
(●) Scale all by chess board image
( ) Scale all by display panel
( ) Scale only coordinates by chess board image
( ) Scale only coordinates by display panel
( ) Do not scale

Fig. 5.1t.5 A finished chess board of XiangQi.

After this step is reached, it is alright to go to the chess piece page to define the chess pieces.

3 To make a chess piece, there must be at least 1 player side. The below example add a RED side and a BLACK side, a piece with name Soldier. Select the soldier profile, the basic code, default size and color of the chess piece is made.

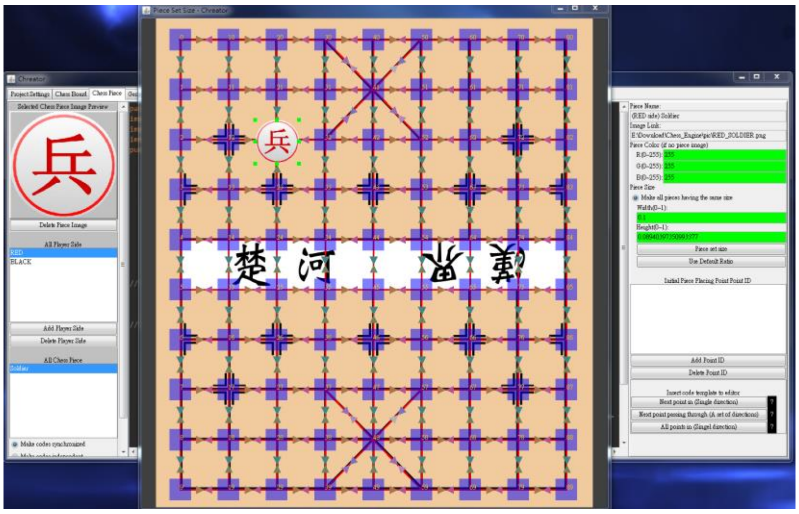


```
package Executable.PieceModel;
import java.util.*;
import Executable.BoardModel.Edge.Direction;
import Executable.BoardModel.Point;
public class BlackSoldier extends Piece {

        public BlackSoldier(String s, String l, double w, double h, String n) {
                super(s, l, w, h, n);
        }

        @Override
        public ArrayList<Point> moveInvolvingOtherPiece(Point p) {

                ArrayList<Point> validMoves = new ArrayList<>();
                Point tempPoint;
                ArrayList<Point> tempPointList;
                Direction[] dirs;
// Do not alter the codes above---------------------------



// Do not alter the codes below---------------------------
                for (int i = 0; i < validMoves.size(); i++) {
                        if (validMoves.get(i) == null) {
                                validMoves.remove(i);
                                i--;
                        }
                }

                for (int i = 0; i < validMoves.size(); i++) {
                        if (validMoves.get(i).getPiece() != null && validMoves.get(i).getPiece().getSide() ==
                                validMoves.remove(i);
                                i--;
```

Piece Name: (BLACK side) Soldier
Image Link: E:\Download\Chess_Engine\pic\BLACK_SOLDIER.png
Piece Color (if no piece image)
R(0~255): 255
G(0~255): 255
B(0~255): 255
Piece Size
Make all pieces having the same size
Width(0~1): 0.1
Height(0~1): 0.0894039735099337

Fig.5.1t.6 fresh build of a chess piece.

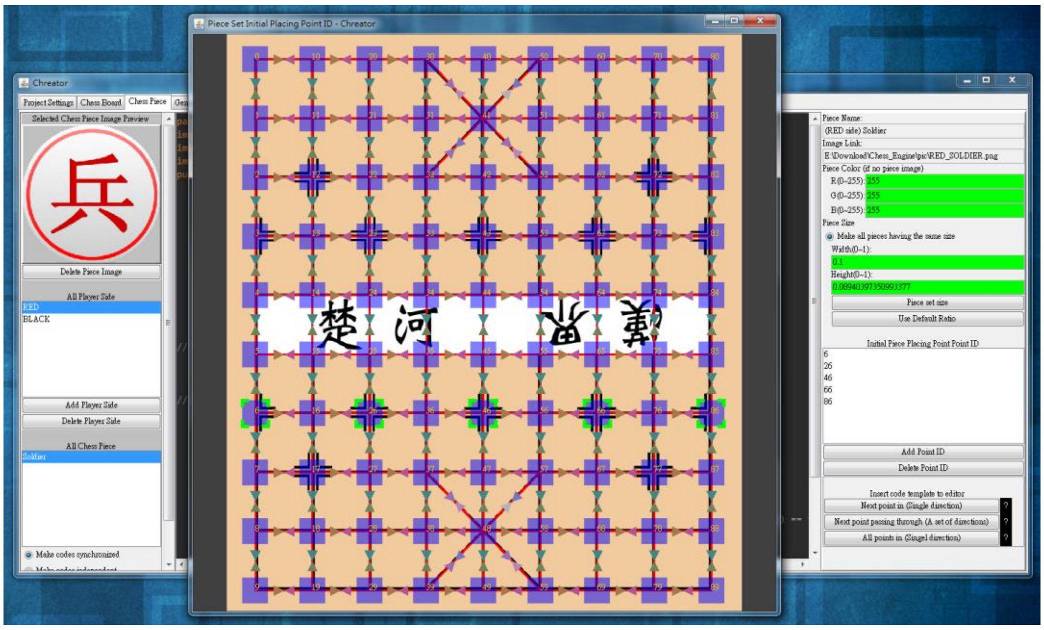
The chess piece can be left with plain color or use an image. Also the ratio and size of the chess piece can be adjusted by GUI by pressing the Piece set size button. Below shows the set size layout for a red solider of a XiangQi.



Piece Name: (RED side) Soldier
Image Link: E:\Download\Chess_Engine\pic\RED_SOLDIER.png
R(0~255): 255
G(0~255): 255
B(0~255): 255
Width(0~1): 0.1
Height(0~1): 0.0894039735099337

Fig. 5.1t.7 setting size and ratio for a red soldier of a XiangQi

To set the location of the chess piece when the game starts, user can use the Add point ID button under the Initial Piece Placing Point Point ID Area. User can type the series of point IDs if user chooses to input the point ID through dialog. Otherwise user can choose to select the points by GUI, below is an example:



Piece Set Initial Placing Point ID - Chreator
Chreator
Project Settings
Chess Board
Chess Piece
Selected Chess Piece Image Preview
Delete Piece Image
All Player Side
Add Player Side
Delete Player Side
All Chess Piece
Make codes synchronized
Piece set size
Use Default Ratio
Add Point ID
Delete Point ID
Insert code template to editor
Next point in (Single direction)
Next point passing through (A set of directions)
All points in (Singel direction)

RED
BLACK

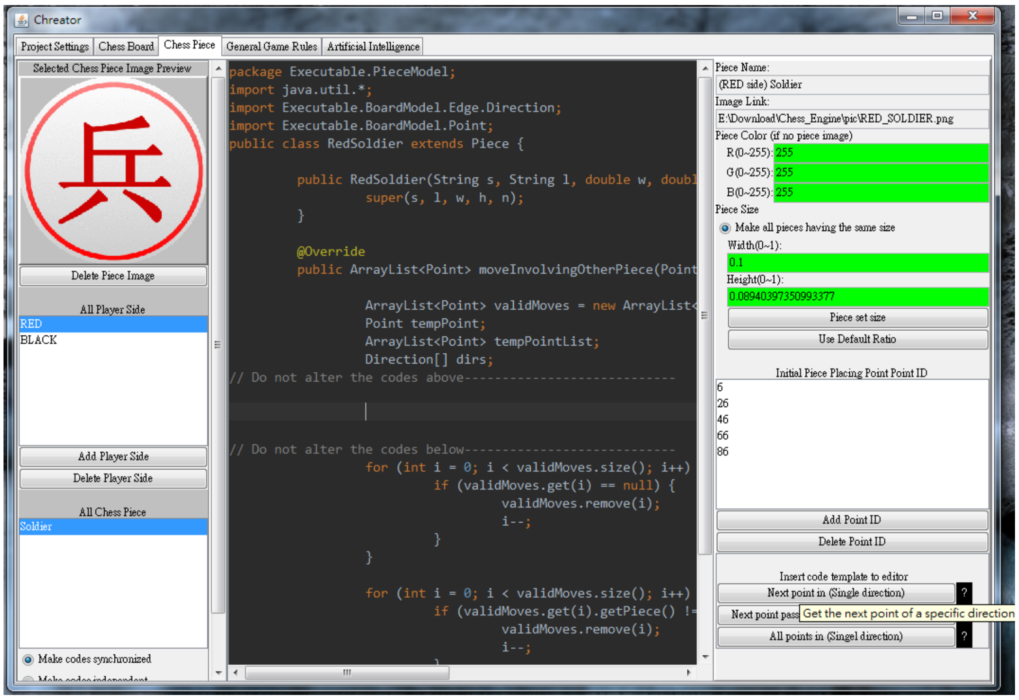
Soldier

Piece Name:
(RED side) Soldier
Image Link:
E:\Download\Chess_Engine\pic\RED_SOLDIER.png
Piece Color (if no piece image)
R(0~255): 255
G(0~255): 255
B(0~255): 255
Piece Size
Make all pieces having the same size
Width(0~1):
0.1
Height(0~1):
0.08940397350993377

Initial Piece Placing Point Point ID
6
26
46
66
86

Fig. 5.1t.8 setting initial piece placing point point ID for a red soldier of a XiangQi

To define the moving rules of the chess piece, user need to type codes within the location specified. There is a “Insert code template to editor” area:



RED
BLACK

Soldier

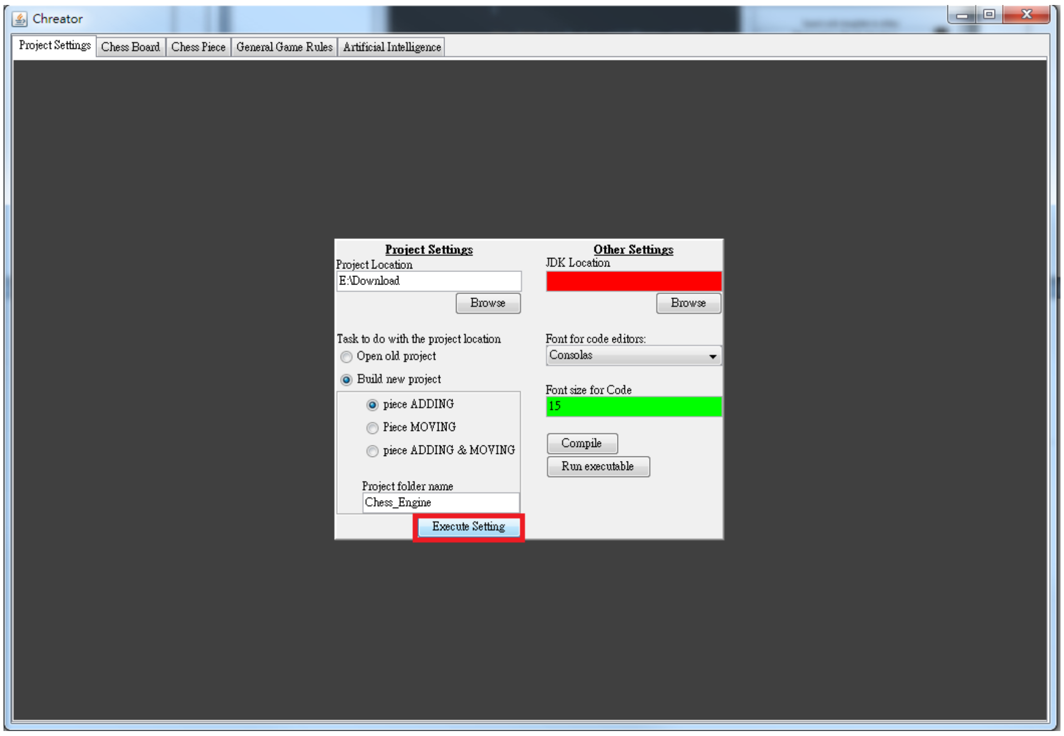
```
package Executable.PieceModel;
import java.util.*;
import Executable.BoardModel.Edge.Direction;
import Executable.BoardModel.Point;
public class RedSoldier extends Piece {

        public RedSoldier(String s, String l, double w, doubl
                super(s, l, w, h, n);
        }

        @Override
        public ArrayList<Point> moveInvolvingOtherPiece(Point

                ArrayList<Point> validMoves = new ArrayList<
                Point tempPoint;
                ArrayList<Point> tempPointList;
                Direction[] dirs;
// Do not alter the codes above---------------------------



// Do not alter the codes below---------------------------
                for (int i = 0; i < validMoves.size(); i++)
                        if (validMoves.get(i) == null) {
                                validMoves.remove(i);
                                i--;
                        }
                }

                for (int i = 0; i < validMoves.size(); i++)
                        if (validMoves.get(i).getPiece() !=
                                validMoves.remove(i);
                                i--;
```

Piece Name:
(RED side) Soldier
Image Link:
E:\Download\Chess_Engine\pic\RED_SOLDIER.png
Piece Color (if no piece image)
R(0~255): 255
G(0~255): 255
B(0~255): 255
Piece Size
Make all pieces having the same size
Width(0~1):
0.1
Height(0~1):
0.08940397350993377

Initial Piece Placing Point Point ID
6
26
46
66
86

Fig.5.1t.9 The caret is within the valid location. The tool tip is showing the usage of the code to be inserted to the code editor.

Once the definition is done, switching back to the project setting page and press execute setting again to save the project:



Project Settings

Project Location: E:\Download

Task to do with the project location
- Open old project
- [✓ Build new project]
  - [✓ piece ADDING]
  - Piece MOVING
  - piece ADDING & MOVING

Project folder name: Chess_Engine

Other Settings

JDK Location: 🟥

Font for code editors: Consolas

Font size for Code: 15

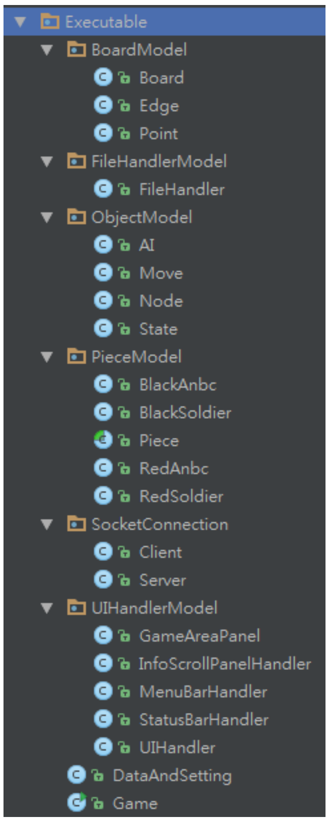
Fig. 5.1t.10 save the project by pressing the execute setting button but never change the other settings.

The discussion until now has covered most of the implemented function of the Chreator. In the next part, the basic system architecture and the brief system design and usage of the Chess Game Executable will be discussed.

Part 2: Chess Game Executable

<u>System architecture of the entire chess game executable</u>

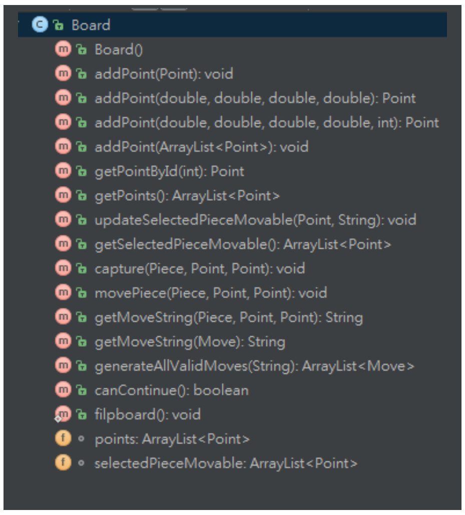
The structure of the Chess Game Executable is far less complicated than that of the Chreator. The entire structure of the Chess Game Executable is as shown as follow:



```
▼ Executable
  ▼ BoardModel
      Board
      Edge
      Point
  ▼ FileHandlerModel
      FileHandler
  ▼ ObjectModel
      AI
      Move
      Node
      State
  ▼ PieceModel
      BlackAnbc
      BlackSoldier
      Piece
      RedAnbc
      RedSoldier
  ▼ SocketConnection
      Client
      Server
  ▼ UIHandlerModel
      GameAreaPanel
      InfoScrollPanelHandler
      MenuBarHandler
      StatusBarHandler
      UIHandler
    DataAndSetting
    Game
```

Fig. 5.2a entire structure of the chess game executable

<u>Executable / BoardModel / Board</u>



```
Board
  Board()
  addPoint(Point): void
  addPoint(double, double, double, double): Point
  addPoint(double, double, double, double, int): Point
  addPoint(ArrayList<Point>): void
  getPointById(int): Point
  getPoints(): ArrayList<Point>
  updateSelectedPieceMovable(Point, String): void
  getSelectedPieceMovable(): ArrayList<Point>
  capture(Piece, Point, Point): void
  movePiece(Piece, Point, Point): void
  getMoveString(Piece, Point, Point): String
  getMoveString(Move): String
  generateAllValidMoves(String): ArrayList<Move>
  canContinue(): boolean
  filpboard(): void
  points: ArrayList<Point>
  selectedPieceMovable: ArrayList<Point>
```

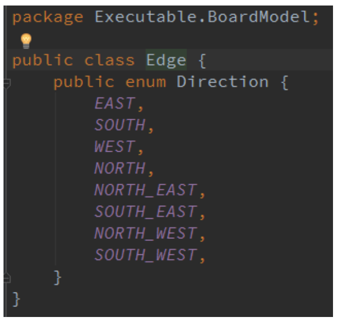
The Board Model represents the chess board. A chess board contains all the piece placing points. Therefore, when the chess game executable starts, the defined chess piece placing points will be added to the board by using addPoint(). All the size and coordinates of the chess piece placing points are represented in relative. For the meaning

Fig.5.2b full hierarchy of the Board

of relative size and relative coordinates, please see the explanation of the Point class of the Chreator. The board also returns the chess piece placing point by given the ID of the point by using the getPoint() method. The capture() and movePiece() methods are the save way to piece a piece from point to point; updateSelectedPieceMoable() and generateAllValidMoves() functions return the all reachable points.

Executable / BoardModel / Edge

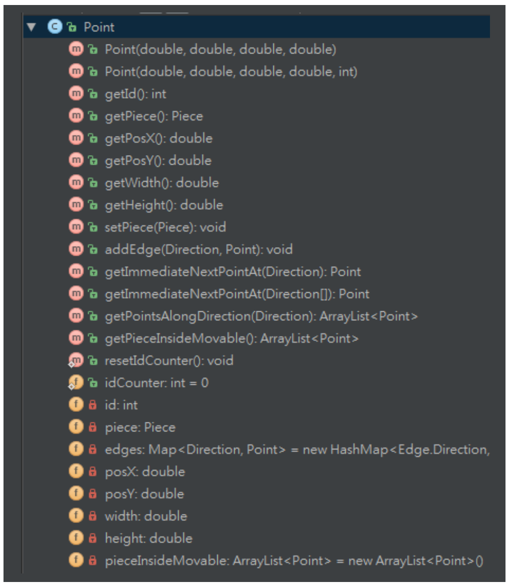
This class only contains the edge direction registered in the Chreator. Here is the content of the Edge class for a XiangQi game as an example.



```java
package Executable.BoardModel;

public class Edge {
    public enum Direction {
        EAST,
        SOUTH,
        WEST,
        NORTH,
        NORTH_EAST,
        SOUTH_EAST,
        NORTH_WEST,
        SOUTH_WEST,
    }
}
```

Fig.5.2c full class of Edge for a XiangQi game.

Executable / BoardModel / Point



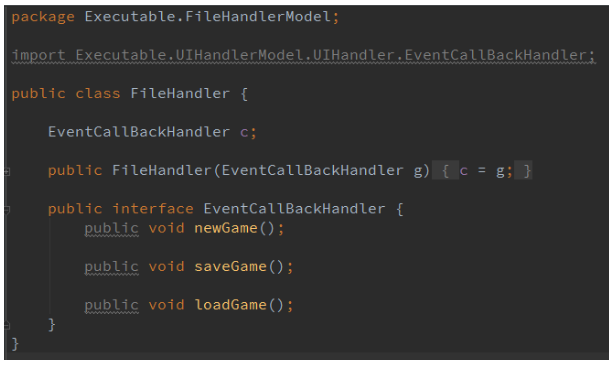
```
▼ Point
    Point(double, double, double, double)
    Point(double, double, double, double, int)
    getId(): int
    getPiece(): Piece
    getPosX(): double
    getPosY(): double
    getWidth(): double
    getHeight(): double
    setPiece(Piece): void
    addEdge(Direction, Point): void
    getImmediateNextPointAt(Direction): Point
    getImmediateNextPointAt(Direction[]): Point
    getPointsAlongDirection(Direction): ArrayList<Point>
    getPieceInsideMovable(): ArrayList<Point>
    resetIdCounter(): void
    idCounter: int = 0
    id: int
    piece: Piece
    edges: Map<Direction, Point> = new HashMap<Edge.Direction,
    posX: double
    posY: double
    width: double
    height: double
    pieceInsideMovable: ArrayList<Point> = new ArrayList<Point>()
```

The class represents the chess piece placing point similar to that of the Chreator. Instead, it contains the ID of the point, relative size and coordinates and a hash map containing the neighbor points with the direction. It has methods that finding all the reachable points and so on.

Fig.5.2d full hierarchy of the Board

Executable / FileHandlerModel / FileHandler

This class is an interface that force the Game class to handle “new game”, “save game” and “load game” function.



```
package Executable.FileHandlerModel;

import Executable.UIHandlerModel.UIHandler.EventCallBackHandler;

public class FileHandler {

    EventCallBackHandler c;

    public FileHandler(EventCallBackHandler g) { c = g; }

    public interface EventCallBackHandler {
        public void newGame();

        public void saveGame();

        public void loadGame();
    }
}
```

Fig. 5.2e: The entire class of FileHandler.

Executable / ObjectModel / AI

This class is originally to be implemented by the user. Since it is discussed in the interim presentation that there should be a decision determining function in the AI class but we failed to implement it due to time limitation, the AI determination class written by stepTNT[7] is found by Li Chun Ki is included as a demo.

The class contains a minMax() function that a typical game tree is adopted. The class also provides the getBestMove() and sortMoves() methods to find the most beneficial movement with the most efficient way as data sorting is used.

Executable / ObjectModel / Move

This class is the data exchange package class used among the calculation of AI,

[7] StepTNT, site: http://stackoverflow.com/questions/27527090/finding-the-best-move-using-minmax-with-alpha-beta-pruning

network playing class Server and Client, the Game class and the Board class. It carries the information about the chess movement action, including the ID of the source point, target point, chess piece and the piece name. This class can be considered as the moving action for a chess piece.

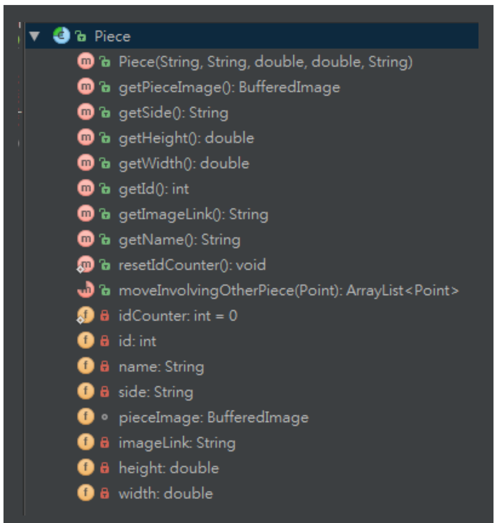
Executable / ObjectModel / Node

Node is a moving record for each step. It is used to store the record as history. It contains a state, a Move action object, the source point ID, target point ID, parent Node, depth and the path cost;

Executable / ObjectModel / State

The state uses a hash map to store all the pieceID and the pointID they are currently in. That means the chess board is backup. The side of the State is also recorded. This class is only be used in Node class.

Executable / PieceModel / Piece



```
▼ Piece
    Piece(String, String, double, double, String)
    getPieceImage(): BufferedImage
    getSide(): String
    getHeight(): double
    getWidth(): double
    getId(): int
    getImageLink(): String
    getName(): String
    resetIdCounter(): void
    moveInvolvingOtherPiece(Point): ArrayList<Point>
    idCounter: int = 0
    id: int
    name: String
    side: String
    pieceImage: BufferedImage
    imageLink: String
    height: double
    width: double
```

Fig.5.2f full hierarchy of the Piece

The abstract Piece class is a template for all the other actual piece classes. It defines the shared information for all the actual pieces. For example the piece image, name, relative height and relative width. All the get[Name]() functions retrieve the information of the piece. Function resetIdCounter() resets the counter for producing the unique ID of the chess piece.

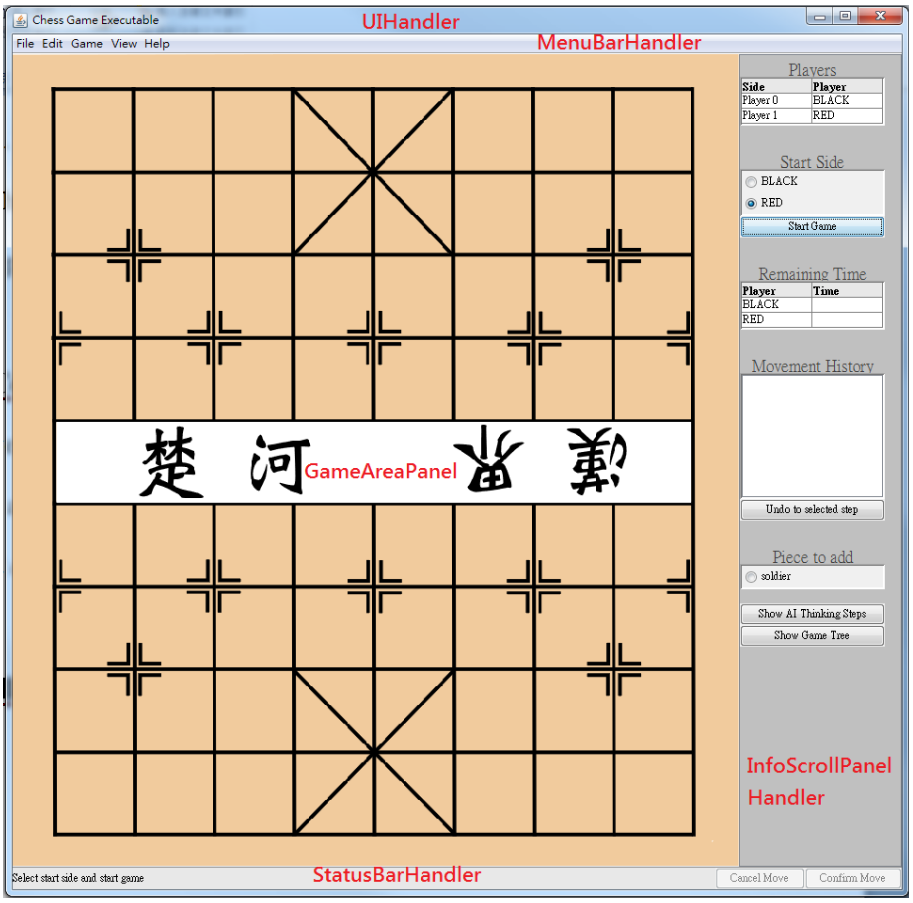
All the piece classes generated in the Executable / PieceModel by the Chreater extends this abstract class. Thus the abstract method moveInvolvingOtherPiece() must be implemented by all the actual piece classes. This method actually defines the moving rules of the chess piece itself.

Executable / SocketConnection

There are Client and Server under this packet. They are responsible for network chess playing. Both the client and server contain the sendMove() method. This method exchanges data between 2 chess game executables through network.

Executable / UIHandlerModel

There are 5 modules in the UIHandlerModel. They are UIHandler, GameAreaPanel, InfoScrollPanelHandler, MenuBarHandler and StatusBarHandler. Each module handles each part of the chess game window as follow:



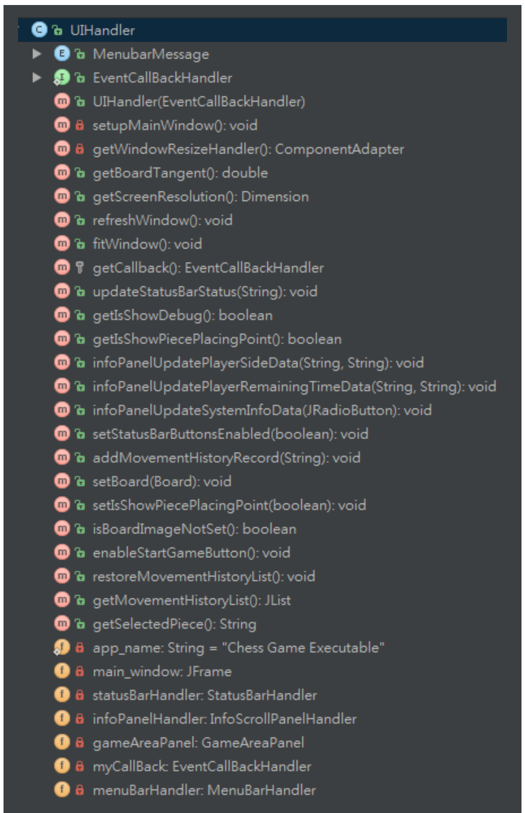
Side	Player
Player 0	BLACK
Player 1	RED

Player	Time
BLACK	
RED	

Select start side and start game

Fig. 5.2.g Control of each model to the UI of the chess game executable.

<u>Executable / UIHandlerModel / UIHandler</u>



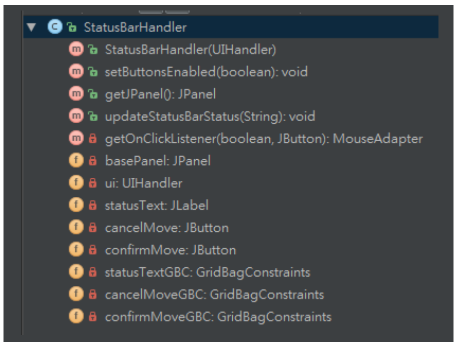
```
UIHandler
  MenubarMessage
  EventCallBackHandler
  UIHandler(EventCallBackHandler)
  setupMainWindow(): void
  getWindowResizeHandler(): ComponentAdapter
  getBoardTangent(): double
  getScreenResolution(): Dimension
  refreshWindow(): void
  fitWindow(): void
  getCallback(): EventCallBackHandler
  updateStatusBarStatus(String): void
  getIsShowDebug(): boolean
  getIsShowPiecePlacingPoint(): boolean
  infoPanelUpdatePlayerSideData(String, String): void
  infoPanelUpdatePlayerRemainingTimeData(String, String): void
  infoPanelUpdateSystemInfoData(JRadioButton): void
  setStatusBarButtonsEnabled(boolean): void
  addMovementHistoryRecord(String): void
  setBoard(Board): void
  setIsShowPiecePlacingPoint(boolean): void
  isBoardImageNotSet(): boolean
  enableStartGameButton(): void
  restoreMovementHistoryList(): void
  getMovementHistoryList(): JList
  getSelectedPiece(): String
  app_name: String = "Chess Game Executable"
  main_window: JFrame
  statusBarHandler: StatusBarHandler
  infoPanelHandler: InfoScrollPanelHandler
  gameAreaPanel: GameAreaPanel
  myCallBack: EventCallBackHandler
  menuBarHandler: MenuBarHandler
```

Fig.5.2h entire UIHandler class

The UIHandler use the similar approach to the UIHandler of the Chreator. It is the bridge of all the 4 panels inside the UIHandlerModel to the outer world, such that all the panels can exchange data through UIHandler.

- setupMainWindow() builds all the panel objects and setup the layout.
- get[Name]() functions collect the required information from the corresponding panel and return the result to the caller class.
- Set[Name]() and infoPanelUpdate[Name]() functions install the data to the corresponding panel.
- The remaining functions are most likely the utility functions. Such as refresh the main windows, return the screen resolution, recover the moving history of the chess game and so on.

<u>Executable / UIHandlerModel / StatusBarHandler</u>



```
StatusBarHandler
  StatusBarHandler(UIHandler)
  setButtonsEnabled(boolean): void
  getJPanel(): JPanel
  updateStatusBarStatus(String): void
  getOnClickListener(boolean, JButton): MouseAdapter
  basePanel: JPanel
  ui: UIHandler
  statusText: JLabel
  cancelMove: JButton
  confirmMove: JButton
  statusTextGBC: GridBagConstraints
  cancelMoveGBC: GridBagConstraints
  confirmMoveGBC: GridBagConstraints
```

- There are 2 buttons and a text label in the status bar
- Any mouse event happened will be reported to the UIHandler directly.

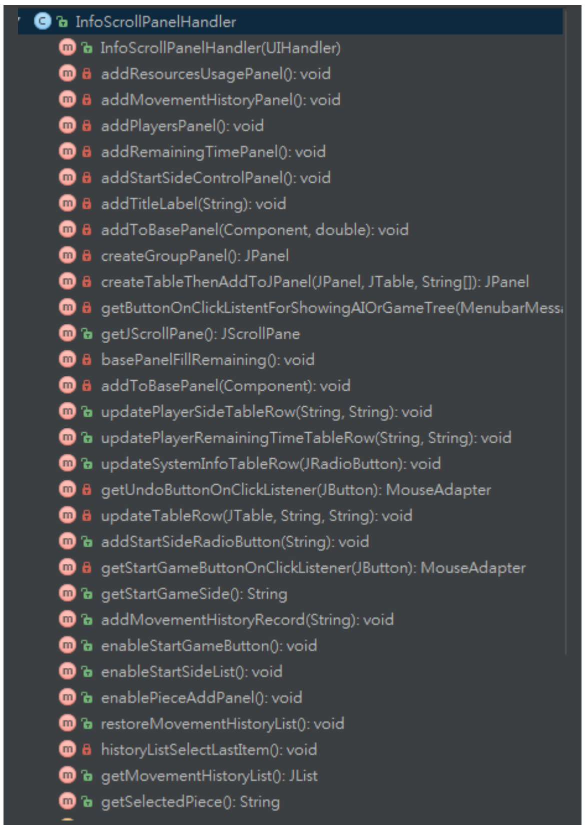
Fig.5.2i entire StatusBarHandler class

- updateStatusBarStatus() set the text of the text label.

Executable / UIHandlerModel / MenuBarHandler

This class set up the Menubar, the menus and the menu items of the menu bar. All the menu items are registered the GUI event listener. User chosen any menu item will be directly report to the UIHandler.

Executable / UIHandlerModel / InfoScrollPanelHandler



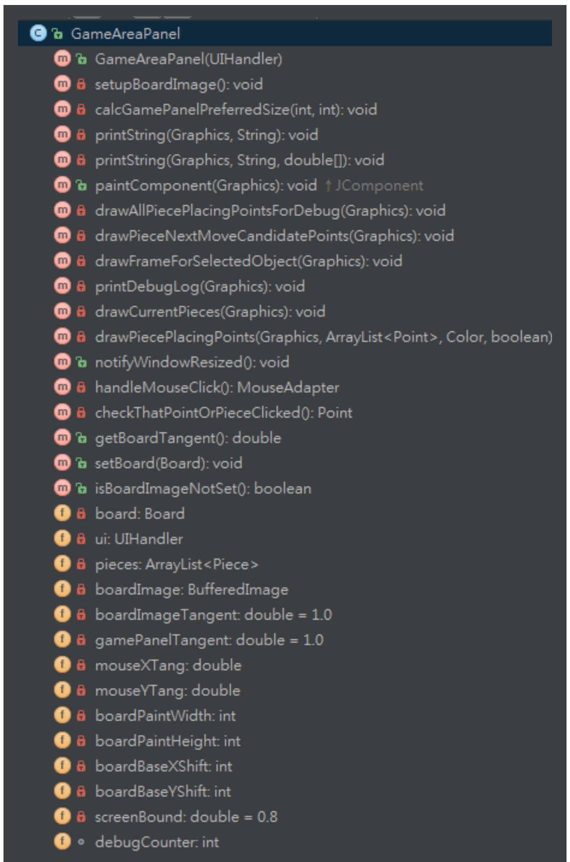
```
InfoScrollPanelHandler
    InfoScrollPanelHandler(UIHandler)
    addResourcesUsagePanel(): void
    addMovementHistoryPanel(): void
    addPlayersPanel(): void
    addRemainingTimePanel(): void
    addStartSideControlPanel(): void
    addTitleLabel(String): void
    addToBasePanel(Component, double): void
    createGroupPanel(): JPanel
    createTableThenAddToJPanel(JPanel, JTable, String[]): JPanel
    getButtonOnClickListentForShowingAIOrGameTree(MenubarMess
    getJScrollPane(): JScrollPane
    basePanelFillRemaining(): void
    addToBasePanel(Component): void
    updatePlayerSideTableRow(String, String): void
    updatePlayerRemainingTimeTableRow(String, String): void
    updateSystemInfoTableRow(JRadioButton): void
    getUndoButtonOnClickListener(JButton): MouseAdapter
    updateTableRow(JTable, String, String): void
    addStartSideRadioButton(String): void
    getStartGameButtonOnClickListener(JButton): MouseAdapter
    getStartGameSide(): String
    addMovementHistoryRecord(String): void
    enableStartGameButton(): void
    enableStartSideList(): void
    enablePieceAddPanel(): void
    restoreMovementHistoryList(): void
    historyListSelectLastItem(): void
    getMovementHistoryList(): JList
    getSelectedPiece(): String
```

Fig.5.2j methods of InfoScrollPanelHandler class

- The add[Name]() methods create each small area in the information scroll panel. It shows the player siders, movement history, piece to be added and so on.
- update[Name]() methods use to update the data content in the information scroll panel.
- get[Name]() methods retrieve the data from the panels and return to the tartget class.
- Enable[Name]() functions set whether the GUI components are enabled to be used.

Executable / UIHandlerModel / GameAreaPanel

The Chess GameAreaPanel shows the chess board and the game content.



```
GameAreaPanel
    GameAreaPanel(UIHandler)
    setupBoardImage(): void
    calcGamePanelPreferredSize(int, int): void
    printString(Graphics, String): void
    printString(Graphics, String, double[]): void
    paintComponent(Graphics): void ↑JComponent
    drawAllPiecePlacingPointsForDebug(Graphics): void
    drawPieceNextMoveCandidatePoints(Graphics): void
    drawFrameForSelectedObject(Graphics): void
    printDebugLog(Graphics): void
    drawCurrentPieces(Graphics): void
    drawPiecePlacingPoints(Graphics, ArrayList<Point>, Color, boolean)
    notifyWindowResized(): void
    handleMouseClick(): MouseAdapter
    checkThatPointOrPieceClicked(): Point
    getBoardTangent(): double
    setBoard(Board): void
    isBoardImageNotSet(): boolean
    board: Board
    ui: UIHandler
    pieces: ArrayList<Piece>
    boardImage: BufferedImage
    boardImageTangent: double = 1.0
    gamePanelTangent: double = 1.0
    mouseXTang: double
    mouseYTang: double
    boardPaintWidth: int
    boardPaintHeight: int
    boardBaseXShift: int
    boardBaseYShift: int
    screenBound: double = 0.8
    debugCounter: int
```

Fig.5.2k entire content of the GameAreaPanel

The GameAreaPanel supposes to have a similar structure to ChessBoardGraphicAreaPanel in Chreator. Since this class is written quite a long time ago, the structure of this class is a bit messy. Generally, the print[Name]() methods draw strings on the chess board, draw[Name]() methods draw the figures and images of the chess pieces, chess board and so on. The point that triggers the drawing is the paintComponent() method.

The remaining methods are utility such as set and get the chess board, calculating the ratio of the chess board and so on.

Executable / DataAndSetting

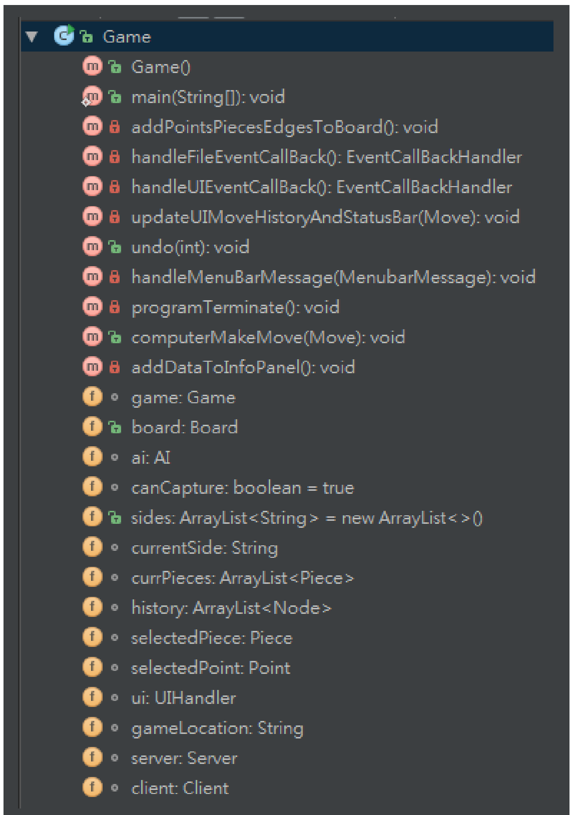
It is a very long and big class generated by Chreator to store the graphical data of the chess board, chess piece and point edge of the game.

- Preferred size of the chess board is stored, such that the ratio of the chess board still holds although the image of the chess board is missing.
- Player sides, relative size, piece name and the image link of a chess piece are also stored. The class will return a well-defined chess piece by calling the makeStandardPiece() method, providing the piece type and piece side.
- The initial piece placing point for the piece is recorded. That is, the point ID that the chess piece will be immediately added to when the game starts

- Relative size, relative coordinates and the ID for all the piece placing points are stored
- Information about the neighbor points are stored, including neighbor point ID and the edge direction to that point.

Executable / Game

The Game class works the sample principle as the ChreatorLauncher.



```
▼ Game
    Game()
    main(String[]): void
    addPointsPiecesEdgesToBoard(): void
    handleFileEventCallBack(): EventCallBackHandler
    handleUIEventCallBack(): EventCallBackHandler
    updateUIMoveHistoryAndStatusBar(Move): void
    undo(int): void
    handleMenuBarMessage(MenubarMessage): void
    programTerminate(): void
    computerMakeMove(Move): void
    addDataToInfoPanel(): void
    game: Game
    board: Board
    ai: AI
    canCapture: boolean = true
    sides: ArrayList<String> = new ArrayList<>()
    currentSide: String
    currPieces: ArrayList<Piece>
    history: ArrayList<Node>
    selectedPiece: Piece
    selectedPoint: Point
    ui: UIHandler
    gameLocation: String
    server: Server
    client: Client
```

Fig.5.21 entire content of the Game class

- Game holds an instance for the high level handlers and class objects, such as the UIHandler, AI, Server, Client and so on.
- The main() function start the create the Game instant. Game instance builds the handlers and class objects.
- Game should handle the action of “new game”, “save game”, “and load game” function for the corresponding menu item pressed; “cancel movement”, “confirm movement”, “start game”, “undo”, for point selected messages from the UIHandler. And providing the current selected point at its valid move available points to the UIHandler.
- Updating the step history and other information in the InfoScrollPanelHandler

Usage of the entire chess game executable

In the project setting page of the Chreator, compile and run the executable, a game will show as figure 5.2g.

- Select the start side, and press the start game, the chess board should be filled with the chess piece that placing in the initial piece placing points defined in the Chreator. Please follow the game flow to play the game.
- At the bottom on the status bar, there are 2 buttons – cancel move and confirm move. They are used to cancel the move or confirm the move depending on the need.
- Pressing [File] → [New Game], all the data in the game will be re-initialized.
- Press [File]→[Load Game], a file chooser will prompt up and ask to provide the game save
- Press [File]→[Save Game], file chooser will prompt up to ask for the saving location and the file name
- Press [File]→[Exit], the game terminates
- Press [Edit]→[Undo], 1 step will be undone
- Press [Game] → [Wait for player] or [Connect to server], online playing mode is launched. Chess board for the game executables that are successfully synchronized will shares the same chess board to play with.
- Press [View] → [Show Debug], debug message about the relative cursor position, board image size, panel size, select points will be shown
- Press [View] → [Show Piece Placing Points] highlights all the piece placing points.

Chapter 6: Phase work, Result and Deliverables

The project started from the early September last year. At that time we were finalizing our target, project plan and the system structure. Since we decided to divide the project into the chess game executable and chess game making engine Chreator, I started implementing the basic structure of the chess game executable from the early October. By the early December, I finished the implementation of the chess game executable and I pass the remaining non-GUI related implementation work to Kit. He focus on improving the structure of making the chess board, chess piece programming structure combatable with the AI algorithm and supporting the more types of chess game. At that time we paused our work due to examination and resumed the work at the end of December. By interim presentation in the middle of the January, half of the functions of the chess board making tab in the Chreator is done. We speeded up our developing process. I focused on implementing the Project Setting Page, Chess Board Editing Page and Chess Piece Editing Page of the Chreator. Kit focus on implementing the code reading and producing of a chess game project in the Chreator, and other fundamental functions in both the chess game source code and Chreator.

By the few days before the submission of the project, all the development process has to be stopped. At that time, all the functions in the Project Setting tab, Chess Boarding Editing tab and the Chess Piece Editing tab of the Chreator is done. About one fifth of the Java Code Editor for the Chreator is done. Chreator can now at least produce and read a correct chess game project, applying the data to the chess game source code project. The chess game executable now supports normal chess piece adding, moving and mixing playing; saving and loading states of a game; network remote playing and other new functions.

There are planned functions failed to be implemented due to time limitation. The

AI page and the Game Rule Page in the Chreator are not implemented.

For the chess game executable, there are AI, node, state and other classes related to the implementation of the AI. The AI class uses the modified library from the author on the Internet for demonstration. We do not test on whether the demonstration AI class works on the game formally.

Since there are lots to implement for the basic functions of both the software, we at the end evenly fail to implement some function. There may be lots of defense, bugs and unknown errors.

The deliverable of this stage is the executable jar file of the Chreator – "Chreator.jar", reviewing source code of the Chreator and the chess game executable.

Chapter 7: Difficulties, Reflection and Improvement

In this project, I consider our work as not that successful. There are lots of difficulties when we are dealing with this project. They are mainly the time management, technical problem, communication problem with the partner and project approach problem.

For the time management and technical problem, I found that there are actually too many components to be implemented if using the method we proposed. Originally, I think that it is easy to implement so many sub-tasks especially on the GUI programming as I think that those sub-tasks can be implemented within very short time. However, in order to program the GUI, test for the interaction and find for bugs, I may need 2 weeks for just dealing with 1000 lines of code for certain component. This may be the problem of troublesome Java Swing programming. We may consider using the other approaches if we have chance to discuss on this topic again in the future.

There are communication problem exists between I and Kit. We do not meet that frequently. In the later developing process, we may not meet very regularly due to work load of other courses. We may easily miss the working progress of each other and delay the whole developing progress.

For the project approach problem, I am thinking of if there are any new programming languages, libraries or approach to implement the same project but the amount of coding and the coding difficulties can be greatly reduced. We always think that Java is the best programming language as it is frequently used and popular, such as Android application programming uses Java. We may need to redesign the implementation approach.

Chapter 8: Division of Labor

Below is my work done in my point of view, this may be differing to the version of Kit.

Chreator:

- Implementation and design of the ChreatorLauncher class, ProjecCompiler class, entire UIModule package, ObjectModel package of the Chreator

Chess Game Executable:

- Design and implementation of the most of the UIHandlerModel package, DataAndSetting class
- Design with Kit, first generation of implementation on BoardModel package, Piece class inside PieceModel, Game class

Below is the work done by Kit:

Chreator:

- Implementation and design of the CodeLoader package and CodeProducer package of the Chreator
- Modification and improvement for some classes inside the UIModule of the Chreator

Chess Game Executable:

- Design and implementation of the FileHandlerModel package, ObjectModel package and SocketConnection package
- Design with me, later generation of implementation on BoardModel package, Piece class inside PieceModel, Game class and UIHandlerModel

Chapter 9: Conclusion

Although I personally do not satisfy with the result and the current stage of the project, there are lots of planned functions not implemented. Testing is not done seriously as a result there may contain bugs in the program. However, I cherish the programming technique and coding practice in this project. I learn a lot about the disadvantage of my coding practice and design of the system structure. I can improve the design of the system I implement in the future. Also I learn the importance of how a highly object-oriented structure can reduce the trouble when sharing code with programmers working with the same project, help speeding up the system maintenance efficiency and probability of producing defense.

I am glad that I can participate in such an interesting topic, it is hope that I can get in touch with this topic again.