



TCP2201 Myrmidon Chess Project
Trimester 1, 2018/2019
by iChess

Team Leader:

Ewana Ayesha binti Redzuan, 013-280 8377,
ewanaayesha.ea@gmail.com

Team members:

Muhamad Irfan Dhiyauddin Bin Mohd Hafizul, 0194926318,
irfan181298@gmail.com

Muhammad Fattah Bin Sallehuddin , 019-6331728,
fattah.sallehuddin@gmail.com

Ahmad Faiq bin Ahmad Radzali, 011-3535 6362
faiqradzali@gmail.com

Table Of Content

UML Class Diagram	2
2. Use Case Diagram	4
3. Sequence Diagram	6
4. Design Patterns	8
4.1 Builder Pattern	8
4.1.1 Implementation	8
4.2 Model-View-Controller Pattern	9
4.2.1 Implementation	9
5. User Guide	10
5.1 Compile & Run Instructions	10
5.2 How To Use The Program	13

1. UML Class Diagram

The UML class diagram below shows the relationship between classes. It depicts the type of relationship each class have with each other such as inheritance, composition and aggregation.

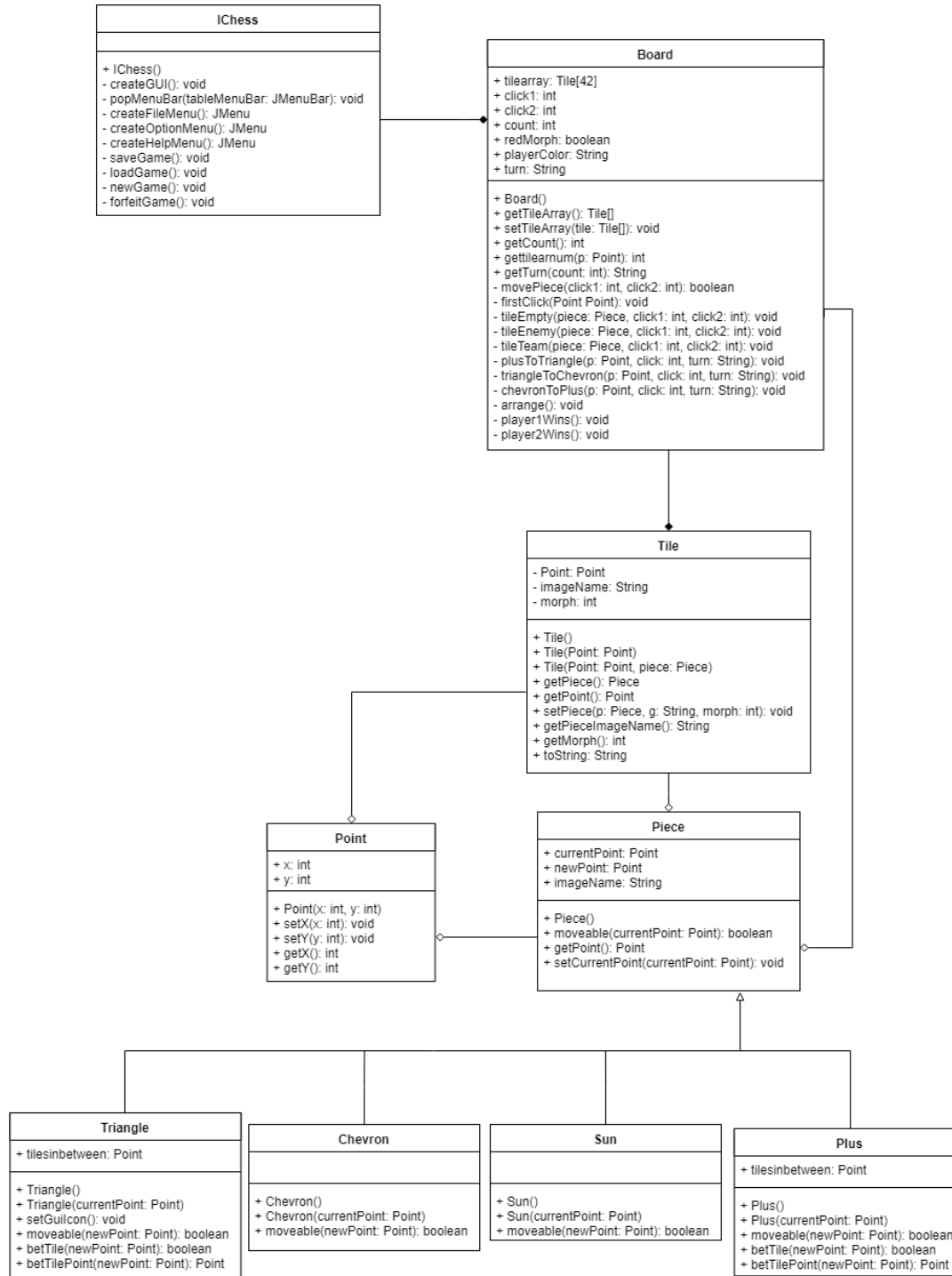


Diagram 1: UML Class Diagram of IChess: Myrmidon Chess

2. Use Case Diagram

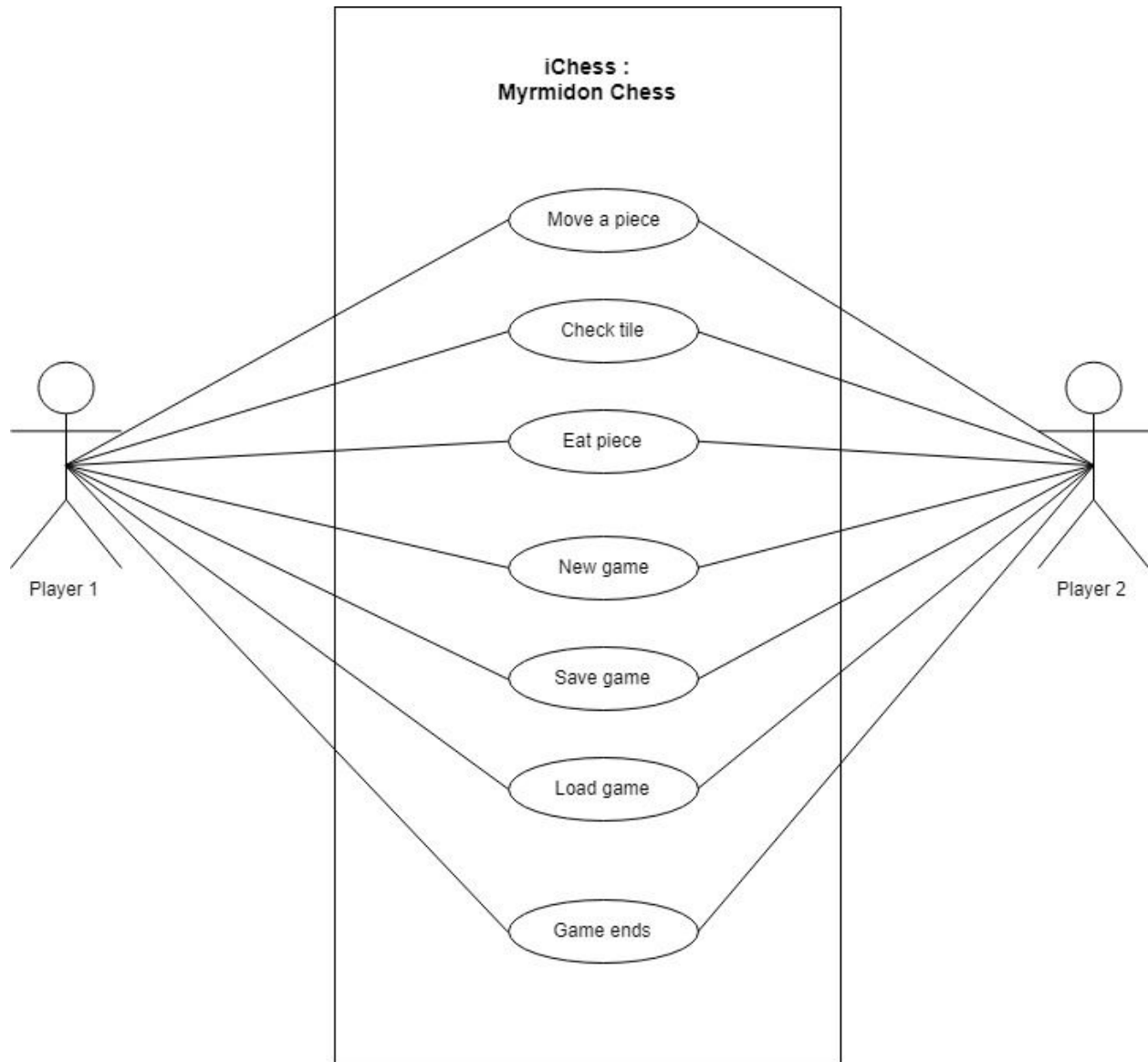


Diagram 2: Use case diagram of iChess: Myrmidon Chess

The iChess: Myrmidon Chess game has only 1 user and that ranges from Player 1 to Player 2. In the use case diagram, both Player 1 and Player 2 are able to access the main functions of the game, but not simultaneously. The program counts the turn in a `getTurn()` method that makes sure that either a Blue Piece or a Red Piece must have one and only one turn.

A player can only move a piece when a game has started. At the 'move a Piece' function, the player is able to do these possibilities:

- Move a piece on an empty tile
- Eat a piece
- Morph pieces: The system morphs the pieces into their respective forms after each team went through 3 turns. The pieces morph according to this cycle.

For each movement, the system collects the current Piece and a Tile destination in the form of integer. The first click of a Tile, must have a Piece to be moved and second click of a Tile must be either empty (have no pieces) or contains enemy Piece. The System restricts a destination that contains the Player's own Team Piece. When a movement fails, the player will have to go back to the first click and choose his/her movement again.

3. Sequence Diagram

This sequence diagram shows the general program and how it works arranged in time sequence. It depicts how the objects and classes are involved in the game iChess. As how the program will work, at first player 1 need to move any piece to another tile. If the tile that player 1 choose is valid, the piece will move. If the tile is invalid, the piece will not be moved and the system will wait for player 1 to move any piece again. Until then, in order to move, player 2 needs to wait until player 1 has done with his movement. The movement logic is like this; first, the system will check if the tile is empty and the movable is true, then the piece will be moved. If the tile has player 1 own piece, the piece will not move because “friendly-fire” concept is not allowed in chess. Else, if the tile has opponent’s piece, player 1 will eat the piece and will move there.

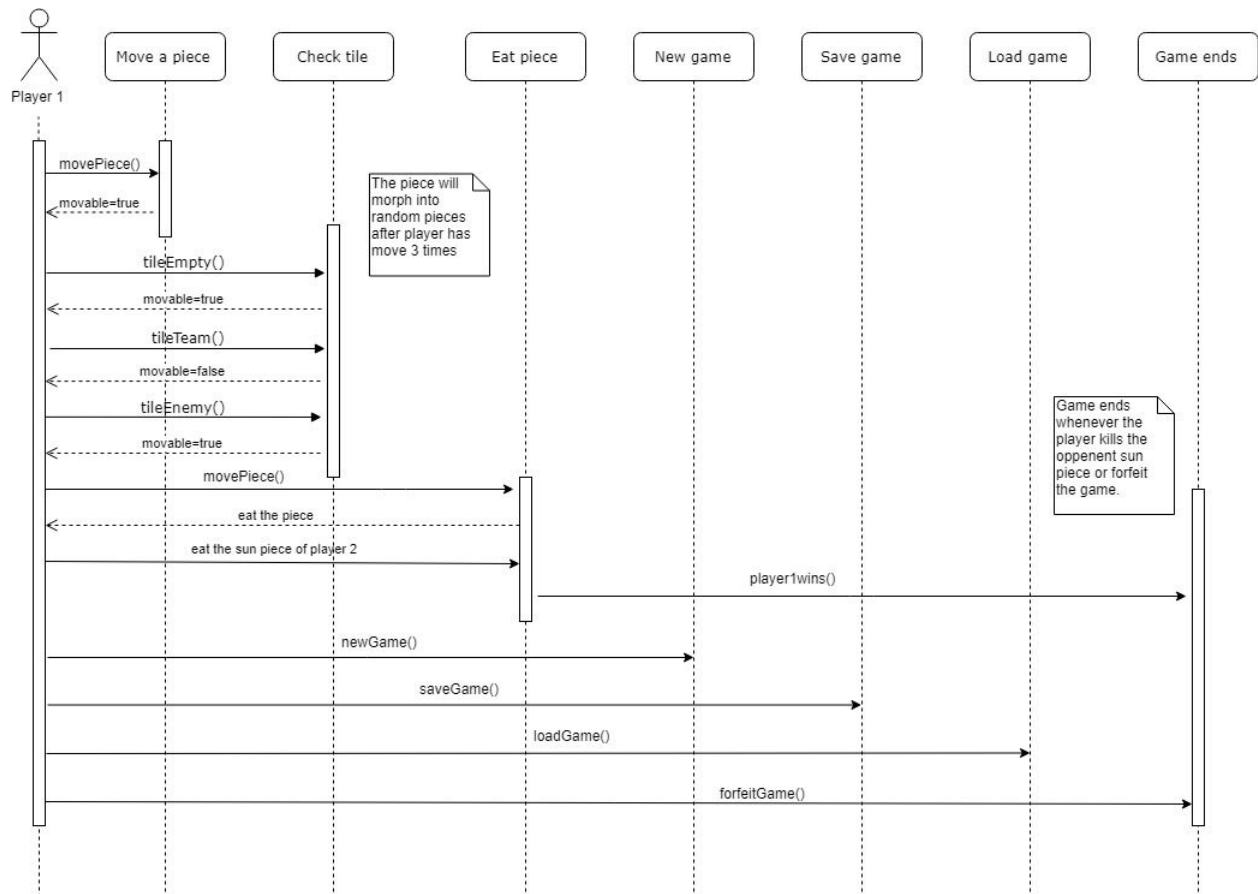


Diagram 3.1: Sequence Diagram of iChess: Myrmidon Chess for Player1

If player 1 eats the sun piece of player 2 or vice versa, the player that eat the piece will be considered as the winner, thus the game will end. In addition, both player 1 and player 2 can choose to start a new game, save the game, load the game (if there is a saved file to load from), and forfeit if they want to. Same as eating the sun concept, forfeit allows the user to give the opponent the winner title thus, ending the game.

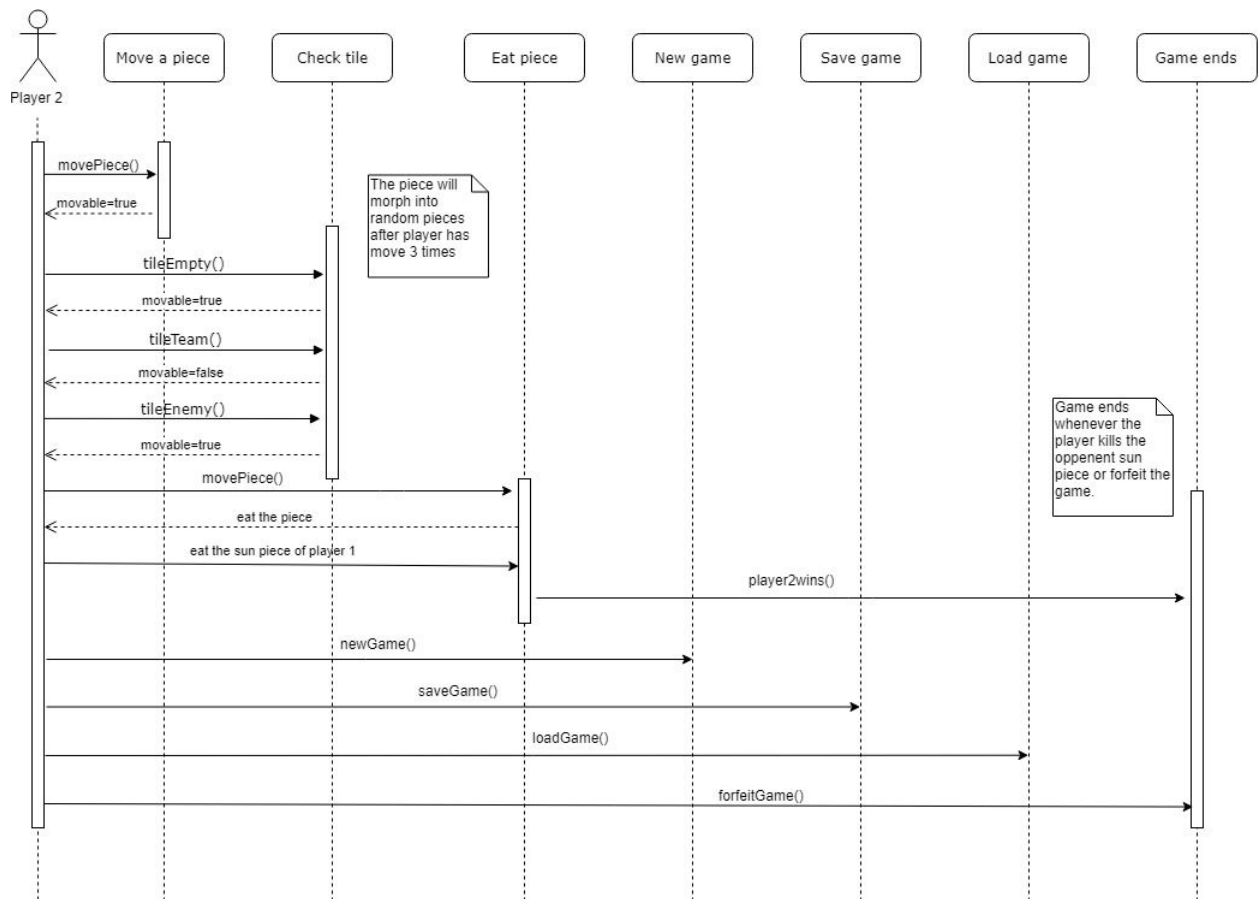


Diagram 3.2: Sequence Diagram of IChess: Myrmidon Chess for player 2

4. Design Patterns

4.1 Builder Pattern

Builder pattern builds a complex object using simple objects and using a step by step approach.

4.1.1 Implementation

We use this pattern to create the Board of our iChess: Myrmidon Chess game, as Board extends the JPanel as the parent, the components of the panels includes 42 Tiles in an array called tilearray. In order to create the Board, the Tile must be built by the instantiation of Piece and Point.

```
1 class Board extends JPanel{
2     //creating an array of 42 tile buttons
3     Tile[] tilearray = new Tile[42];
4     Piece piece = new Piece();
5
6     Board(){
7         super(new GridLayout(6,7));
8         int a = 1 , b = 1;
9         for(int i=0; i<42; i++){
10             if (b > 7){
11                 a++;
12                 b=1;
13             }
14             Point p = new Point(a,b);
15             Tile tile_obj = new Tile(p);
16             ...
17
18             tilearray[i]=tile_obj;
19             this.add(tilearray[i]);
20             b++;
21         }
22     }
23 }
```

Code Snippet 4.1.1: Simplified Board Class

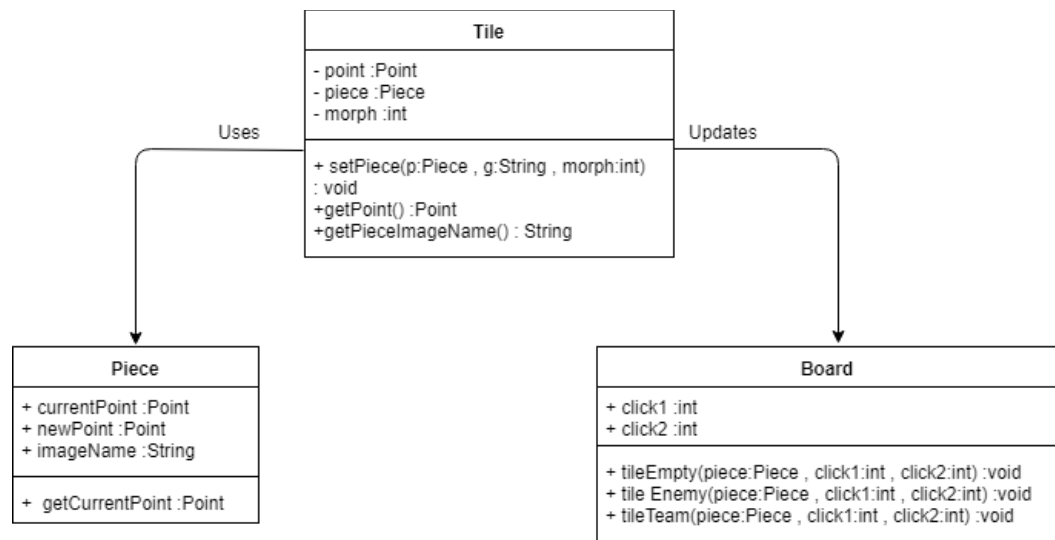
A Board must contain an array of Tiles. Each Tile built with having a compulsory Point but not a compulsory a Piece. This is because a Tile might not have a Piece. The Board object must be created step by step.

4.2 Model-View-Controller Pattern

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

- Model - Model represents an object or carrying data. It can also have logic to update controller if its data changes.
- View - View represents the visualization of the data that model contains.
- Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

4.2.1 Implementation



**Diagram 4.2.1 : A simplified UML Class Diagram of how MVC Design pattern is implemented in the program*

Based on Diagram 4.2.1, the Piece class acts as the model, Tile class as the controller and Board method's specifically tileEmpty, tileEnemy and tileTeam as the View. The Board can only access the information for piece through the Tile(controller). To further explain this, in order to retrieve, the current Point of Piece, the Board uses `tilearray[click1].getPoint()` .

The controller sets as well as get both point and the imageName. These changes then are updated on the Board(View) as methods like tileEmpty(), uses it to display to the GUI interface that the movement is successful when move to an empty tile.

5. User Guide

5.1 Compile & Run Instructions

- i) Download and extract the file's content into a folder of your choosing as shown in Figure 1.

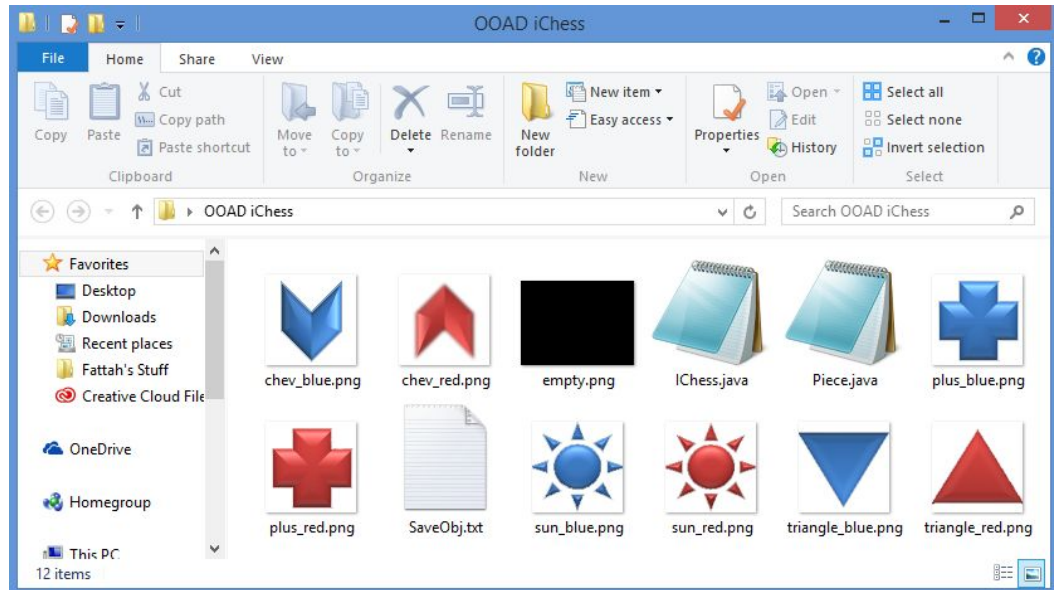


Figure 5.1.1

- ii) For Windows user, open Command Prompt (CMD) and enter the destination of the file you just extracted using the command "cd ..".

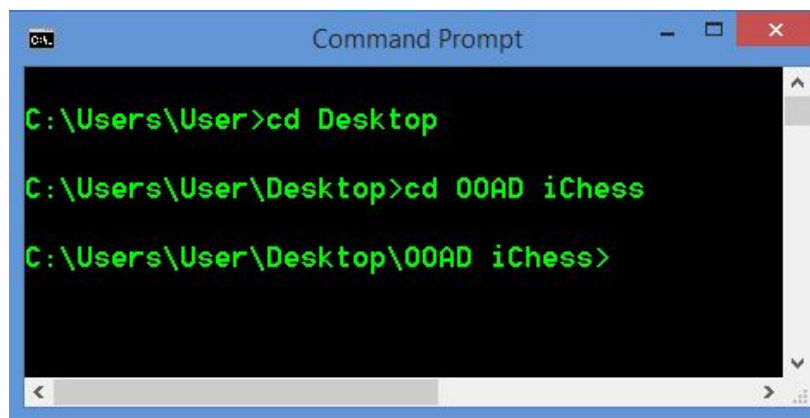
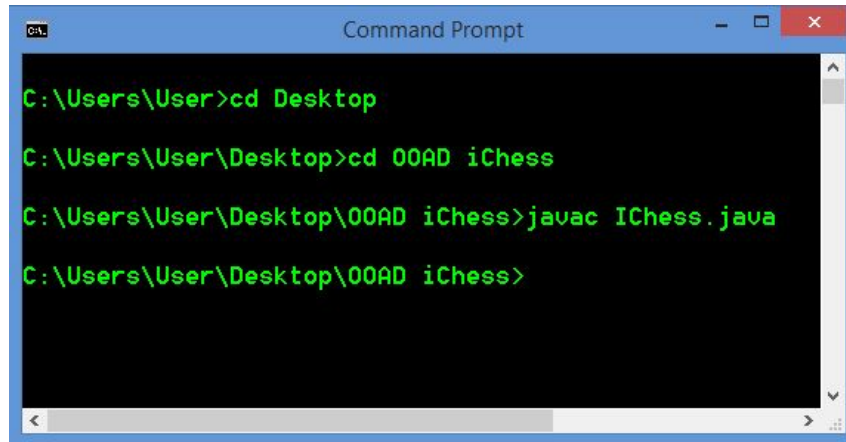


Figure 5.1.2

- iii) Compile the program using the command “Javac IChess.java”.
Please make sure the capitalization of the words are correct based on the file’s name as shown in Figure 5.1.3.

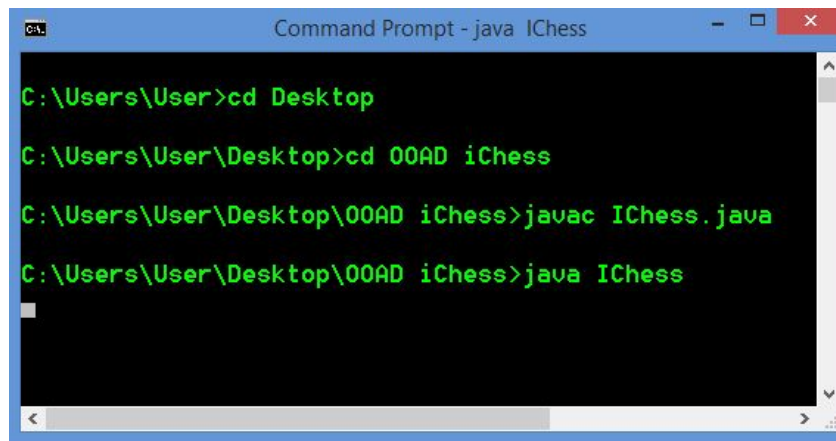


```
Command Prompt

C:\Users\User>cd Desktop
C:\Users\User\Desktop>cd 00AD iChess
C:\Users\User\Desktop\00AD iChess>javac IChess.java
C:\Users\User\Desktop\00AD iChess>
```

Figure 5.1.3

- iv) Finally, run the compiled program using the command “Java IChess” as in Figure 5.1.4.



```
Command Prompt - java IChess

C:\Users\User>cd Desktop
C:\Users\User\Desktop>cd 00AD iChess
C:\Users\User\Desktop\00AD iChess>javac IChess.java
C:\Users\User\Desktop\00AD iChess>java IChess
```

Figure 5.1.4

5.2 How To Use The Program

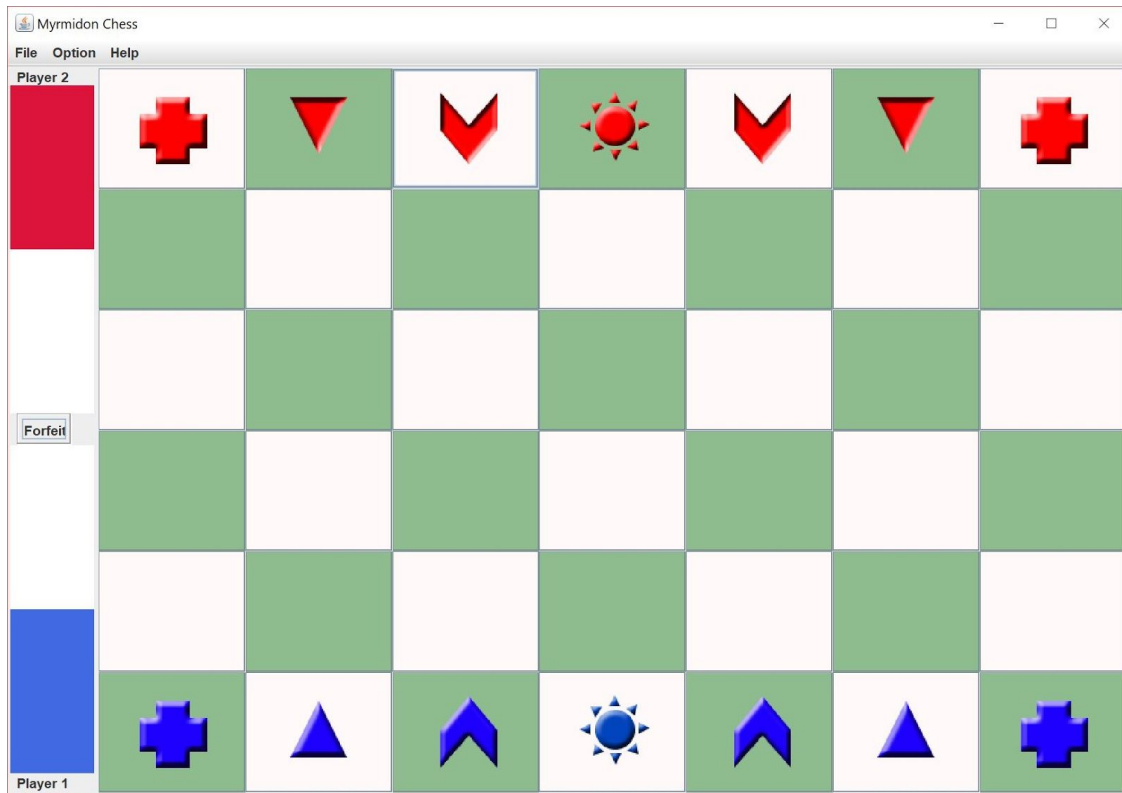


Figure 5.2.1

Figure 5.2.1 shows the main screen that will appear as soon as user started the Java program. User can immediately play the game or view available drop down menus on top of the frame.

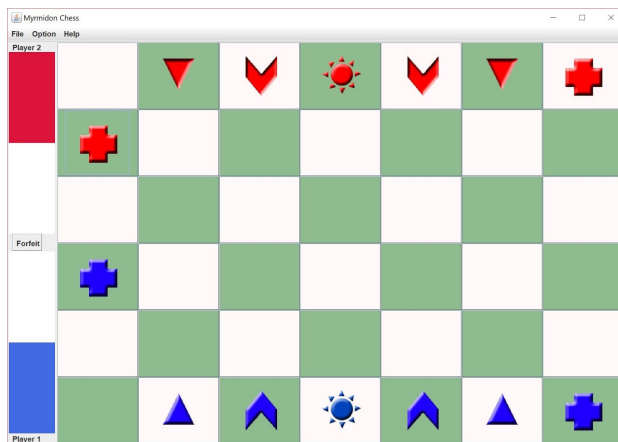


Figure 5.2.2

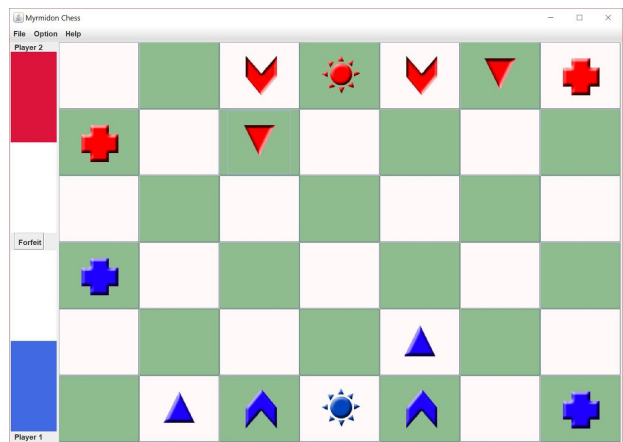


Figure 5.2.3

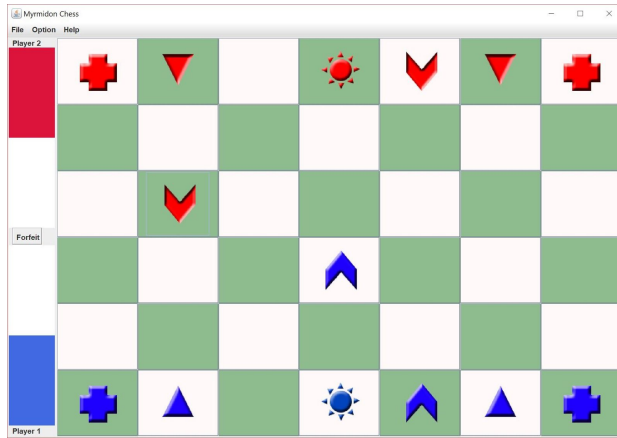


Figure 5.2.4

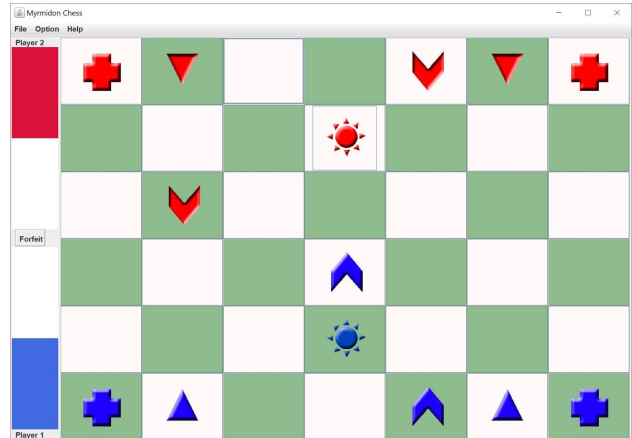


Figure 5.2.5

Different movement styles are applied to different piece. Figure 5.2.2 shows the movement criteria for a 'plus'. A 'plus' can be moved vertically or horizontally for a maximum of two tiles. Figure 5.2.3 shows how a 'triangle' is moved. From its original position, a triangle can be moved left and right diagonally for a maximum of two tiles. Based on Figure 5.2.4, a 'chevron' move in an L shape with extra conditions of exactly 2 steps straight then 1 step to the left or right. No any other shape of L is allowed. A 'sun' can move 1 tile to any directions, as shown in Figure 5.2.5. All pieces can move either forward or backward as long as there is enough space.

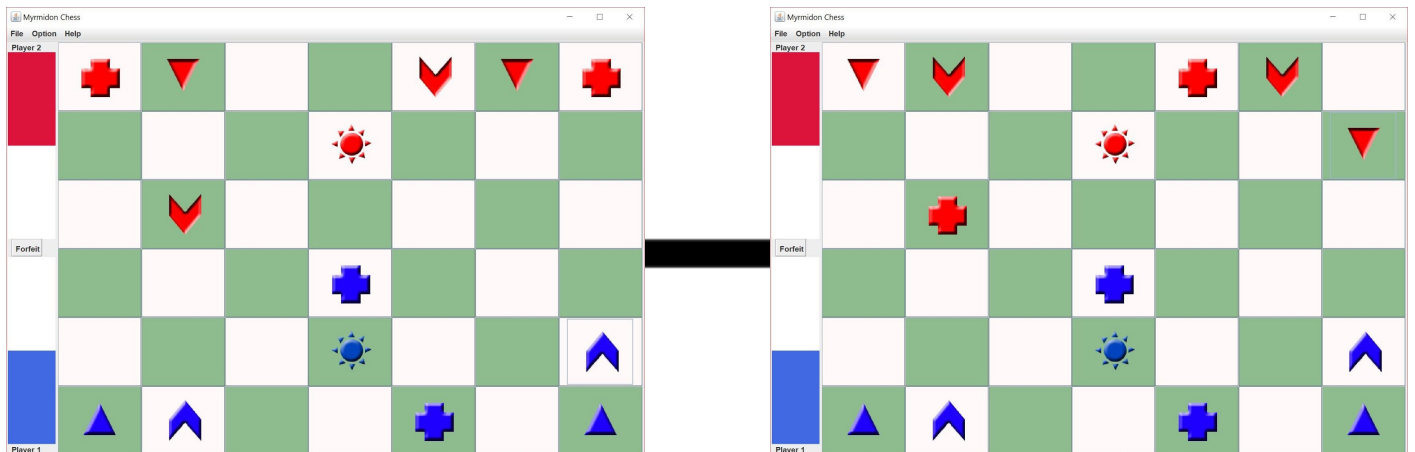


Figure 5.2.6

Figure 5.2.6 shows how the red pieces are transformed after 3 successful turns. 'Plus' is transformed to 'triangle', 'triangle' is transformed to 'chevron', and 'chevron' is transformed to 'plus'. This transformation will always repeat after 3 turns of an individual player. The 'Sun' will not be transformed since it is the main piece in this game.



Figure 5.2.7

This is a turn-based game where Player 1 (Blue) will move first and then followed by Player 2 (Red) sequentially. If players fails to follow the turn, an error popup such as in Figure 11 will come out telling players the current turn.

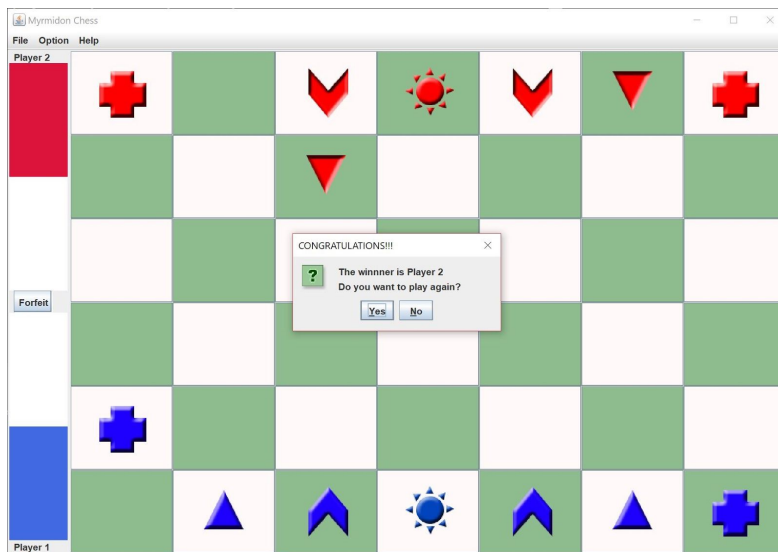


Figure 5.2.8

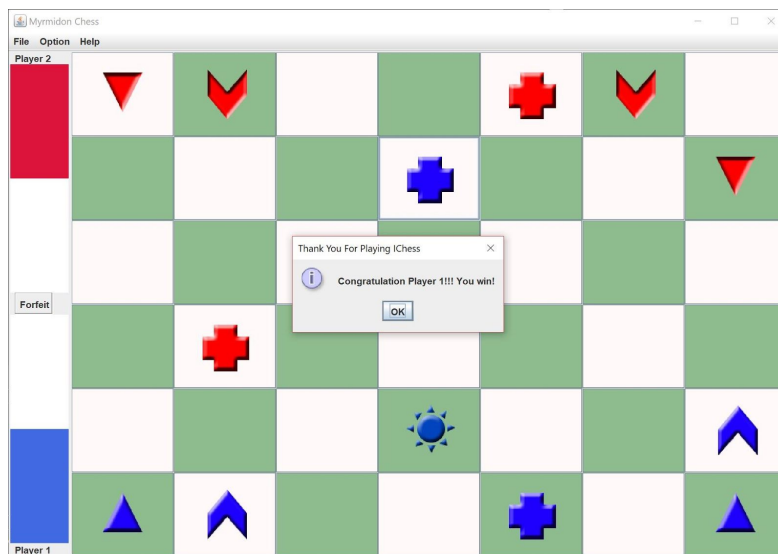


Figure 5.2.9

The game ends in two conditions, if a player press the forfeit button, the player who pressed it will be declared as the loser, and they will be asked if they would like to start a new game as in Figure 5.2.8. If any player manages to capture the ‘Sun’ of another player, the player will be declared as the winner as in Figure 5.2.9.



Figure 5.2.10

Dropdown menus are available for multiple options, clicking ‘Save game’ will save the currently played game. ‘Load game’ will restore the most recent saved game and the players can continue playing. The ‘Help’ menus consists of ‘How to play the game’ and ‘About’.

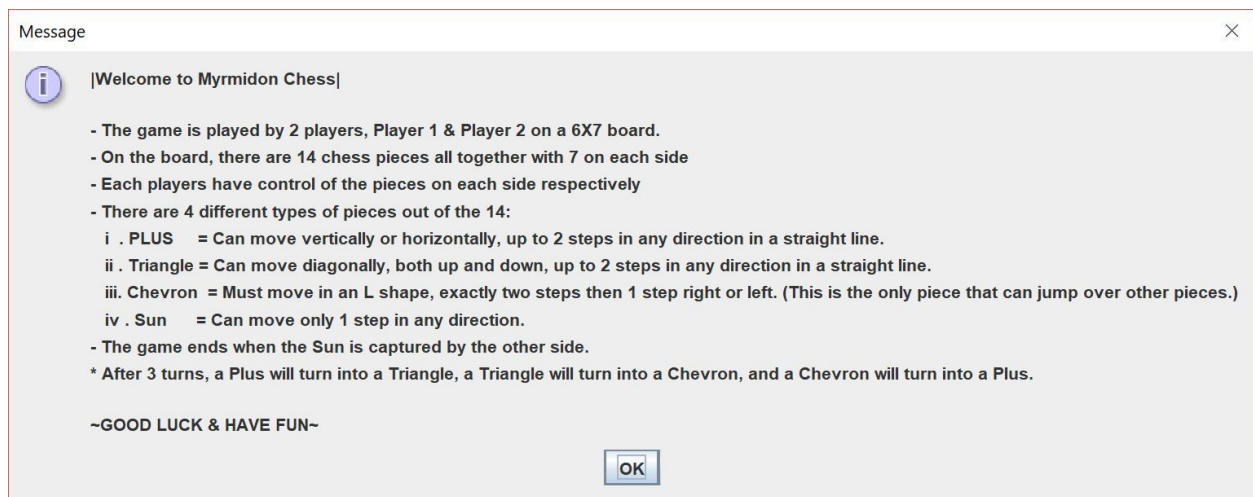


Figure 5.2.11

Clicking ‘How to play the game’ will pop up a window explaining this game’s procedure as shown in Figure 5.2.11.



Figure 5.2.12

Clicking 'About' will pop up a list of all developers for this game as shown in Figure 5.2.12.

TCP2201 Project Evaluation Form (30%)

Tutorial Section:	
Team Name:	
Group Leader:	
Member	
Member	
Member	

Prototype and Presentation (20%)

Item	Maximum marks	Actual Marks
Comments, indentation, following proper Java naming conventions, other Java style issues.	2	
Object-oriented concepts like subclassing, delegation, composition, aggregation, polymorphism, etc.	3	
Appropriate use of Design Patterns	3	
User friendliness and appropriate GUI components used, windows resize properly and the board scales properly, menus still work during game play, etc.	4	
Functional requirements fulfilled, e.g. the board is set up correctly, players can play through a game properly, all the pieces move and transform correctly, winner is declared, save game, load saved game, etc.	8	
Total:	20	

Report (10%)

Item	Maximum marks	Actual Marks
UML Class Diagram done and is coherent with the implementation	3	
Use Case Diagram done and is coherent with the implementation	2	
Sequence Diagrams done and is coherent with the implementation	3	
User Documentation done and is coherent with the implementation	2	
Total:	10	