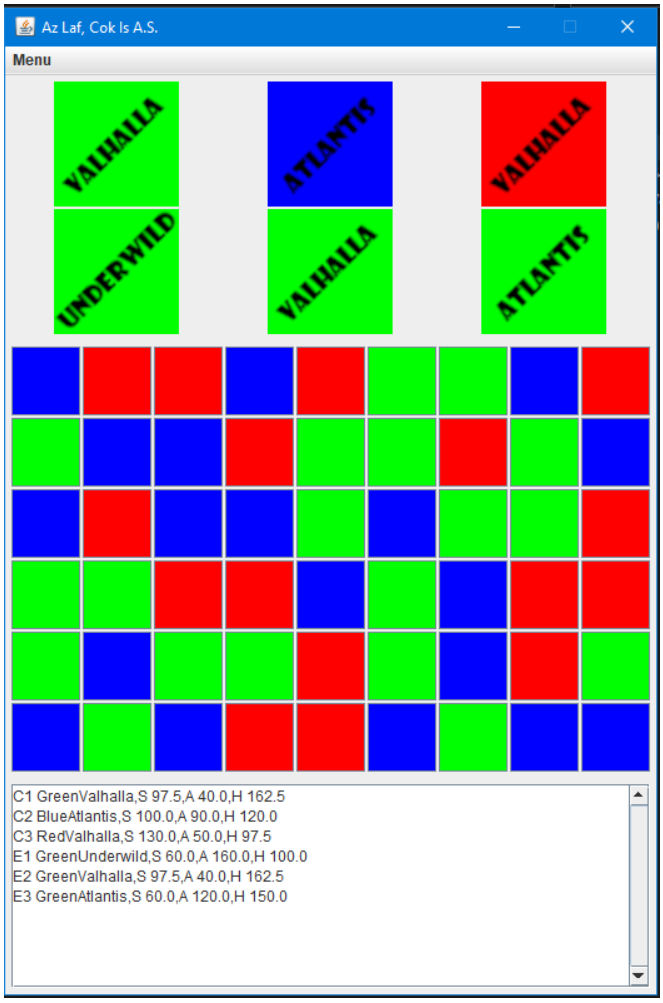
# 00Object Oriented Analysis and Design

# Homework-2 Report

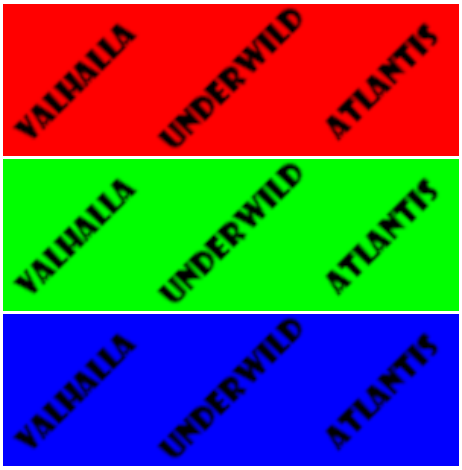

# Muhammed Özkan 151044084


## Game Explanation

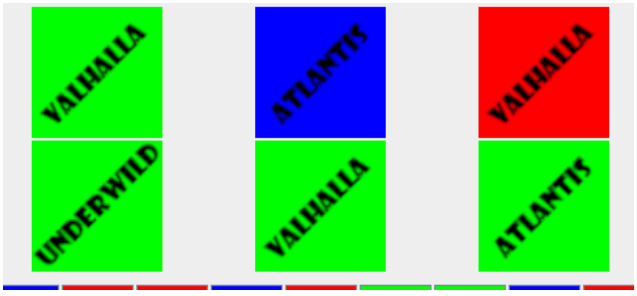
The game screen consists of 3 parts. Upper part player characters and enemy characters. The middle part is 3 different colored stones in a 6*9 size table. The bottom part is the message screen. The game part is the colored stones in the middle. These will try to damage the characters at the top by moving them. The game is turn-based. After the user makes the move, he will make a move on the computer. If the player's characters die, the game is over. If the characters of the computer die, all the characters are renewed and the game continues. The menu is displayed at the top of the screen. There are start, pause and exit buttons in the menu. At the bottom, there is a section where the events in the game will be written as a message.



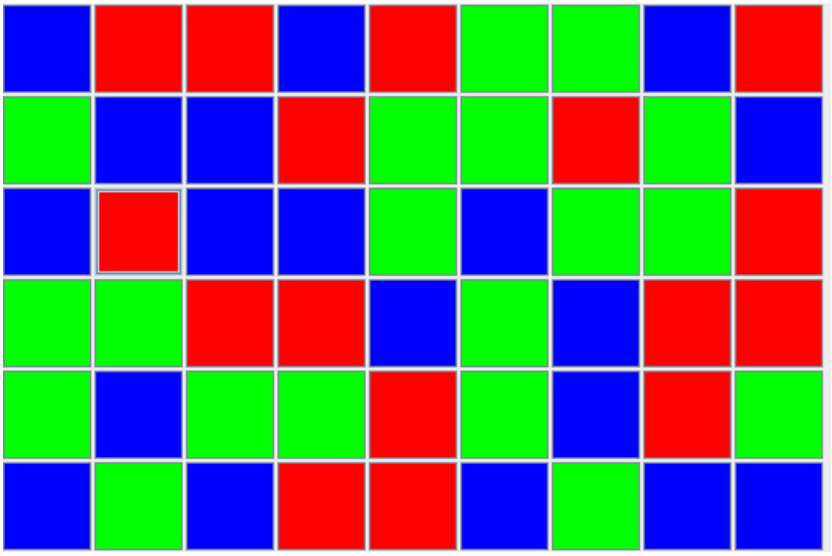
1. There are 3 different types and 3 different styles in the game. There are 9 different characters in total. This character will appear on the screen as follows.




2. The characters in the first row on the screen are the user's characters and the lower ones are the enemy (computer) characters.




3. In this part, the tiles will be changed left and right, up and down, and when matched 3 according to their colors, they will damage the enemy at the top.



## Design Decision

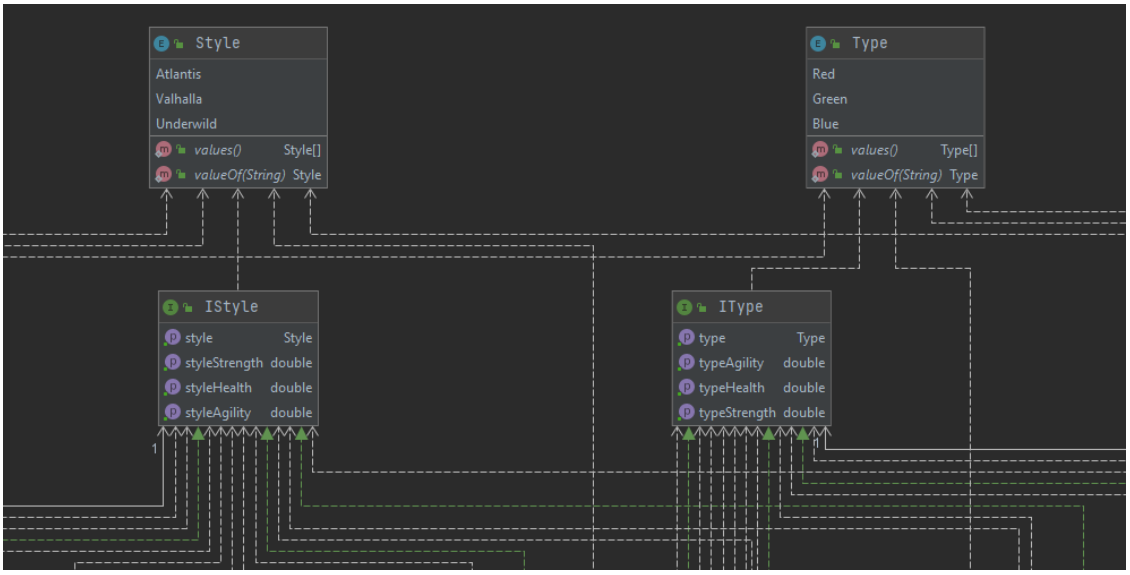
The abstract factory pattern is loosely coupled with the GameCharacter class during character creation as desired. The creation process is abstracted from the user. The factory is built on 2 different interfaces as type and style. By using these, 9 different concrete classes were created.

In the stone matching part, some algorithms at https://github.com/sriniketh923/Candy-Crush are used. The methods used are integrated into the game as desired. At the beginning of the game, tiles are randomly created. then a move is expected from the user. After the user makes the move, the matches are found and the damage to the relevant character is calculated and reduced health as desired. Afterwards, a move is expected from the computer in the same way. At the end of these moves, the side that loses all its characters is defeated. If the computer wins, the game is over. If the user wins, the characters are renewed and the game continues.

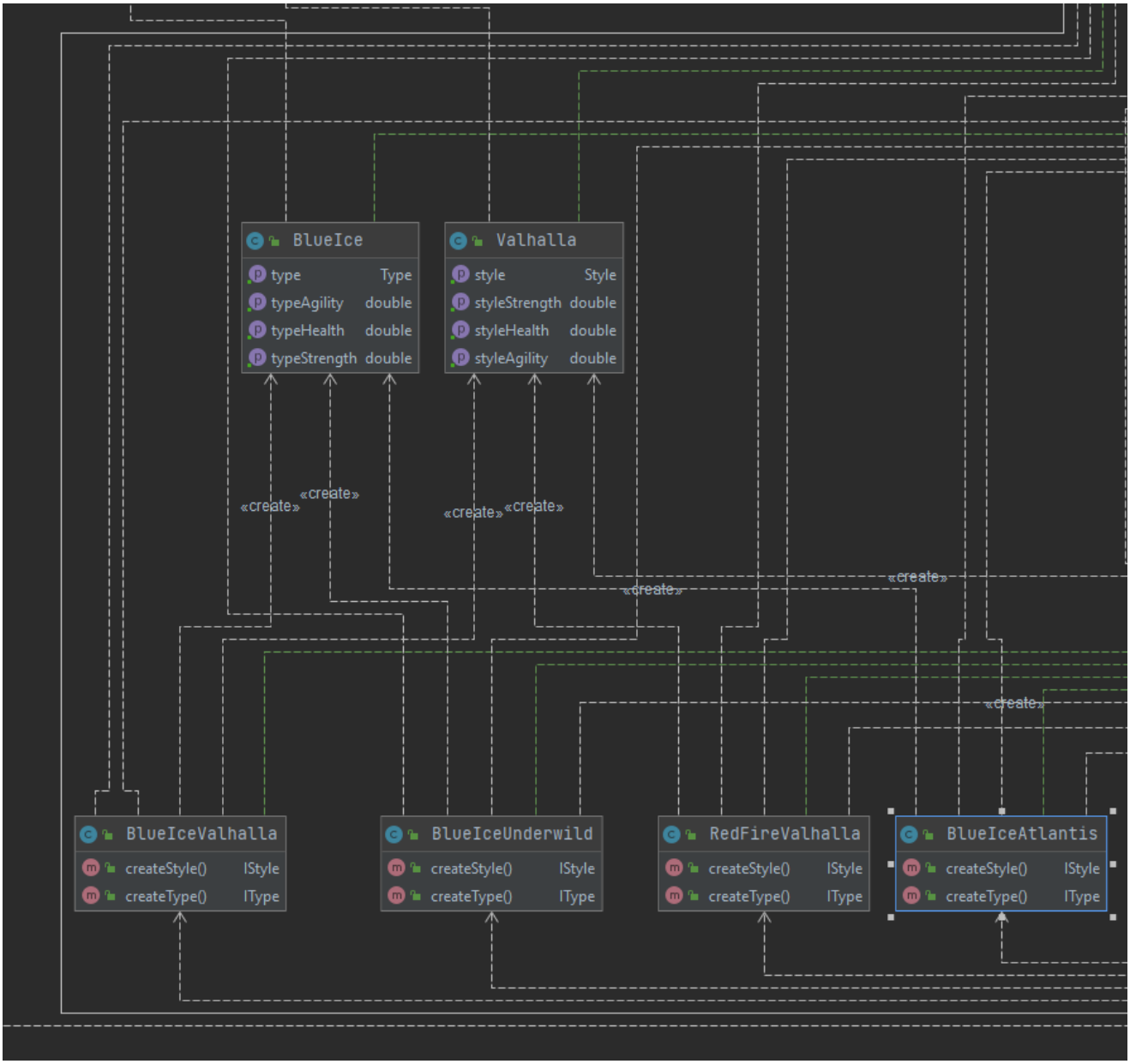
Every movement and repetitive tiles matches are written as a message at the bottom. Operations in between cannot be displayed as animations.

## Uml Diagram

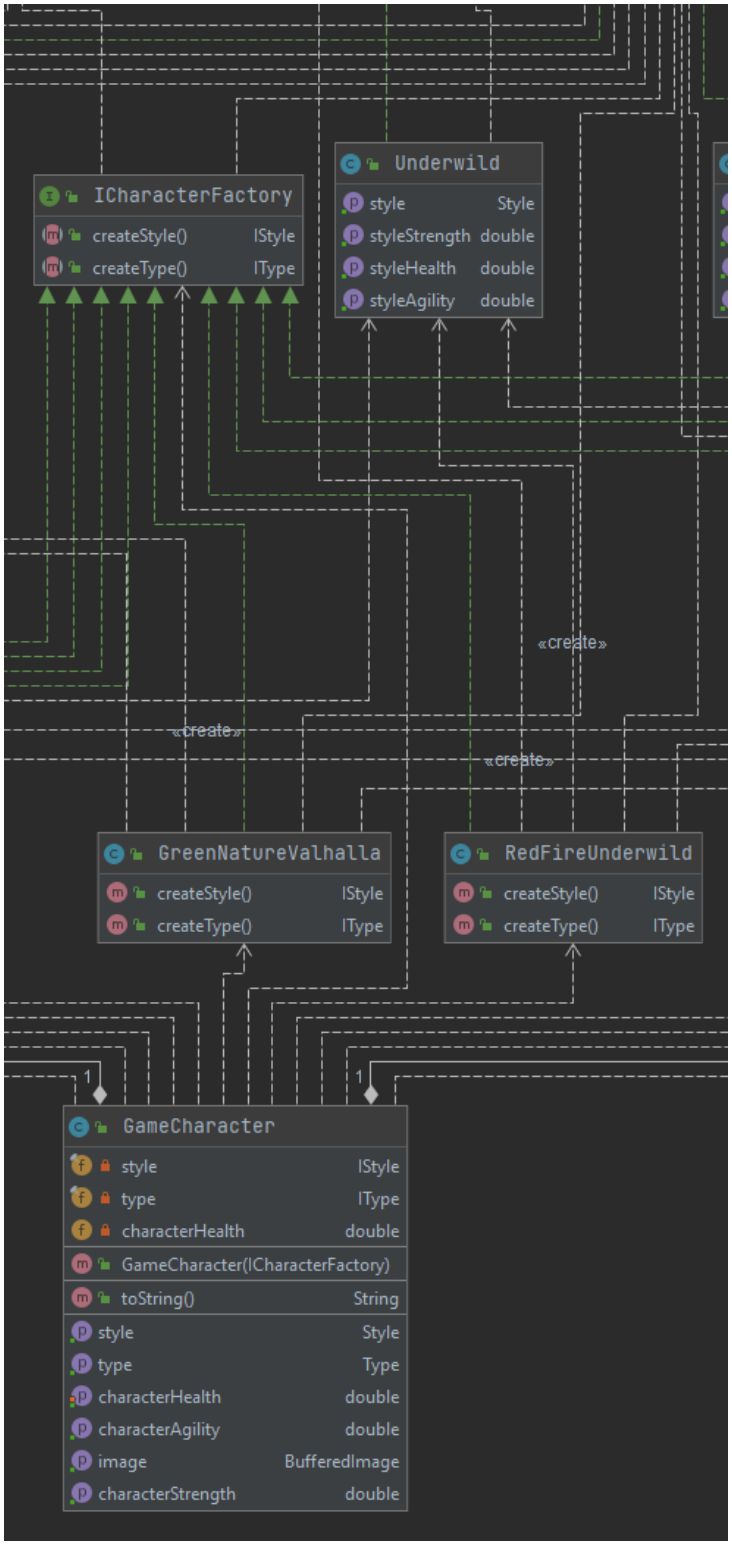
The types and styles of characters are expressed by an ENUM. Using them, type and style interfaces are defined. they describe strength, agility and health methods. *(The diagrams below are available in the files.)*



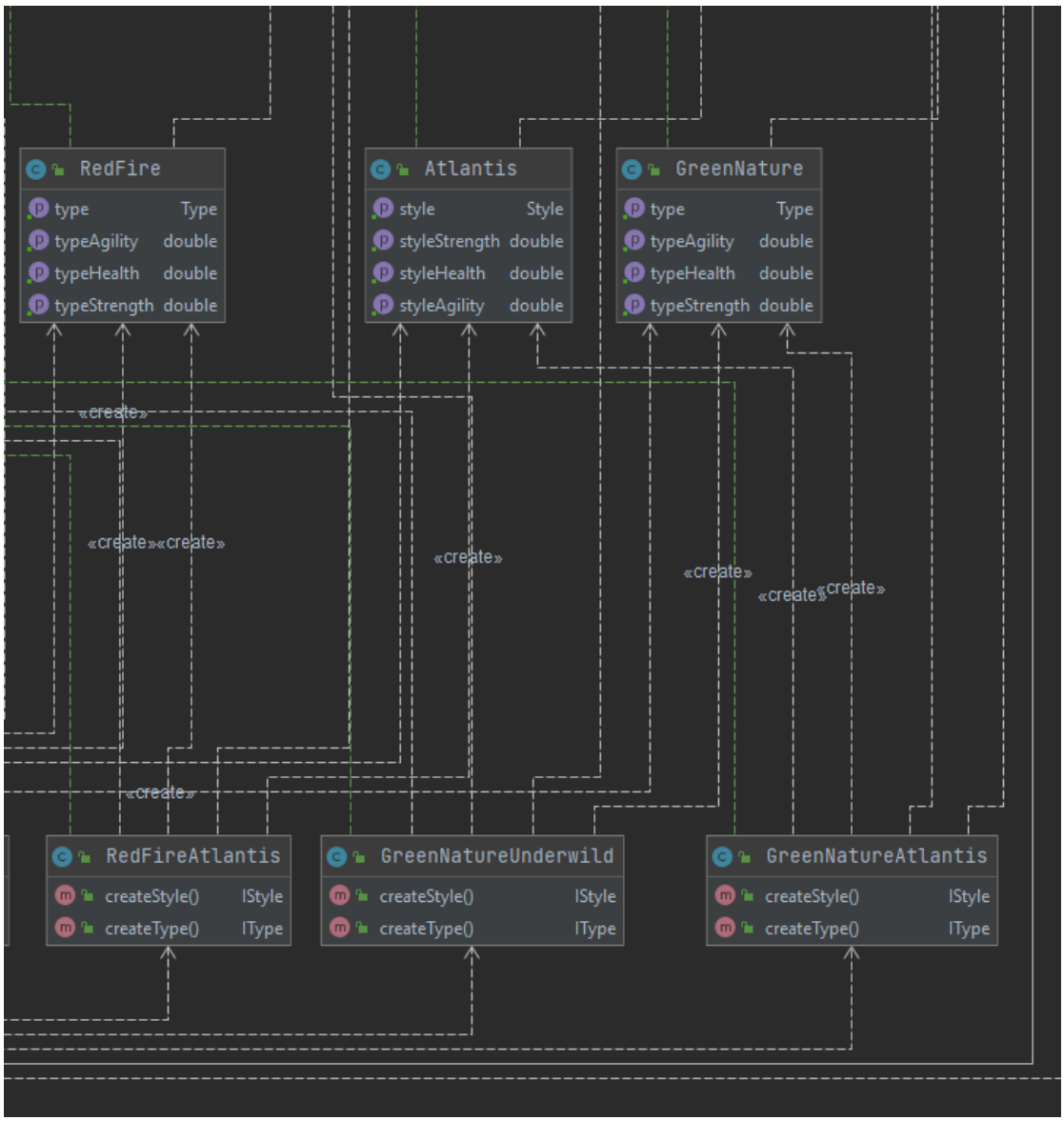

While RedFire, GreenNature, BlueIce classes implement the IType interface. Valhalla, Atlantis, and Underwild implement the IStyle interface.

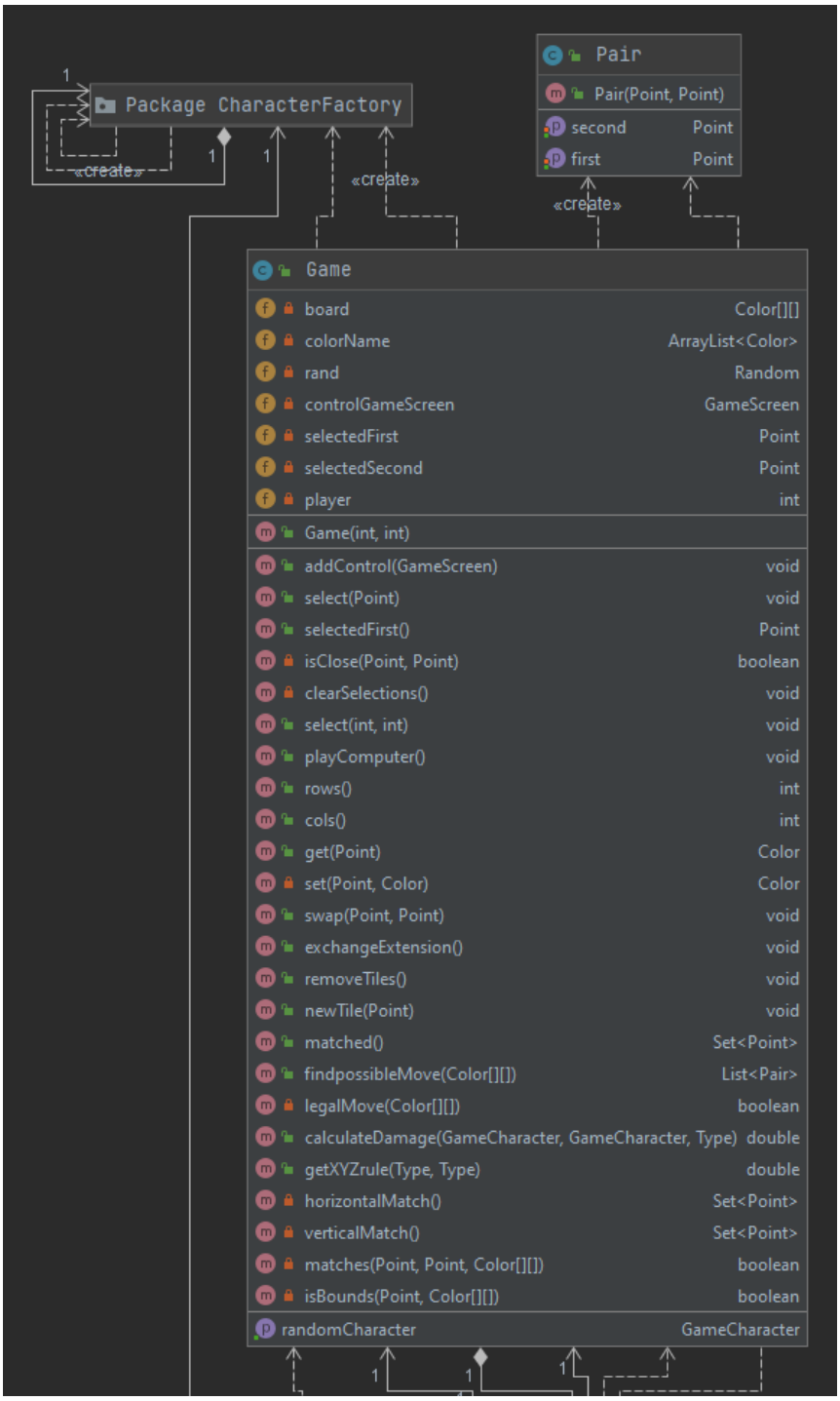

BlueIce
type Type
typeAgility double
typeHealth double
typeStrength double
Valhalla
style Style
styleStrength double
styleHealth double
styleAgility double
«create»
«create»
«create»
«create»
«create»
«create»
«create»
BlueIceValhalla
createStyle() IStyle
createType() IType
BlueIceUnderwild
createStyle() IStyle
createType() IType
RedFireValhalla
createStyle() IStyle
createType() IType
BlueIceAtlantis
createStyle() IStyle
createType() IType

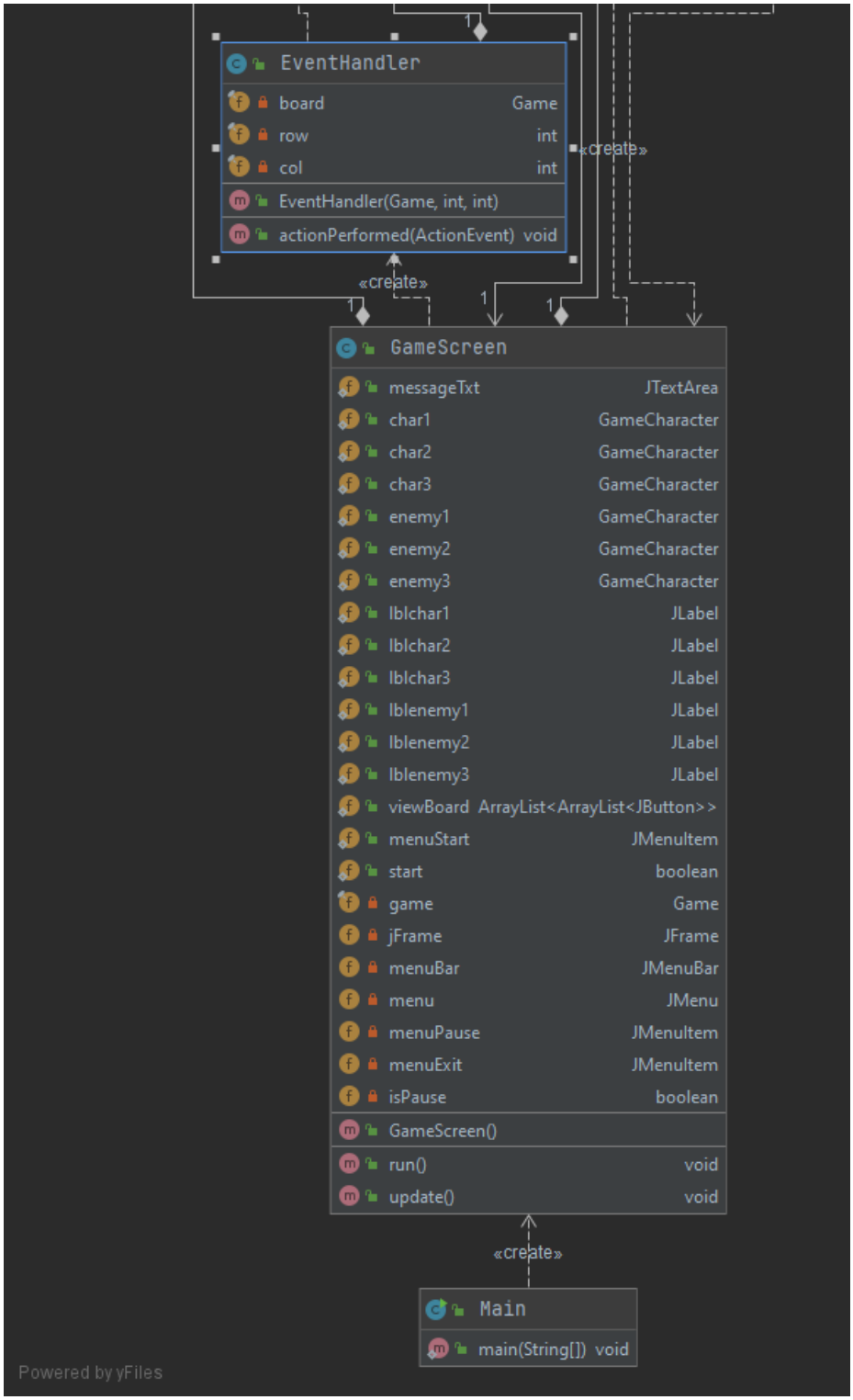
ICharacterFactory is an interface that implements the abstract factory design pattern. and defines 2 methods. The Game Character class is the client class. It is a class for creating characters from any class that implements the ICharacter Factory class.



RedFireValhalla, GreenNatureValhalla, BlueIceValhalla, RedFireAtlantis, GreenNatureAtlantis, BlueIceAtlantis, RedFireUnderwild , GreenNatureUnderwild , BlueIceUnderwild these classes are the concentrated classes that implement the ICharacterFactory interface.




Package CharacterFactory
«create»
«create»
«create»
Pair
Pair(Point, Point)
second Point
first Point
Game
board Color[][]
colorName ArrayList<Color>
rand Random
controlGameScreen GameScreen
selectedFirst Point
selectedSecond Point
player int
Game(int, int)
addControl(GameScreen) void
select(Point) void
selectedFirst() Point
isClose(Point, Point) boolean
clearSelections() void
select(int, int) void
playComputer() void
rows() int
cols() int
get(Point) Color
set(Point, Color) Color
swap(Point, Point) void
exchangeExtension() void
removeTiles() void
newTile(Point) void
matched() Set<Point>
findpossibleMove(Color[][]) List<Pair>
legalMove(Color[][]) boolean
calculateDamage(GameCharacter, GameCharacter, Type) double
getXYZrule(Type, Type) double
horizontalMatch() Set<Point>
verticalMatch() Set<Point>
matches(Point, Point, Color[][]) boolean
isBounds(Point, Color[][]) boolean
randomCharacter GameCharacter


EventHandler
board Game
row int
col int
EventHandler(Game, int, int)
actionPerformed(ActionEvent) void
«create»
«create»
«create»
1
1
1
1
GameScreen
messageTxt JTextArea
char1 GameCharacter
char2 GameCharacter
char3 GameCharacter
enemy1 GameCharacter
enemy2 GameCharacter
enemy3 GameCharacter
lblchar1 JLabel
lblchar2 JLabel
lblchar3 JLabel
lblenemy1 JLabel
lblenemy2 JLabel
lblenemy3 JLabel
viewBoard ArrayList<ArrayList<JButton>>
menuStart JMenuItem
start boolean
game Game
jFrame JFrame
menuBar JMenuBar
menu JMenu
menuPause JMenuItem
menuExit JMenuItem
isPause boolean
GameScreen()
run() void
update() void
Main
main(String[]) void
Powered by yFiles