HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

GÖKHAN ÖZELOĞLU
21627557

BEJEWELED

2.1. SOFTWARE USAGE

This program runs 2 in different ways.
1- `$ java Main`
*User can run the program and game is played on console until the player enter **E** character from the keyboard.
*If the user enter **E**, the user also enter own name to print out the rank result.
*Finally, "*leaderboard.txt*" file is updated.

2- ***$ java Main < test_case.txt***
*Program can run as redirecting on command line. This provides that user do not need to enter the coordinates on run time.
*Program reads the coordinates from predefined file.

test_case.txt : This file includes coordinates, **E** character and name of the player.
*This file should not have any empty line inside of the file. If there is empty lines, program terminates the program immediately.

gridMap.txt : This file includes game's map.

leaderboard.txt : This file includes all players who are already played this game and their scores. Before stops the game, this file is updated automatically.

2.2. ERROR MESSAGES

This game have 2 different type of error messages. One of them occurs if the player choose unmatched coordinate and the other one occurs if the player choose a coordinate which is out of the grid map.

Other built-in errors can occur while reading files. If the program cannot found the files for reading, error messages prints out the screen.

3. SOFTWARE DESIGN NOTES

3.1. Description of the Program

3.1.1.Problem

The problem is parsing the game in subclasses and creating jewels from abstract class.

3.1.2. Solution

I defined two different abstract classes in my solution. The first one is **Jewel** and the other one is **Search**. **Jewel** class is extended by jewel types. Also, **Search** class is extended by searching directions. These classes have some common methods and fields to reuse in subclasses. So, this approach provides me that not need to initialize the same methods and variables in every classes. We have 9 different jewel types in this assignment but I defined 10 different classes for each of them. One extra class was defined to fill empty these places with objects after deleting. So, I thought that empty object is a jewel. All of jewel type's classes have **shift()** method which returns updated grid map. This method controls the triplet and if there is a triplet in given coordinates, it updates the grid map, prints out the outputs and returns the grid map. This method calls different methods from **Search** classes sub-classes. Firstly, searches the triplets with calling method. Initializes the new coordinates.. After that, controls the triplet. If there is triplet calls the **move()** method from **Search** class to shift and delete the grid map. If there is not triplet, prints out the error message.

Moreover, **Search** class is extended by 4 different classes according to their searching directions. This classes are specialized according to type of searching jewel. One of these methods searches for **Diamond**, **Square** and **Triangle**, another methods searches for *math operators*, and third methods searches for **Wild card**. I have to define 3 different methods because there is not common attributes for searching. But there is only one shifting-deleting method in every classes because there is not any differences among all jewels while shifting them. Nevertheless, I could not define only one in abstract class because some of jewels are shifting in horizontal, some of them are shifting in vertical and some of them are shifting in diagonal.

Furthermore, program stores the grid map in **Grid** class. Also, this class has an method which calls the **shift** method from jewel's classes. I have filled grid map with jewel objects. So, this provides me that not writing the lots of if-else statements. Program finds the object with accessing given coordinates and calls this objects **shift()** method. If the player enters the coordinates which is out of the map, program prints out the error messages. I controlled this situation with ***try-catch*** block. After updating grid map, **shift()** method returns grid map.

Also, I stored the players which are in ***leaderboard.txt*** file in **LeaderBoard** class. This class implements **Comparable** interface class to find rank of the player. I implemented **compareTo()** method to sort array list which stores players. This class have some of helper methods to print out the final message for player's rank. In order to find player's rank, I used ***binarySearch()*** method. Also, I updated *leaderboard.txt* file before program terminates. If the other player wants to play this game, this new player's leaderboard file is updated file.

OPERATION CLASS

void readGridFile() : This method reads the grid map and stores map in 2D array list with jewel objects. All of jewel are added into ***gridMapList*** as an jewel object.

void readLeaderBoardFile() : This method reads the **leaderboard.txt** file line by line. Splits the line by space and adds the players as an object into ***leaderBoardList***.

void readTestCaseFile() : This method takes the coordinates and calls the **findPlace()** method to search and shift. If the user enters coordinate, takes these coordinate and split it by space. Converts the integer type. This taking coordinate from user continues until the user enter **E** on keyboard. After entering **E**, the user enters the own name and sees the rank result on the screen. When **updateLeaderBoardFile()** method calls, program terminates immediately.

void printMenuMessage() : This method prints out the menu message.

void updateLeaderBoardFile() : This method updates the *leaderboard.txt* file with writing on file.

GRID CLASS

void addNewLine() : This method adds the lines in 2D array list as a sub-array.

void printGrid() : This method prints out the grid map wherever it is needed.

void findPlaces() : Calls the **shift()** methods which given coordinates. There is no need to use if-else statements because every element is object in 2D array list.

JEWEL CLASS(Abstract)

This class have 2 different variables in field. These are ***symbol*** and ***score***. These are defined in ***protected*** type because these are common attributes and will being used in sub-classes.
There are 2 accessor methods to access in different classes to protect encapsulation.
Also, there is one "abstract" method to concrete in sub-classes.

BACKSLASH CLASS

* This class is extended by Jewel class.
* An ***leftDiagonal*** object is defined in field to call methods from **LeftDiagonal** class.
* Also, there is a constructor to create new objects from this class. Assigns the symbol and score its own variables.

ArrayList<ArrayList<Jewel>> shift() : This method return 2D array list after updating grid map. Firstly, calls the **findMathCoordinates()** method from **LeftDiagonal** class. Assigns the coordinates into **coordinates** as an array. Then, controls the triplets. If there is triplets, calls the **move()** method to shifting operation. Adds the score into player. If the program cannot find any triplet, just prints out the error message on the screen.

DIAMOND CLASS

* This class extended by Jewel class.
* An ***leftDiagonal*** and ***rightDiagonal*** objects are defined in field to call search and move methods from these 2 classes.
* Also, there is a constructor to create new objects from this class. Assigns the symbol and score its own variables.

ArrayList<ArrayList<Jewel>> shift() : This method controls the triplets with calling ***findCoordinates()*** methods from **LeftDiagonal** and **RightDiagonal** classes. Firstly, calls from the **LeftDiagonal** class. If the program cannot find any triplets, execution continues with calling the same method from **RightDiagonal** class. The program prints out the error message if these methods cannot find triples. Otherwise, calls the **move()** method from the class which is found triplet.

EMPTY CLASS

* This class is created to concrete the empty places with objects. Actually, there is not any operation in this class. ***shift()*** method prints out the error message and grid map

---

MINUS CLASS

* This class extended by Jewel class.
* There is an object declaration on field. This object is being used for calling methods from **Horizontal** class.
* Also, there is an constructor to create object.

* ***shift()*** method, firstly, calls the ***findMathCoordinates()*** method to control triplet. If there is triplet in given coordinate, the program calls the ***move()*** method from **Horizontal** class. In non-triplet situation, the program just prints out the error message on the screen.

PLUS CLASS

* This class extended by Jewel class.
* There are 2 object declaration on field. These objects is being used for calling methods if necessary.
* Firstly, the program controls the triplet. If triplet is found in horizontal, calls the move method from **Horizontal** class. Otherwise, program controls on the vertical direction. If triplet is found on vertical, ***move()*** method calls from the **Vertical** class. If the program cannot find vertical and horizontal directions, error message is being prints out.

SLASH CLASS

* This class extended by Jewel class.
* There is an object declaration in field. This object is being used for calling methods from **RightDiagonal** class.
* Also, there is an constructor to create object.
* **shift()** method calls the ***findMathCoordinates()*** method to control triplet. If there is triplet in given coordinate, the program calls the ***move()*** method from **RightDiagonal** class. In non-triplet situation, the program just prints out the error message on the screen.

SQUARE CLASS

* This class extended by Jewel class.
* There is an object declaration in field. This object is being used for calling methods from **Horizontal** class.
* Also, there is an constructor to create object.

ArrayList<ArrayList<Jewel>> shift() : This method calls the ***findCoordinates()*** method from **Horizontal** class. If there is a triplet in given coordinates, adds the player's score with using ***addScore()*** method. After that, calls the ***move()*** method to shift grid map. If the program cannot find any triplet, prints out the error message on the screen.

STRAIGHTLINE CLASS

*This class extended by Jewel class.
* There is an object declaration in field. This object is being used for calling method from **Vertical** class.
* Also, there is an constructor to create object.

ArrayList<ArrayList<Jewel>> shift() : This method controls the triplet situations with calling **Vertical** class. Firstly, the program calls the ***findMathCoordinates()*** method and controls the triplet. If triplet can being found in given coordinates, ***move()*** method is being called. If there is not triplet, the program just prints out the error message on the screen.

TRIANGLE CLASS

*This class extended by Jewel class.
*There is an object declaration in field. This object is being used for calling method from **Vertical** class.
*Also, there is a constructor to create object

ArrayList<ArrayList<Jewel>> shift() : This method calls the ***findCoordinates()*** method from **Vertical** class. If there is a triplet in given coordinates, ***move()*** method is being called to shift grid map. If the program cannot find any triplet, prints out the error message on the screen.

WILDCARD CLASS

*This class extended by Jewel class.
*There is an object declarations in field. These objects are being used for calling methods from **Search** class's sub-classes.
*Also, there is a constructor to create object.

ArrayList<ArrayList<Jewel>> shift() : This method calls the ***findWildCardCoordinates()*** methods from other classes in order. The order is, vertical, horizontal, left diagonal and right diagonal. If the program can manage to find triplet one of these directions, stops the searching immediately. In non-triplet situation, the program prints out the error message. If triplet is being found, ***move()*** method calls and grid map is being shifted.

SEARCH CLASS(Abstract class)

* This class is being extended by 4 different classes.
* The methods which names start with ***initialize-*** initialize the array lists to compare jewels while looking for triplets. Calling these method in necessary situations gives an advantage is that not wasting the memory.
* Also, there are 3 different abstract classes. These are concrete in sub-classes.

HORIZONTAL CLASS

int[] findCoordinates() : This method searches the triplet firstly in left part of given coordinate for the jewel. If the program cannot found triplet in left part, looks at right part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findMathCoordinates() : This method searches the triplet firstly in left part of the given coordinate for math operators. If the program cannot found triplet in left part, looks at right part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findWildCardCoordinates() : This method searches the triplet firstly in left part of the given coordinate for wild card. If the program cannot found triplet in left part, looks at right part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

ArrayList<ArrayList<Jewel>> move() : Shifting-deleting operations occur in this method with *for* loop. One object sets the other object. When for loop terminates, the top of the map's is being put *Empty* objects.

LEFTDIAGONAL CLASS

int[] findCoordinates() : This method searches the triplet firstly in left-up part of given coordinate for the jewel. If the program cannot found triplet in left-up part, looks at right-down part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findMathCoordinates() : This method searches the triplet firstly in left-up part of the given coordinate for math operators. If the program cannot found triplet in left-up part, looks at right-down part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findWildCardCoordinates() : This method searches the triplet firstly in left-up part of the given coordinate for wild card. If the program cannot found triplet in left part, looks at right-up part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

ArrayList<ArrayList<Jewel>> move() : Shifting-deleting operations occur in this method with *for* loops. One object sets the other object. When for loop terminates, the top of the map's is being put **Empty** objects.

RIGHTDIAGONAL CLASS

int[] findCoordinates() : This method searches the triplet firstly in right-up part of given coordinate for the jewel. If the program cannot found triplet in right-up part, looks at left-down part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findMathCoordinates() : This method searches the triplet firstly in right-up part of the given coordinate for math operators. If the program cannot found triplet in right-up part, looks at left-down part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findWildCardCoordinates() : This method searches the triplet firstly in right-up part of the given coordinate for wild card. If the program cannot found triplet in right-up part, looks at left-down part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

ArrayList<ArrayList<Jewel>> move() : Shifting-deleting operations occur in this method with *for* loops. One object sets the other object. When for loop terminates, the top of the map's is being put **Empty** objects.

VERTICAL CLASS

int[] findCoordinates() : This method searches the triplet firstly in upper part of given coordinate for the jewel. If the program cannot found triplet in upper part, looks at lower part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findMathCoordinates() : This method searches the triplet firstly in upper part of the given coordinate for math operators. If the program cannot found triplet in upper part, looks at lower part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

int[] findWildCardCoordinates() : This method searches the triplet firstly in upper part of the given coordinate for wild card. If the program cannot found triplet in upper part, looks at lower part of the given coordinate. This searching operation provides with 2 while loops. If the program can find triplet, it returns the coordinate. Otherwise, returns the coming coordinate again.

ArrayList<ArrayList<Jewel>> move() : Shifting-deleting operations occur in this method with *for* loop. One object sets the other object. When for loop terminates, the top of the map's is being put **Empty** objects.

LEADERBOARD CLASS

*This class implements ***Comparable*** interface class for sorting finding rank of the player operations.
* Player's name and score holds on field in private. Also, there is an array list to store players as an object. This array list's elements are in ***LeaderBoard*** type.

void addScore() : This method adds the score to total score.

int findLowerPlayer() : This method finds the previous player. It returns the index of the previous player. If there is not previous player, the method returns the player's index.

int findHigherPlayer() : This method finds the next player. It returns the index of the next player. If there is not next player, the method returns the player's index.

void printRankMessage() : This method prints out the ranking message when finish the game.

int compareTo() : This method override from ***Comparable*** class. I re-defined this method according to player's scores.

3.1.3. Algorithm

Main.java

- Program starts in this class.
- Creates an object from **Operation** class to read ***gridMap.txt***, ***leaderboard.txt*** and ***test_case.txt*** files in order.
- Program calls the ***readGridFile()*** method to game's map.

    ➢ **readGridFile()** reads the **gridMap.txt** file line by line with using ***BufferedReader*** class's object.
    ➢ Then, splits the line by space character.
    ➢ After that creates objects according to letter from Jewel's subclasses.
    ➢ One line holds on an array list which it is created every new line. When finished the reading line, calls the ***addNewLine()*** to add 2D array list.

- After reading grid map, program comes back to Main class.
- Program calls the ***readLeaderBoardFile()*** to read players and their scores in ***leaderboard.txt*** file.

    ✔ ***readLeaderBoardFile()*** method reads players and their scores line by line.
    ✔ Each lines read with using *BufferedReader* object.
    ✔ After reading a line, program splits the line by space.
    ✔ Finally, calls the **addNewPlayer()** method to add new players as a object which is related to **LeaderBoard** class.

- Actually, program starts at this point that comes back to **Main** class. The first two methods prepare the program to start processing.
- Program calls the ***readTestCaseFile()*** method from Operation class.

  - ➔ Program uses the redirecting in this case. If the user starts the program like ***$ java Main < test_case.txt***, program reads the file line by line.
  - ➔ This reading operation implementing with ***Scanner*** object.
  - ➔ Takes the coordinates and assigns them to **x**, **y** variables with converting Integer from String.
  - ➔ After that calls the ***findPlace()*** from **Grid** class.

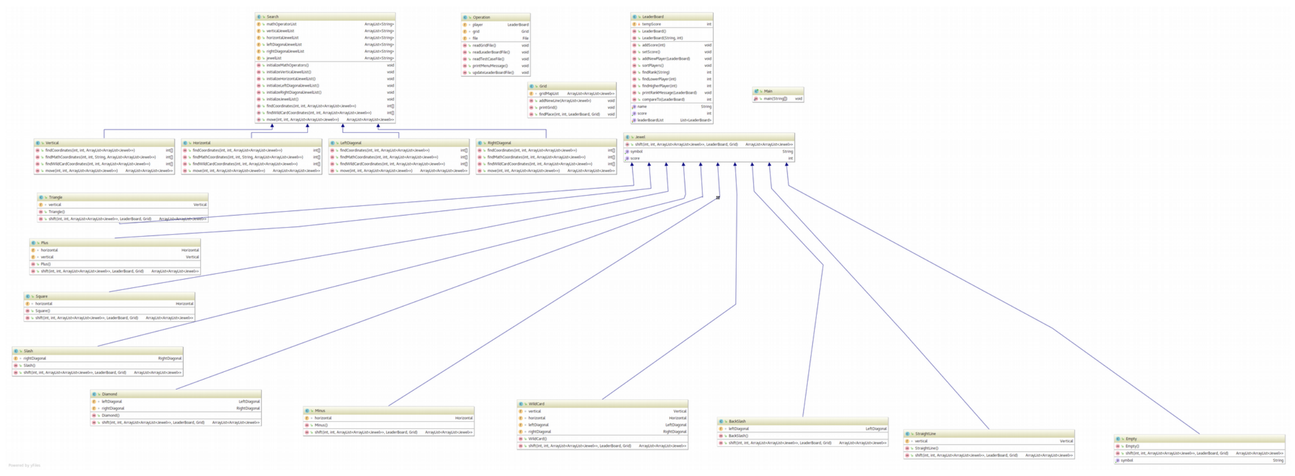
    - ■ This ***findPlace()*** method just calls the ***shift()*** method. I need to implement **Grid** class because program stores the map in this class.
    - ■ If the user enters the coordinates which are out of the map, program gives the error message. This situation controls with ***try-catch*** block.
    - ■ Program calls the ***shift()*** method from the classes which are extended by ***Jewel*** abstract class. So, this gives me that making program dynamically.
      - ✗ Program jumps to ***shift()*** method.
      - ✗ This method implements in every subclasses according to jewel’s searching and shifting directions.
      - ✗ Inside of the ***shift()*** method, program calls the methods which are controls the triplets and assigns the coordinates after controlling triplets.
      - ✗ After assignment the new coordinates, program controls that there is any triplet matches.
      - ✗ If the program can find triplet according to given coordinates, calls the ***move()*** method from the **Search** classes subclasses.
      - ✗ If there is no triplet in given coordinate, the program prints out the error message on the screen.
      - ✗ These all of methods return the grid map as being updated.

    - ■ After updating the grid map, program comes back to **Grid** class.

  - ➔ Then, program comes back to **Operation** class.
  - ➔ Reading test_case file or playing the game on the consol continue until the user enters **E** character.
  - ➔ Finally, program gets the name of player and prints out the rank as a message on the screen.
  - ➔ Before exit the program, ***leaderboard.txt*** file is updated.

HOW TO ADD NEW JEWEL INTO THE GAME?

I tried to write this game in the best way to add new jewels on my program easily. The person who wants to add new jewel in my code need to create a class which is extends to **Jewel** abstract class. Also, the person must override the ***shift()*** method according to jewel’s searching and shifting directions. If this new jewel’s searching and shifting directions are not being initialized yet, the person must implement these method with creating new sub-class of **Search** abstract class. If searching-shifting methods are already defined, these methods calls from the new jewel’s class. So, other classes is not being affected in this way.


Search
mathOperatorList ArrayList<String>
verticalJewelList ArrayList<String>
horizontalJewelList ArrayList<String>
leftDiagonalJewelList ArrayList<String>
rightDiagonalJewelList ArrayList<String>
jewelList ArrayList<String>
initializeMathOperators() void
initializeVerticalJewelList() void
initializeHorizontalJewelList() void
initializeLeftDiagonalJewelList() void
initializeRightDiagonalJewelList() void
initializeJewelList() void
findCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findWildCardCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
move(int, int, ArrayList<ArrayList<Jewel>>) ArrayList<ArrayList<Jewel>>
Operation
player LeaderBoard
grid Grid
file File
readGridFile() void
readLeaderBoardFile() void
readTestCaseFile() void
printMenuMessage() void
updateLeaderBoardFile() void
LeaderBoard
tempScore int
LeaderBoard()
LeaderBoard(String, int)
addScore(int) void
setScore() void
addNewPlayer(LeaderBoard) void
sortPlayers() void
findRank(String) int
findLowerPlayer(int) int
findHigherPlayer(int) int
printRankMessage(LeaderBoard) void
compareTo(LeaderBoard) int
name String
score int
leaderBoardList List<LeaderBoard>
Grid
gridMapList ArrayList<ArrayList<Jewel>>
addNewLine(ArrayList<Jewel>) void
printGrid() void
findPlace(int, int, LeaderBoard, Grid) void
Main
main(String[]) void
Vertical
findCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findMathCoordinates(int, int, String, ArrayList<ArrayList<Jewel>>) int[]
findWildCardCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
move(int, int, ArrayList<ArrayList<Jewel>>) ArrayList<ArrayList<Jewel>>
Horizontal
findCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findMathCoordinates(int, int, String, ArrayList<ArrayList<Jewel>>) int[]
findWildCardCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
move(int, int, ArrayList<ArrayList<Jewel>>) ArrayList<ArrayList<Jewel>>
LeftDiagonal
findCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findMathCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findWildCardCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
move(int, int, ArrayList<ArrayList<Jewel>>) ArrayList<ArrayList<Jewel>>
RightDiagonal
findCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findMathCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
findWildCardCoordinates(int, int, ArrayList<ArrayList<Jewel>>) int[]
move(int, int, ArrayList<ArrayList<Jewel>>) ArrayList<ArrayList<Jewel>>
Jewel
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
symbol String
score int
Triangle
vertical Vertical
Triangle()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
Plus
horizontal Horizontal
vertical Vertical
Plus()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
Square
horizontal Horizontal
Square()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
Slash
rightDiagonal RightDiagonal
Slash()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
Diamond
leftDiagonal LeftDiagonal
rightDiagonal RightDiagonal
Diamond()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
Minus
horizontal Horizontal
Minus()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
WildCard
vertical Vertical
horizontal Horizontal
leftDiagonal LeftDiagonal
rightDiagonal RightDiagonal
WildCard()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
BackSlash
leftDiagonal LeftDiagonal
BackSlash()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
StraightLine
vertical Vertical
StraightLine()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
Empty
Empty()
shift(int, int, ArrayList<ArrayList<Jewel>>, LeaderBoard, Grid) ArrayList<ArrayList<Jewel>>
symbol String
Powered by yFiles

NOTE: There are lots of classes and their methods. So, the UML's size is not enough good for readability. I am sorry for this but you can make zoom while reading.