CS 2210 — Data Structures and Algorithms
Assignment 2: Board Game with Hash Tables
Due Date: October 17 at 11:59 pm
Total marks: 20

1 Overview

For this assignment you need to write a Java program that plays the following board game. Two players take turns putting colored tiles on a square gameboard of size $n \times n$. The goal is for a player to place n of their tiles in the same row, column, or diagonal of the board, similarly as in the game of tic tac toe. However, at least k positions on the board need to be empty, where k is a value specified when running the program. If while playing the game there are exactly k empty positions on the board and no player has won then the players take turns sliding onto one of the empty positions of the board one of their tiles adjacent to that empty position; tiles can be slid horizontally, vertically, or in diagonal. If the number of empty positions is k and on their turn a player does not have any tiles adjacent to the empty positions of the board the game ends in a draw.

Your program will play against a human opponent. The human player uses blue tiles and always starts the game. The computer uses orange tiles. Empty positions on the board are green.

Figure 1 (a) shows a possible set of tiles on a board of size 3×3 where $k = 1$. If the next player to move is the human then the human player can slide any of the tiles in positions 2, 3, or 9 into position 6. If, for example, the tile in position 3 is moved to position 6 the board will look as shown in Figure 1 (b).

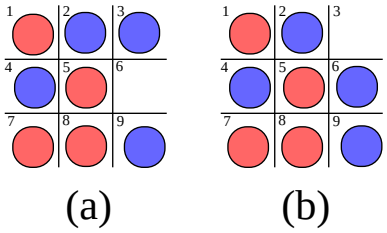

Figure 1: Board game.

You will be given code for displaying the gameboard on the screen and for playing the game. However, there are several Java classes that you need to implement as described in the next sections. If you are interested in knowing how the algorithm for playing the above game works additional information is posted in the course's website.

2 Board Configurations

In each turn the computer examines all possible moves and selects the best one; to do this, each possible move is assigned a score. Your program will use the following 4 scores for moves:

- 0: if a move ensures that the human player wins
- 1: if after a move is selected it is not clear who will win
- 2: if a move leads to a draw
- 3: if a move guarantees that the computer wins the game.

For example, suppose that the gameboard looks like the one in Figure 2(a). If the computer plays in position 8, the game will end in a draw (see Figure 1(b)), so the score for the computer playing in

position 8 is 2. Note that the board displayed in Figure 2(b) is a draw because now it is the turn of the human player, and no blue tile can be moved to the empty spot. We say that the *score for the board configuration* in Figure 1(b) is 2, where a *board configuration* is simply the positioning of the tiles on the gameboard. Similarly, the score for the board configuration in Figure 1(c) is 3, since in this case the computer wins the game.

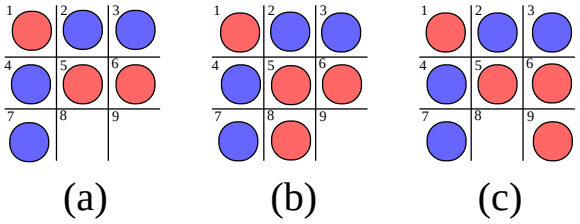

Figure 2: Board configurations.

3 Classes to Implement

You are to implement at least 4 Java classes: `Configuration.java`, `HashDictionary.java`, `DictionaryException.java`, and `BoardGame.java`. You can implement more classes if you need to. You must write all the code yourself. You cannot use code from the textbook, the Internet, or any other sources. You **cannot** use Java's `Hashtable` class or `hashCode()` method.

3.1 Class `Configuration`

This class stores the data that each entry of `HashDictionary` will contain. An object of this class stores a board configuration and its integer score, as explained above. Each board configuration will be represented as a string as follows. A blue tile is represented with the character 'b', an orange tile with 'o', and a green empty space with 'g'. To form a string for a given board configuration we concatenate the characters corresponding to the tiles and empty positions on the board starting at the upper left position and moving top to bottom and left to right. For example, for the configurations in Figure 2, their string representations are "`obbbogbog`", "`obbboobog`", and "`obbbogboo`".

For this class, you must implement all and only the following `public` methods:

- `public Configuration(String config, int score)`: A constructor which returns a new `Configuration` object with the specified configuration string and score. The string `config` will be used as the key attribute for every `Configuration` object.
- `public String getStringConfiguration()`: Returns the configuration string stored in a `Configuration` object.
- `public int getScore()`: Returns the score stored in a `Configuration` object.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods (i.e. not accessible to other classes).

3.2 Class `HashDictionary`

This class implements a dictionary using a hash table with separate chaining. You will decide on the size of the table, keeping in mind that the size of the table must be a prime number. A table of size between 5000-10000, should work well.

You must design your hash function so that it produces few collisions. A bad hash function that induces many collisions will result in a lower mark. As mentioned above, **you cannot use** Java's `hashCode()` method in your hash function.

For this class, you must implement all the `public` methods in the following interface:

```
public interface DictionaryADT {
    public int put(Configuration pair) throws DictionaryException;
    public void remove(String config) throws DictionaryException;
    public int getScore(String config);
}
```

The descriptions of these methods follows:

- `public int put(Configuration data) throws DictionaryException`: Inserts the given `Configuration` object referenced by `data` in the dictionary. This method must throw a `DictionaryException` (see below) if the configuration string stored in `data`, is already in the dictionary.

  You are required to implement the dictionary using a hash table with separate chaining. To determine how good your design is, we will count the number of collisions produced by your hash function. Method `put` will return the value 1 if the insertion of the object referenced by `data` into the hash table produces a collision, and it will return the value 0 otherwise. In other words, if for example your hash function is h(key) and the name of your table is T, this method will return the value 1 if the list in $T[h($`data.getStringConfiguration()`$)]$ already stores at least one element; it will return 0 if the list in $T[h($`data.getStringConfiguration()`$)]$ was empty before the insertion.

- `public void remove(String config) throws DictionaryException`: Removes the entry with configuration string `config` from the dictionary. Must throw a `DictionaryException` (see below) if the configuration is not in the dictionary.

- `public int getScore(String config)`: A method which returns the score stored in the dictionary for the given configuration string, or -1 if the configuration string is not in the dictionary.

Since your `HashDictionary` class must implement all the methods of the `DictionaryADT` interface, the declaration of your method should be as follows:

```
public class HashDictionary implements DictionaryADT
```

You can download the file `DictionaryADT.java` from the course's website. The only other public method that you can implement in the `HashDictionary` class is the constructor method, which must be declared as follows

```
public HashDictionary(int size)
```

this initializes a dictionary with an empty hash table of the specified size.

You can implement any other methods that you want to in this class, but they must be declared as `private` methods (i.e. not accessible to other classes).

Hint. You might want to implement a class `Node` storing an object of type `Configuration` and a link object of type `Node` to construct the linked list associated to an entry of the hash table. You do not need to follow this suggestion. You can implement the lists associated with the entries of the table in any way you want.

3.3 Class `DictionaryException`

This is just the class implementing the class of exceptions thrown by the `put` and `remove` methods of `HashDictionary`.

3.4 Class `BoardGame`

This class implements all the support methods needed by the algorithm that plays the board game. For details on this algorithm see the additional documentation posted on the course's website. The constructor for this class must be as follows

```
public BoardGame (int board_size, int empty_positions, int max_levels)
```

The first parameter specifies the size of the gameboard, the second one is the number of positions on the board that must remain empty, and the third one specifies the playing quality of the program (the higher this value is the better the program will play, but the slower it will be; when you test your program use values between 3 and 5 so the program plays OK and it is not too slow).

This class must have an instance variable called `gameBoard` of type `char[][]` to store the gameboard. This variable is initialized inside the above constructor method so that every entry of `gameBoard` stores a 'g'. As the game is played, every entry of `gameBoard` will store one of the characters 'b', 'o', or 'g'. This class must also implement the following public methods.

- `public HashDictionary makeDictionary()`: returns an empty `HashDictionary` of the size that you have selected.
- `public int isRepeatedConfig(HashDictionary dict)`: This method first represents the content of `gameBoard` as a string as described above; then it checks whether the string representing the `gameBoard` is in `dict`: If it is, this method returns its associated score; otherwise it returns the value -1.
- `public void putConfig(HashDictionary dict, int score)`: This method first represents the content of `gameBoard` as a string as described above; then it inserts this string and its `score` in `dict`.
- `public void savePlay(int row, int col, char symbol)`: This method stores `symbol` in `gameBoard[row][col]`.
- `public boolean positionIsEmpty (int row, int col)`: This method returns true if `gameBoard[row][col]` is 'g'; otherwise it returns false.
- `public boolean tileOfComputer (int row, int col)`: This method returns true if `gameBoard[row][col]` is 'o'; otherwise it returns false.
- `public boolean tileOfHuman (int row, int col)`: Returns true if `gameBoard[row][col]` is 'b'; otherwise it returns false.
- `public boolean wins (char symbol)`: Returns true if there are n adjacent tiles of type `symbol` in the same row, column, or diagonal of `gameBoard`, where n is the size of the gameboard.
- `public boolean isDraw(char symbol, int empty_positions)`: Returns true if the game configuration corresponding to `gameBoard` is a draw assuming that the player that will perform the next move uses tiles of the type specified by `symbol`. The second parameter is the number of positions of the gameboard that must remain empty.

  Remember that a game is a draw if no player has won and either:

  - `empty_positions` $= 0$ and there are no empty positions left on the game board, or
  - `empty_positions` > 0 and none of the empty positions on the gameboard has a tile of the type specified by `symbol` adjacent to it.

- `public int evalBoard(char symbol, int empty_positions)`: Returns one of the following values:

  ▷ 3, if the computer has won, i.e. there are n adjacent 'o's in the same row, column, or diagonal of `gameBoard`.

  ▷ 0, if the human player has won, i.e. there are n adjacent 'b's in the same row, column, or diagonal of `gameBoard`.

  ▷ 2, if the game is a draw when the player that needs to make the next move uses tiles of the type specified by `symbol`. The second parameter is the number of positions of the gameboard that must remain empty.

  ▷ 1, if the game is still undecided, i.e. no player has won and the game is not a draw.

You can implement more methods in this class, if you want, but they must be declared as `private`.

4 Classes Provided

You can download classes `DictionaryADT.java`, `PosPlay.java` and `PlayGame.java` from the course's website. Class `PosPlay` is an auxiliary class used by `PlayGame` to represent plays. Class `PlayGame` includes the main method for your program. The program will be executed by typing

```
java PlayGame size empty_positions max_levels
```

where `size` is the size of the gameboard, `empty_positions` is the number of positions on the gameboard that must remain empty, and `max_levels` is a parameter specifying the playing quality of the program. Remember that the larger the value of `max_levels` is the better the program will play, but the slower it will be.

Class `PlayGame` also contains methods for displaying the gameboard on the screen and for reading the moves of the human player.

5 Testing your Program

We will perform two kinds of tests on your program: (1) tests for your implementation of the dictionary, and (2) tests for your implementation of `BoardGame`. For testing the dictionary we will run a test program called `TestDict` which performs a few simple tests to check whether your dictionary works as specified. We will supply you with a copy of `TestDict` so you can use it to test your implementation.

6 Coding Style

Your mark will be based partly on your coding style.

- Variable and method names should be meaningful and they must reflect their purpose in the program.
- Comments, indenting, and white spaces should be used to improve readability.

- No variable declarations should appear outside methods ("instance variables") unless they contain data which needs to be maintained in an object from call to call. In other words, variables which are needed only inside methods should be declared inside those methods.
- All variables declared outside methods ("instance variables") should be declared `private` (not `protected`), to maximize information hiding. Any outside access to these variables should be done with accessor methods (like `getScore()` for `Configuration`).

7 Marking

Your mark will be computed as follows.

- Program compiles, produces meaningful output: 2 marks.
- `HashDictionary` tests pass: 4 marks.
- `BoardGame` tests pass: 4 marks.
- Coding style: 2 marks.
- Hash table implementation: 4 marks.
- `BoardGame` program implementation: 4 marks.

8 Handing In Your Program

You must submit an electronic copy of your program. To submit your program, login to OWL and submit **your java** files from there. Please **do not** put your code in sub-directories. **Do not** compress your files or submit a .zip, .rar, .gzip, or any other compressed file. Only your .java files should be submitted.

Read the tutorials posted in the course's website to learn how to copy your program onto your GAUL account and how to test it there. Remember that the TA's will test your program on the computers of the Department. Please also read the tutorials about how to configure Eclipse to read command line arguments.

When you submit your program, we will receive a copy of it with a datestamp and timestamp. If you submit your program more than once please send me an email message to let me know. We will take the latest program submitted as the final version, and will deduct marks accordingly if it is late.