Corpus-based online Scrabble game

Kévin Dumoulin
Supervisor: Dr. Carl Vogel

April 1, 2012
## Contents

1 Game overview
   1.1 The game screen
   1.2 How to play
   1.3 Creating a game

2 Structure of the server
   2.1 The chosen programming language: Python
   2.2 Saving the game data
      2.2.1 Board
      2.2.2 Game
      2.2.3 Letter_bag
      2.2.4 Players
   2.3 The classes
      2.3.1 Letter
      2.3.2 LetterBag
      2.3.3 Rack
      2.3.4 Square
      2.3.5 Player
      2.3.6 Board
      2.3.7 Game
      2.3.8 GhostBoard

3 Structure of the client
   3.1 The chosen programming language: PHP/JavaScript
   3.2 Actions on a game
      3.2.1 Creating a game
      3.2.2 Joining a game
CONTENTS

3.2.3 Launching a game ........................................... xvii
3.2.4 Playing a game ........................................... xvii

4 Relation between the client & the server xviii
4.1 Use of CGI ...................................................... xviii
4.2 Sending data .................................................... xviii
4.3 Receiving data .................................................. xix

5 Deriving a letter bag from a corpora xx
5.1 Goal ............................................................ xx
5.2 The way of doing it .......................................... xx
5.3 Results ........................................................ xx

6 Playing a word xxiv
6.1 What is the direction of the word and are the letters correctly put? xxiv
6.2 What are the formed words and how much points do they cost? xxv
6.3 Are the words correct? ....................................... xxvi
6.4 The AI ........................................................ xxviii
6.5 Including teamplay & Multiple games handling ..................... xxx
List of Figures

1.1 Example of a game in progress ........................................ vi
1.2 Creation of a game ....................................................... vii

2.1 The game table .............................................................. ix
2.2 The players table .......................................................... x
2.3 Simplified schema of the classes ........................................ xi

5.1 English Scrabble & LOTR derived letter bags ..................... xxii
5.2 English Scrabble & English dictionary derived letter bags ...... xxiii

6.1 Board coordinates ......................................................... xxv
6.2 Letter browsing ........................................................... xxvi
6.3 Example of AI playing ................................................... xxix
Introduction

In this project, I was told to create an online Scrabble game with a particular interest on the development of a corpora based letter bag creation.

The entire project was designed in Python 3, PHP and Javascript with an SQL database to keep the data of the game, and can be found at the following address: http://scrabble-project.alwaysdata.net/scrabble_lobby.php
Chapter 1

Game overview

1.1 The game screen

The game screen is divided in 5 sections:

- The Scrabble Board used to put the letters on, and show the letters already placed.
- The report frame. Says to whose turn it is to play, if your last move is valid and how much points do you scored if it is.
- The rack, composed of the player’s letters and allowing him to exchange some of them or to send a move to the server after putting letters on the board.
- The timer frame which count how much time left the playing player has to play.
- The game information frame. This frame shows the used dictionary and letter bag type, and list the players in order of playing with their score. The player in bold is the one which is on the page, and the player with a ⇒ is the one whose it is the turn to play.

You can also pass your turn, reload the page or return to the lobby.

1.2 How to play

The official rules of scrabble are used here. To play a word, the player has to move his letters on the board to create one or more valid words. Each word formed is worth
several points depending on the value of the used letters and on the bonus cases such as letter x2, letter x3, word x2 and word x3. The middle square is considered as a word x2.

The player can also skip its turn or exchange some of his letters, both of these actions passing to the next player’s turn. If the countdown reaches 0, the player’s turn is automatically skipped.

In the case of placing a blank tile on the board, the player is asked to enter the letter he wants the tile to represent. If several characters are entered, only the first one is counted. To change the character of the blank tile, simply re-depose it on the board.

Each letter placed on the board by the player can be returned to the rack moving them to the right of the board.

When the letter bag is empty and each player has skipped his turn, the game stops and the player with the highest score wins.

1.3 Creating a game

Each game propose several options during their creation:
• Player name: the name of the first player (the creator of the game).

• Password: the password of the first player, which is also used as the password for launching the game after all players have joined it.

• Letter bag: you can choose a classic letter bag from one language, according to the official Scrabble letter values and frequencies, or to derive a letter bag from a corpora. If so, a second frame appears on the right of the screen to enter the address of a web page, or raw text used as the corpora.

• Dictionary: the dictionary used for this game. You can either choose an English, French or German dictionary.

• Number of players: a Scrabble game can handle from 2 to 4 players.

• Time limit: the time limit used can vary from 1 minute to 5 minutes.

Once the game is created, some players can join it and, when the game is full, the creator of the game can launch it and start to play.
Chapter 2

Structure of the server

2.1 The chosen programming language: Python

I chose Python 3 to develop this game for two main reasons: firstly, I was learning it, and developing a project is the best way to learn. Secondly, Python seemed to be the most practical language to use for this project, amongst the other programming languages I knew. It was clear, concise and well suited to manage the many data structures I had to use through an object-oriented design.

2.2 Saving the game data

To store the game data, I used an SQL database (with MySQL). This database is composed of the 4 following tables:

2.2.1 Board

The board table stores each letter put on the board. Each entry is defined by 3 attributes:

- \( x \): the X position on the board (integer from 0 to 14).
- \( y \): the Y position on the board (integer from 0 to 14). The couple \((x, y)\) forms the primary key of the table.
2.2.2 Game

The game table stores all the parameters of the game, including the players names.

Figure 2.1: The game table

<table>
<thead>
<tr>
<th>id</th>
<th>state</th>
<th>player_1</th>
<th>player_2</th>
<th>player_3</th>
<th>player_4</th>
<th>dictionary</th>
<th>language</th>
<th>nb_players</th>
<th>delay</th>
<th>last_move</th>
<th>pass</th>
<th>empty_bag</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2</td>
<td>James</td>
<td>Lars</td>
<td>Kirk</td>
<td>Rob</td>
<td>english</td>
<td>english</td>
<td>4</td>
<td>180</td>
<td>1332583706</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **id** (primary key): the ID of the game, although not really used as there can only be one game at a time.
- **state**: the state of the game, taking one of the following values: 0 (waiting for players), 1 (waiting for launch), 2 (in progress), 3 (finished).
- **player_X**: the 4 (or less) players names.
- **dictionary**: the language of the used dictionary.
- **language**: the language of the used letter bag. If the letter bag is derived from a corpora, then "corpora" is the value of this attribute.
- **nb_players**: the number of players.
- **delay**: the playing delay in seconds.
- **last_move**: the timestamp value corresponding to the last played move, used to know how much time remains for the current player to play.
- **pass**: the number of consecutive players having passed their turn if the letter bag is empty. If this number reaches the number of total players, the game ends.
- **empty_bag**: equals true (1) if the letter bag is empty.

2.2.3 Letter_bag

The letter_bag table stores each tile in the letter bag. Each entry is defined by 3 attributes:

- **letter**: the letter placed at these coordinate. It is a single character from a to z (for a "classic" letter bag).
CHAPTER 2. STRUCTURE OF THE SERVER

- **letter** (primary key): the tile’s letter defined by a character from a to z (for a "classic" letter bag) plus the number ”0” which stands for a blank tile.

- **frequency**: the remaining number of the corresponding tile in the letter bag.

- **value**: the value of the tile.

### 2.2.4 Players

Each player of the game is defined by the following attributes:

![Figure 2.2: The players table](image)

- **id** (primary key): the id of the player.

- **next**: the id of the next player to play.

- **player**: the player’s name.

- **password**: the hashed player’s password.

- **letter**: the 7 letters on the player’s rack.

- **score**: the player’s score.

- **playing**: equals True (1) for the player whose it is the turn to play.

### 2.3 The classes

The server file uses 8 classes in order to represent a Scrabble game. All of these classes have methods which can be used for a basic “game” using some lines of tests without going through the client side. Considering that I decided later in my project to manage more parts of the game using PHP than I planned at first, some of these methods are not actually used in the final version of the project, but still remain for doing tests.
2.3.1 Letter

The Letter class only describe a tile (or letter) by 3 attributes:

- **char**: the character of the tile. Take the value “0” for a blank tile.
- **value**: the value of the tile.
- **frequency**: the frequency of the tile in the letter bag.

2.3.2 LetterBag

The LetterBag class is used to describe a letter bag by its type (the language used to create it), a lbag attribute consisting of a list of letters (from the Letter class) which is the actual letter bag, and letters who only stores one copy of each letters with their value and current frequency in the letter bag. This attribute is used mainly to retrieve a tile (from the Letter class) from a character, or to construct the lbag from a set of letters.

The letter bag can be created from a pre-existing type (English, French or German), from a list of letters corresponding to the lbag or from a set of letters corresponding to the letters attribute.

In the case of a letter bag derived from a corpora, an external method firstly creates
a set of letters and then call the constructor of the LetterBag class using this set as an argument.

2.3.3 Rack

The Rack class is only composed of a rack attribute which is a list of letters. The Rack class has 3 methods: one to add a letter, one to remove a letter, and one to complete the rack with some random letters from a given letter bag.

2.3.4 Square

The Square class defines a square of the board. It has X and Y attributes (from 0 to 14) corresponding to the coordinates of the square on the board, a letter bonus multiplier and a word bonus multiplier (1, 2 or 3), and can be filled with one letter from the Letter class, or be empty.

This class only have two methods: add_letter and remove_letter, who fill the square with a letter or remove it.

2.3.5 Player

A player is represented by a name, and own a score attribute and a rack from the Rack class. A fourth boolean attribute bot has also been implemented to describe the player as an AI, but is not actually used.

The only method on this class can add points to the score of the player.

2.3.6 Board

The Board class describes a classic Scrabble board, giving 4 attributes:

- A players list who turned out to be not actually used.
- A dictionary name and a letter bag, used by the GhostBoard class through the board.
• A matrix of squares.

The *matrix* attribute is basically a list of list of squares from the **Square** class. Each square is in the *matrix*[x][y] where x and y are the x and y attributes of the square.

The matrix is created via the only board method *create_matrix* who takes for argument a string consisting of 15 lines of 15 integers from 0 to 4, as following:

```
40100040001004
03000200020030
00300101000300
10030010003001
00030000030000
02000200020002
00100010100010
40100030001004
00100010100010
02000200020002
00030000030000
10030001003001
00300101000300
03000200020003
03000200020002
40010004000100
```

0 is a normal square, 1 is letter x2, 2 is letter x3, 3 is word x2, 4 is word x3.

This string is implemented by default on the constructor and match an actual Scrabble board.

### 2.3.7 Game

The **Game** class gathers all the information of the game, including several classes. Its attributes are:

• A letter bag and a dictionary, used to construct a Ghost Board.

• A players list.

• A playing player index used to know which player in the list is currently playing.

• A board from the **Board** class.

• A ghost board from the **GhostBoard** class, re-constructed for each player’s turn.

• The two boolean attributes *waiting_for_players* and *launched*. Depending on their value, only 3 combinations of the couple (*waiting_for_players*, *launched*) are possi-
ble:
- (False, False) ⇒ No existing game.
- (True, False) ⇒ Game created but still waiting for players.
- (False, True) ⇒ Game launched.

The (True, True) combination is not possible as a game cannot be launched if it is still waiting for players. However, it could have been used for describing a finished game, but there was no need for it.

The Game class is also composed of some methods to:

- Add or remove a player.
- Launch a game (creates the Board).
- Begin a player’s turn (creating its Ghost Board) and ending it if the move is valid.
- Switch the turn to the next player.
- Making the players playing one after the other, and sending a signal (returning True) when the letter bag is empty.

Some of these methods are not actually used in the online playing version of the project, but are still functional for doing tests or running a ”console-playing” game.

2.3.8 GhostBoard

The ghost board is used to allow a player to put letters on it, to check if the move is valid and, if so, to actualize the board, the player rack, and to return how many points the player scored.

Rather than directly putting the letters on the board, then check if these letters form valid words, and remove them if not, I choose to use what I called a ghost board as a ”buffer board”. After all verifications are done about the player’s move, the ghost board can either be reinitialized if the move is not correct, or it can ”merge” with the board otherwise.

The other advantage of a ghost board is that there is no need to differentiate the letters already put on the board and the letters who have just been moved on it by the playing
player. Thus, the board always contains valid words, and the ghost board handle by itself all the verifications during a move attempt.

A ghost board has a dictionary and the game board as attributes to proceed to the verifications. It also contains a dictionary of the placed letters with their coordinates as a key. All other attributes are linked to the methods of the GhostBoard class depicted in the Playing a word chapter, as the ghost board only use to handle this part of the game.
Chapter 3

Structure of the client

3.1 The chosen programming language: PHP/JavaScript

For designing the client side, accessible through a simple web browser, I was not very comfortable with Python. I therefore decided to use PHP and Javascript as I have a good experience with them. The relation between the PHP files and the Python server is depicted in the chapter Relation between the client & the server.

3.2 Actions on a game

There are 4 main actions on a game, which I chose to separate in 4 different pages: scrabble_create.php, scrabble_join.php, scrabble_launch.php and scrabble_game.php. The scrabble_lobby.php works as a simple lobby allowing to go on these pages depending on the state of the game.

3.2.1 Creating a game

When no game is in progress, this page allows to create a game (see Creating a game in the Game overview chapter. Once the game is created, the following tables in the SQL database are updated:

- **Game**: a new game is created with the state at 0, and player_1 with the name of the
player who created the game. All others fields are filled with the given information and last_move, pass and empty_bag are initialized with the value 0.

- **Letter_bag**: the letter bag is filled with the letters corresponding to the chosen letter bag. The letter bag creation is done by the python server through an exchange via CGI (see the next chapter).

- **Players**: the first player is added with a score at 0, no letters in his rack, and a playing value at 1 considering that the player who creates the game always plays first.

### 3.2.2 Joining a game

To join a game, a player only needs to give a player’s name and a password. He is then inserted in the `players` table, and its name is added to the `game` table. Once the maximum players number is reached, no more players can join and the game state switch to “waiting for launch”.

### 3.2.3 Launching a game

Launching a game only requires that all players have joined. The game creator has to enter his password in order to initiate the game. The players’ racks only fill when the corresponding player connects to the game page, and the countdown begin when the first player to play (the game creator) logs in.

### 3.2.4 Playing a game

The most important page, `scrabble_game.php`, is the one depicted in The game screen section in the Game overview chapter. From this page, a player can do everything relating to the game: playing a move, seeing whose turn it is to play, consulting the scores, taking tiles in the letter bag to complete his rack etc...
Chapter 4

Relation between the client & the server

4.1 Use of CGI

To make the link between the client side (using PHP/JavaScript) and the server side (using Python), I used the Python’s CGI (Common Gateway Interface) as it is a simple way to communicate with forms between the two sides of the program. It may be a little bit “limited” for more complex operations, but it just fitted my needs in this case.

4.2 Sending data

To send data to the server, I used the Javascript object XHR (XMLHttpRequest) as follows:

```javascript
var xhr = new XMLHttpRequest();
xhr.open('POST', 'game.py');
var form = document.global_form;
var form_data = new FormData(form);
xhr.send(form_data);
```

The form variable correspond to the page’s form which contains all of the data needed by the server.

In the Python file Game.py, the form data are simply recovered as follows:

```python
import cgi
form=cgi.FieldStorage()
```
4.3 Receiving data

When the server has finished computing data, the result is returned as an html text. Each information needed is printed on the Game.py file. On the client side, the JavaScript code used to send the data to the server also receive and use the returned information:

```javascript
xhr.onreadystatechange = function()
{
    if (xhr.readyState == 4 && xhr.status == 200){
        var response = xhr.responseText;
        //...
        //Using the response variable
        //...
    }
}
```

With the data received from the Python server, the JavaScript code can modify the form and send it in order to make a PHP “post-computing”, consisting of actualizing the data on the database and displaying whatever it has to be displayed on the player’s screen.
Chapter 5

Deriving a letter bag from a corpora

5.1 Goal

Deriving a letter bag from a text, in terms of letter frequencies and values, can give a more accurate letter bag regarding the language of the text. Having the same letter frequencies than in a standard text, and the letter values depending on this frequency, a Scrabble game using a dictionary of the same language can only be more balanced. It is also interesting to compare the differences between a letter bag derived from a corpora given a specific language, and the actual letter bag of this language.

5.2 The way of doing it

For a given text, which can be either taken from an HTML page, of from a raw piece of text, the ”transformation” into a letter bag is done in 5 steps

- Firstly, we get rid of all HTML tags in order to have ”pure” text and don’t take in account the parasite letters in the tags.
- Then, we replace all accented letters into simple letters from A to Z. The German letter ”ß” is replace by a ”ss”. The inconvenient of it is that some languages have accented letters in their letter bag (for example Spanish or Irish) and cannot produce a good letter bag from this method. A solution could be to check the language of the text and allow or not some accented letters to be replaced.
• We store all the letters with their respective occurrence.

• Each letter gets the same frequency in the letter bag than in the text, in order to get around 100 letters plus 2 blank tiles. As frequencies on a one hundred elements lot are rarely integers, rounding can lead to have a little more or less than 100 letters. However, it does not really matter as each letter bag is not composed of exactly the same number of tiles depending on their language (from 100 to 120, including blank tiles).

• Each letter gets a value depending on its frequency from 1 (frequency of 5% and more) to 10 (around 0%).

The exact formula is \( \text{max}(1, \text{round}(10 - \text{frequency} \times 180)) \), so the actual minimum letter frequency to get a value of 1 is 4.73% (in the English language, the letters E, T, A, O, I, N, S, H, R) and the maximum letter frequency to get a value of 10 points is 0.275% (in the English language, the letters Z, Q, X, J).

The choice of a value of 1 for all letters with a frequency of more than 5% is due to the fact that in most Scrabble games, the division between the letters with a value of 1 and the letters with a value of 2 is situated between the letters having an occurrence of 5 or 4 in a 100 tiles letter bag.

### 5.3 Results

This is a comparison between the English Scrabble letter bag and the letter bag derived from the first chapter of the Lord of the Ring (10,000 characters).


Note: The values of the letters are not taken in account here as their designation depended on a personal choice.

We can see that the frequencies are very close to the official one, except for the letters H and T. This may be due to the fact that the word “the” (but maybe not only this word) is very present in an actual English text, but considering that you cannot play Scrabble only with one word, it has been decided that their importance in the Scrabble letter bag had to be reduced. In fact, it is more probable than the frequencies of the letters are based on a dictionary and not on a raw text. Therefore, I decided to use the
Figure 5.1: English Scrabble & LOTR derived letter bags

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
<th>Value</th>
<th>Letter</th>
<th>Frequency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
<td>a</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>3</td>
<td>b</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>3</td>
<td>c</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>2</td>
<td>d</td>
<td>+1</td>
<td>5</td>
</tr>
<tr>
<td>e</td>
<td>12</td>
<td>1</td>
<td>e</td>
<td>-1</td>
<td>11</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>4</td>
<td>f</td>
<td>+1</td>
<td>3</td>
</tr>
<tr>
<td>g</td>
<td>3</td>
<td>2</td>
<td>g</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>h</td>
<td>2</td>
<td>4</td>
<td>h</td>
<td>+3</td>
<td>5</td>
</tr>
<tr>
<td>i</td>
<td>9</td>
<td>1</td>
<td>i</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>j</td>
<td>1</td>
<td>8</td>
<td>j</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>5</td>
<td>k</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>l</td>
<td>4</td>
<td>1</td>
<td>l</td>
<td>+1</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>2</td>
<td>3</td>
<td>m</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>n</td>
<td>6</td>
<td>1</td>
<td>n</td>
<td>+1</td>
<td>7</td>
</tr>
<tr>
<td>o</td>
<td>8</td>
<td>1</td>
<td>o</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>p</td>
<td>2</td>
<td>3</td>
<td>p</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>q</td>
<td>1</td>
<td>10</td>
<td>q</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>r</td>
<td>6</td>
<td>1</td>
<td>r</td>
<td>-1</td>
<td>5</td>
</tr>
<tr>
<td>s</td>
<td>4</td>
<td>1</td>
<td>s</td>
<td>+1</td>
<td>5</td>
</tr>
<tr>
<td>t</td>
<td>6</td>
<td>1</td>
<td>t</td>
<td>+3</td>
<td>9</td>
</tr>
<tr>
<td>u</td>
<td>4</td>
<td>1</td>
<td>u</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>4</td>
<td>v</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
<td>4</td>
<td>w</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>8</td>
<td>x</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>4</td>
<td>y</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>z</td>
<td>1</td>
<td>10</td>
<td>z</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

English dictionary I found (58,000 words) to create another letter bag and check if this last hypothesis is confirmed:

We can see now that the total gap is reduced from 18 to 13, without taking in account the big difference of frequency of the letter "S", due to the fact that many words of the
used dictionary, as it a Scrabble dictionary, had their plural version.

Figure 5.2: English Scrabble & English dictionary derived letter bags

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
<th>Value</th>
<th>Letter</th>
<th>Frequency</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>9</td>
<td>1</td>
<td>a</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>3</td>
<td>b</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>3</td>
<td>c</td>
<td>-2</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>2</td>
<td>d</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>12</td>
<td>1</td>
<td>e</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>4</td>
<td>f</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>g</td>
<td>3</td>
<td>2</td>
<td>g</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>h</td>
<td>2</td>
<td>4</td>
<td>h</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>9</td>
<td>1</td>
<td>i</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>j</td>
<td>1</td>
<td>8</td>
<td>j</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>k</td>
<td>1</td>
<td>5</td>
<td>k</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>4</td>
<td>1</td>
<td>l</td>
<td>+1</td>
<td>5</td>
</tr>
<tr>
<td>m</td>
<td>2</td>
<td>3</td>
<td>m</td>
<td>+1</td>
<td>3</td>
</tr>
<tr>
<td>n</td>
<td>6</td>
<td>1</td>
<td>n</td>
<td>+1</td>
<td>7</td>
</tr>
<tr>
<td>o</td>
<td>8</td>
<td>1</td>
<td>o</td>
<td>-2</td>
<td>6</td>
</tr>
<tr>
<td>p</td>
<td>2</td>
<td>3</td>
<td>p</td>
<td>+1</td>
<td>3</td>
</tr>
<tr>
<td>q</td>
<td>1</td>
<td>10</td>
<td>q</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>6</td>
<td>1</td>
<td>r</td>
<td>+1</td>
<td>7</td>
</tr>
<tr>
<td>s</td>
<td>4</td>
<td>1</td>
<td>s</td>
<td>+5</td>
<td>9</td>
</tr>
<tr>
<td>t</td>
<td>6</td>
<td>1</td>
<td>t</td>
<td>+1</td>
<td>7</td>
</tr>
<tr>
<td>u</td>
<td>4</td>
<td>1</td>
<td>u</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>4</td>
<td>v</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
<td>4</td>
<td>w</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>8</td>
<td>x</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>4</td>
<td>y</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>1</td>
<td>10</td>
<td>z</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Chapter 6

Playing a word

When a word is played, the placed letters are saved in the ghost board and the validity of the move is checked by the Python server in the 3 following main steps. All of these steps takes place in the GhostBoard class through the many methods used to analyse a move.

6.1 What is the direction of the word and are the letters correctly put?

Firstly, when the letters are placed on the ghost board, we check the direction of the played word. Knowing the direction of the played word is important to know how to read the entire formed word. To do so, if the word is composed of at least 2 letters, we find the “steady coordinates” of the first two letters: if X is steady, the word is horizontal, if Y is steady, the word is vertical.

We check then if the other letters match this direction. If not, then the move is not valid.

If there is only one letter placed, we consider the direction as unknown and we will check the formed words later, including horizontal and vertical words. If no letters have been placed, then the move is obviously not valid.

We also check if there is already a letter at the same place on the board for each placed letter.
6.2 What are the formed words and how much points do they cost?

If there is more than one letter placed, depending on the direction of the played word, we browse each letter placed by the player from left to right, and then from right to left, and concatenate the first with the reverse of the second in order to have the main formed
word. We browse the letters in both way until no letters is encountered, such that we have read the entire word. If some letters on the ghost board haven’t been used, this mean that at least one placed letter is separated from the rest of the word, and therefore the move is not valid. We also check if the created word crosses another or go through the middle of the board (7,7).

Then, for each letter placed, we search in the perpendicular direction all the new words formed with the same method.

For each letter browsed in a word, we calculate the points it is worth considering the bonuses. We have in the end a set of words and the points it gets. But there is one more verification to do before validate...

### 6.3 Are the words correct ?

That last part is the simpler : for each new word formed, we check if it exists in the given dictionary. If at least one word is not recognize, the move is not valid.
When all of these steps are successful, the ghost board merges with the board, the player’s score is updated, the letters used are taken off the player’s rack and new letters from the letter bag are randomly added to it.
Some features could have been added and were the subject of a thinking, but were not implemented due to some circumstances.

6.4 The AI

Including AI players was one the my main objectives after finishing the project. I though of several ways to implement it but it was a more difficult issue than I though first. Making a simple AI which plays some words is complicated enough, and although I did not coded it, I have explored some ways to do so:

- One way would be to browse each word of the dictionary (from a random point to don’t always get the first words), and see if the word can be written with some letters on the rack plus one from the board.

- Another way would be to generate all the possible words with the letters on the rack plus one “joker” going from a to z, and see if the word can be put in one place on the board with the joker letter taking place in a matching letter already in the board.

- Finally, the better way I would do it is to take a letter on the board at random, see if there is some empty squares on a particular direction, count it and check all the word in the dictionary that could fit in. If a word can be written with the letters on the tile, do it. Otherwise, take another letter on the board, until one correct word is found or a counter of tests reach a certain value.

For example, is the following picture, if the ‘A’ is taken as a base, the following templates of words can be taken in the dictionary (with - as any letter):
2 letters: -A ; A-
3 letters: --A ; -A-
4 letters: --A- ; A- -B
5 letters: -A- -B ; A- -B-
6 letters: --A- -B ; -A- -B- ; A- -B- -
7 letters: --A- -B- ; -A- -B- - ; A- -B- - -
8 letters: --A- -B- - ; -A- -B- - - ; A- -B- - - -
9 letters: --A- -B- - - ; -A- -B- - - - ; A- -B- - - - -

Figure 6.3: Example of AI playing
CHAPTER 6. PLAYING A WORD

However, these AI would not take in account the score each move is worth (but could try some moves and take the better to increase difficulty), and will not use advanced strategy as blocking a possible move for the opponents.

6.5 Including teamplay & Multiple games handling

Allowing a game with teams was a discussed development. It could have lead to another way to play, taking in account the letters of your team mate to place words which would help him play a word worth many points. That would have required a team partition (2 vs 2 if no more than 4 players), a single score for both team, a visible team mate rack and a chat box for each team. Multiple games handling was also discussed, but as this project did not need a large set of players (only one game at a time was sufficient to test the functionalities of the project), I did not focused on this particular part.
Conclusion

Although it can be improved, as every piece of software, this online Scrabble game is fully functional and implements a way to create its own letter bag from a corpora. This allows us to compare it with the classic letter bags and better understand how the letters frequencies and values have been decided.

Using Python through such a project was a really good way to become confident with it, and doing researches in order to allow the PHP client and the Python server to communicate with XHR led me to use it in another personal projects.

For these reasons, I really enjoyed taking part of this project, and a future development is expected, especially in the field of the AI.
Sources

English dictionary (58.000 words):
http://www.mieliestronk.com/wordlist.html

French dictionary (364.000 words):
http://perso.infonie.fr/chr.amet/ods3.zip

German dictionary (190.000 words):
http://www.netzmafia.de/software/wordlists/deutsch.txt

Scrabble letters distributions:
http://en.wikipedia.org/wiki/Scrabble_letter_distributions#German

Rules of Scrabble:
http://www.fisf.net/index.php?Itemid=52&id=29&option=com_content&task=view