The Scrabble Game by “La Squad”

Enzo Merotto	61540460051	Enzo
Sabri Durand	61540460050	Sabsab
Rattapon Jindalak	61070503441	Kong
Léo Tavernier	61540460052	Léo

Summary statement

In CPE 111, we have one semester to develop a program in C language. For this project we need to create a Scrabble Game in a group of four. In our program we need to implement a board, a list of letters for each player, the moves that the player can do, a random way to pick letters, a score and a save. We will use two different data structures.

Requirements list

1. The system can let the user create/save/delete a game and load an existing game if there's a saved one.
2. The system let the player choose the number of players that will play (two to four).
3. The system let the user define a name for every player in a newly create game.
4. The system randomly selects letters and delivers to player hand at new game or upon request.
5. The system must always show the current state of a board.
6. The system keeps track of each letter (the letter on the board, on player's hand and in a bag).
7. The system determines the score value of each letter and keeps score in memory.
8. The calculating is based on level of difficulty of a letter and multiplier square the letter is placed on.
9. The system keeps track of each letter being used and not used. So, there is a limitation of each letter in letter bag.
10. The system uses a dictionary to check each new word proposed by the player. And check player's hand that every letter in that word exist.
11. The system will try to prevent the player to seeing each other hand, there will be cutscene such as "Player 1 ready to play? If ready, then type YES ".
12. The system determines which player wins from total calculated score when the tiles in the bag is all gone, or nobody can make a word.

Scrubble Game System

Naming each player

Choose number of players to play(2-4)

<<Include>>

Start a new game

<<Include>>

Load saved game

Save current game

Pass

<<Include>>

Lose the game if 2 pass in a row

<<Extend>>

Take action

<<Extend>>

Play a word

Require

<<Extend>>

Use dictionary to check if a word is valid

Change letters

Require

Require

View letters in your hand

Require

Randomly select letters from the bag

Require

Show who win if all tiles are gone from the bag

<<Include>>

Keep track of letters on player hand and bag

Show the current state of a board

Determine and show a score of each letter

<<Extend>>

Multiply score if it's on special tiles

Player

System

Use case narratives

Use case name: ***Choose number of players to play (2-4)***

Actor: Player

Goal: Determine number of players in a game

Main success narrative:

1. System asks you to enter a player number between 2-4.

Alternative narrative 1:

1. System finds player's enter number is not between 2-4.
2. System displays an error message and asks again.

Use case name: ***Naming each player***

Actor: Player

Goal: Give a name to all players in the game

Main success narrative:

1. System asks you to enter a player name.
2. System asks the name again if the player name is not last player.

Alternative narrative 1:

2. System finds player's enter name is duplicate with another player.
3. System displays an error message and asks again.

Use case name: ***Load saved game***

Actor: Player

Goal: Continue the play of saved game

Main success narrative:

1. System lists the date and number of players in a list.
2. System asks you which game save to load.
3. User types number of saved games in a list to load.
4. System load up a saved game.
5. System displays load success message.

Alternative narrative 1:

4. System finds player's enter number that not exist in the list.
5. System displays an error message and asks again.

Alternative narrative 2:

4. System could not load a game save (permission error, etc.).
5. System displays an error message and asks again.

Use case name: ***Save current game***

Actor: Any player

Goal: Save current game into a file

Main success narrative:

1. System asks you to enter a name of the save.
2. System save the current state of the board, current state of each player hand, current state of player turns into a game save file.
3. System display save success message.

Alternative narrative 1:

2. System couldn't save a game save (permission error, etc.).
3. System displays an error message and asks again.

Use case name: ***Make a move***

Actor: Player

Goal: The player can choose between three actions

Main success narrative:

1. System asks the player which action chooses.
2. Player enters a number that correspond at an action (play a word, change letters, pass, view his letters).
3. System runs the action that the player has chosen.

Alternative narrative 1:

2. Player enters a wrong character that not appears in the proposed option.
3. System display an error and ask again.

Use case name: ***Change letters***

Actor: Player

Goal: Swap the player's letters with new ones

Main success narrative:

1. System asks the player to enter letter(s).
2. Player types the letters that he wants to change.
3. System picks randomly the corresponding number of new letters.
4. System puts the old letters in the bag.

Alternative narrative 1:

3. Player doesn't have the letters that was typed.
4. System displays an error message and asks again.

Alternative narrative 2:

3. There are not enough letters in the bag.
4. System displays an error message and asks if the player wants fewer letters.

Use case name: ***View letters***

Actor: Player

Goal: System shows the letters to the player

Main success narrative:

1. System keeps in memory the letters of each player.
2. Player asks the system to show his letters.
3. System shows to the player his letters.
4. System asks the player to leave.

Use case name: ***Pass***

Actor: Player

Goal: The player decides to pass his turn

Main success narrative:

1. Player enters the number that correspond at the action “pass”.
2. System passes the turn of the player and goes to the next player.

Use case name: ***Play a word***

Actor: Player

Goal: The player decides to play a word on the board

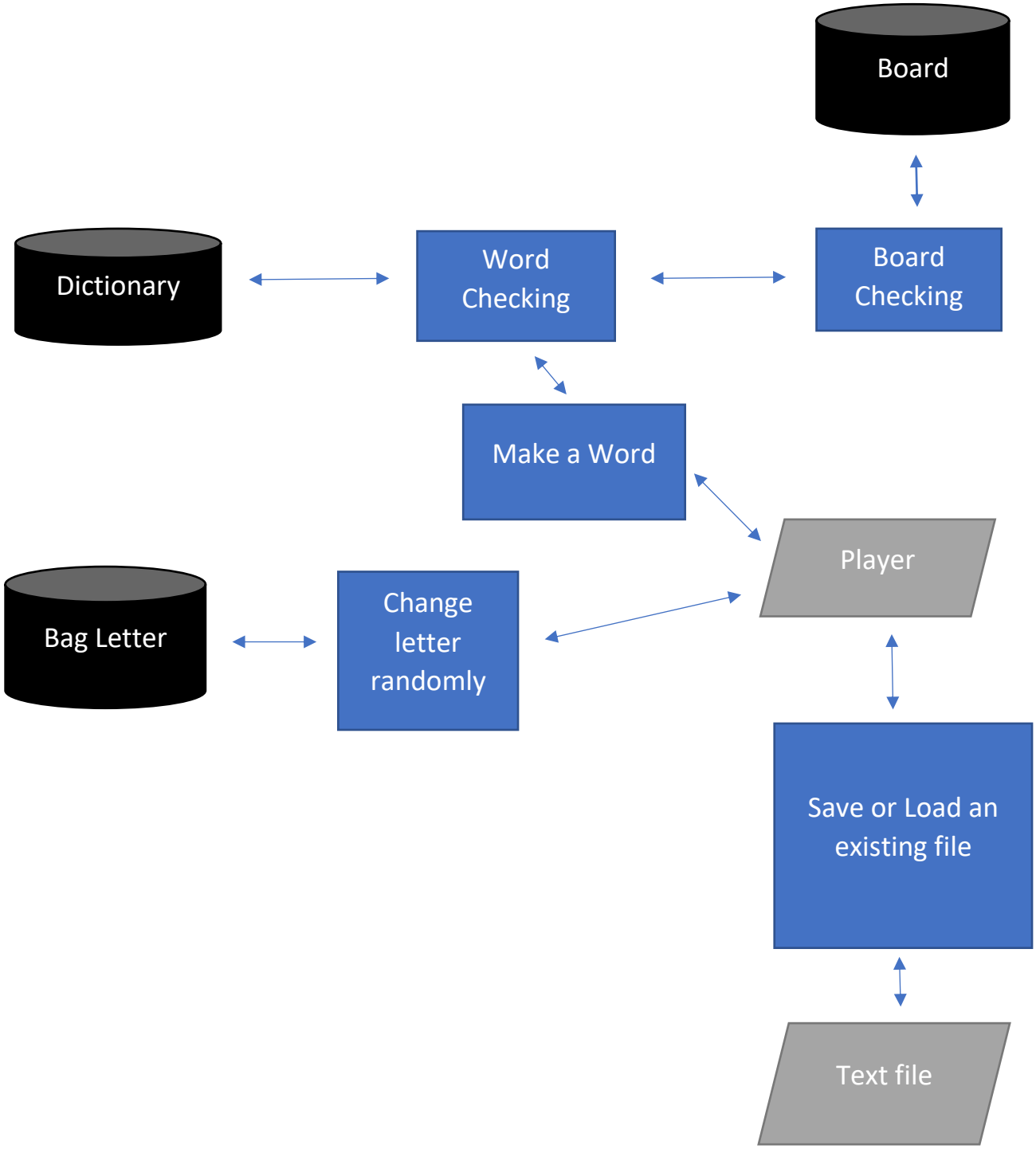
Main success narrative:

1. System asks the player which word the player wants to play.
2. Player enters the word.
3. System files the board with the letters of the word.
4. System calculates the score of the word.

Alternative narrative:

3. Player enter a word that does not exist in the dictionary, or the word cannot be placed on the board or the player does not have the letters.
4. System display an error and ask again.

Architecture Diagram



Discussion and Diagram of major data structures

In the architecture diagram we saw that we will use 3 main data structures in our project.

1) Text File

This text file will be used to store all the information about the current game, to save the game. Thanks to that we will be able to load an existing game that has been saved in the text file.

We will use a text file because we saw in class how to use them and because they are easy to use to store information that you will need to access after quitting the game.

2) Matrix

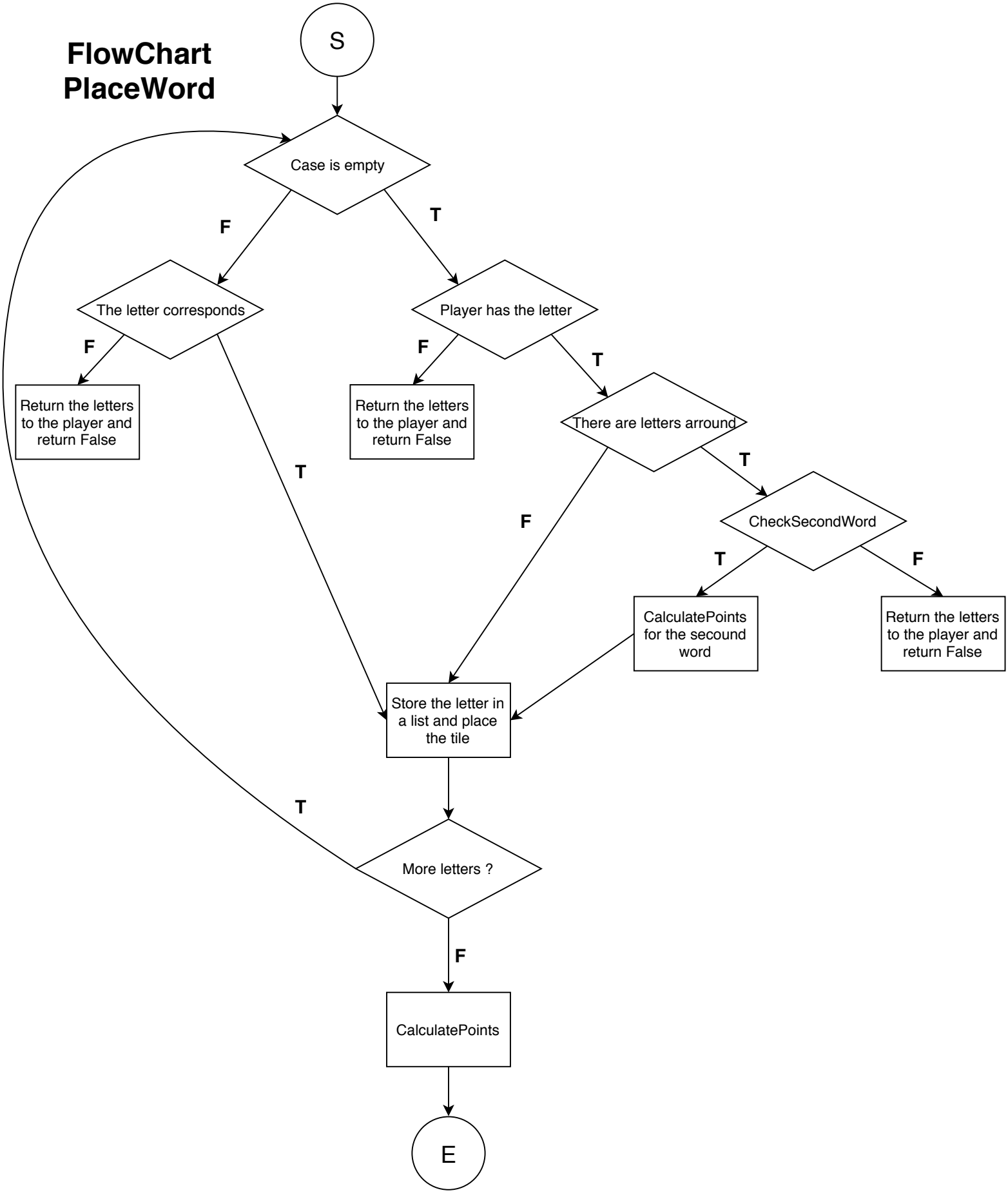
The matrix will be used to store the board of our game.

We will use a matrix because using a matrix is the solution that appear to us as the most explicit solution. Also, we need to be able to access to any cell at any time without traversing all the matrix so using a matrix will be a good solution.

3) Tree (AVL)

We will use an AVL to store our dictionary, we want to use an AVL because we will traverse our tree nearly each turn during the game. So, our structure really needs to be effective to don't slowdown our program. The AVL is the structure that is the most effective to store data that you want to access often. So, we think that using AVL is a good choice.

FlowChart PlaceWord



Description :

This function allows the player to know if the word that he wants to play can be placed. It is a loop that check each letter of the word. If a letter can be placed, the tile is pre-placed on the board and the function go to the next letter(if there is one).If it is not the case, every letters that was placed before, are restored to the player.
The function **CheckSecondWord** is used to see if the others words created are correct.

Pseudocode:

```
DictionaryToAVL(dictionary, left, right, node)
    If right != left
        mid = left + (right - left) / 2
        node.data = dictionary[mid]
        DictionaryToAVL(dictionary, left, mid, node.left)
        DictionaryToAVL(dictionary, mid+1, right, node.right)
    return node
```

Description**:**

The function **DictionaryToAVL** transforms an array of sorted words into an AVL. We use a binary search to create this AVL.

```
CheckWord(word, dictionary)
    if dictionary != null
        if word < dictionary.data
            return CheckWord(word, dictionary.left)
        else if word > dictionary.data
            return CheckWord(word, dictionary.right)
        else
            return True
    return False
```

Description:

The function **CheckWord** take a word as a parameter and the root of the tree that is our dictionary. This function traverses the dictionary tree to check if the word exists in the dictionary and return true if the word exists. If the word doesn't exist in the dictionary or is null, the function will return false.

Overview Diagram