|  |
| --- |
| Institute of Technology Carlow, Kilkenny Road, Carlow, Co. Carlow |
| Scrabble Engine |
| Functional Research Document |

|  |
| --- |
| Keith Byrne & Liam Strevens  11 November 2015 |

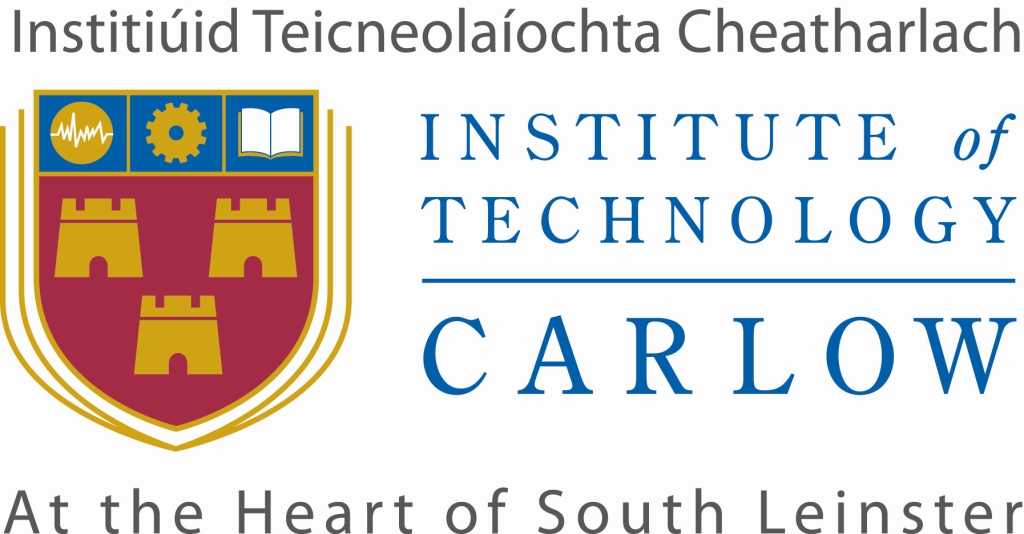


Table of Contents

[1 Project Development (Research Driven Conclusions) 4](#_Toc435548723)

[1.1 Primary Methods (Discovery) 4](#_Toc435548724)

[1.2 Secondary Methods 4](#_Toc435548725)

[2 Project Overview 5](#_Toc435548726)

[2.1 Introduction 5](#_Toc435548727)

[2.2 Conceptual System Description 5](#_Toc435548728)

[2.3 Data Layer 6](#_Toc435548729)

[2.4 Core Engine (Logic) Layer 6](#_Toc435548730)

[2.5 Graphical User Interface Layer 6](#_Toc435548731)

[3 Functional Requirements 7](#_Toc435548732)

[3.1 Use Case Diagram 7](#_Toc435548733)

[3.2 Brief Use Cases 8](#_Toc435548734)

[3.2.1 Set Difficulty 8](#_Toc435548735)

[3.2.2 View Scores 8](#_Toc435548736)

[3.2.3 Login 8](#_Toc435548737)

[3.2.4 Logout 9](#_Toc435548738)

[3.2.5 Sign Up 9](#_Toc435548739)

[3.2.6 Submit Word 9](#_Toc435548740)

[3.2.7 Place Tile 9](#_Toc435548741)

[3.2.8 Remove Tile 10](#_Toc435548742)

[3.2.9 Start Game 10](#_Toc435548743)

[4 Detailed Use Cases 11](#_Toc435548744)

[4.1 Login 11](#_Toc435548745)

[4.2 Logout 11](#_Toc435548746)

[4.3 View Scores 11](#_Toc435548747)

[4.4 Set Difficulty 12](#_Toc435548748)

[4.5 Sign Up 12](#_Toc435548749)

[4.6 Submit Word 12](#_Toc435548750)

[4.7 Place Tile 13](#_Toc435548751)

[4.8 Delete Tile 13](#_Toc435548752)

[5 Non Functional Requirements 14](#_Toc435548753)

[5.1 Functionality 14](#_Toc435548754)

[5.2 Usability 15](#_Toc435548755)

[5.3 Reliability 15](#_Toc435548756)

[5.4 Performance 15](#_Toc435548757)

[5.5 Supportability 15](#_Toc435548758)

[6 Data Structures 16](#_Toc435548759)

[6.1 Linear Array 16](#_Toc435548760)

[6.2 Trie 17](#_Toc435548761)

[6.3 B-Tree 18](#_Toc435548762)

[6.4 Performance Analysis (Secondary Research) 18](#_Toc435548763)

[7 Core Algorithms 19](#_Toc435548764)

[8 Functional Specification Data Requirements 20](#_Toc435548765)

[8.1 Dimensions 20](#_Toc435548766)

[8.2 Metrics 20](#_Toc435548767)

[8.2.1 D001 20](#_Toc435548768)

[8.2.2 D002 20](#_Toc435548769)

[8.2.3 D003 20](#_Toc435548770)

[8.3 FS Dimension-Metric Matrix 21](#_Toc435548771)

[9 Conclusion 22](#_Toc435548772)

[10 References 23](#_Toc435548773)

[11 Table of Figures 24](#_Toc435548774)

Foreword

There is an ancient saying from Greek philosopher, Aristotle, “We are what we repeatedly do. Excellent, then, is not an act, but a habit”. The same principle can be easily applied to how machines learn from repeated simulation efforts to further learn and improve a game strategy or problem solving ability using self-learning algorithms, designed to evolve through trial and error. When a user plays Scrabble, however, we don’t immediately consider the application of an artificial intelligence engine or systematic calculation of words and scoring possibilities.

When a group of people sit around a table to enjoy a game of Scrabble, they are not aware of the very algorithmic nature of how they are playing the game. Humans, at least in the context of a word game like Scrabble, are not efficient organisms for this application. Throw emotional judgement into the equation and the results can be even less satisfactory. An engine purpose built for the application however.

# Project Development (Research Driven Conclusions)

This document will focus on in-project decision making conclusions directed towards functional advancement which will be mainly based off primary research regarding algorithm efficiency, internal logic vs stored procedural logic, learning techniques and artificial intelligence integration. This will be supported by secondary research and tertiary research if necessary or available. Due to the research driven development nature of the project, certain conclusions cannot be made until areas of the project are explored (based off of functional requirements contained herein), tried and tested and compared against previous results to conclude the better approach. This will be detailed as functional investigation later in this document.

## Primary Methods (Discovery)

This will involve the practical testing of different algorithms, for example, searching and sorting algorithms to operate on character and string data retrieved from the data layer.

## Secondary Methods

Certain portions of research can be minimized from previous conclusions made by other projects.

# Project Overview

## Introduction

The proposed Scrabble engine will be a standalone application which can be downloaded and executed on any machine capable of running .NET 4.0 executables. The primary functional goals of the system will be to allow a player to compete against a simulated Scrabble player to determine overall skill. The system should incorporate a level of artificial intelligence to achieve parity with players if they decide, or alternatively a set difficulty level to compete against, for example, Beginner, Intermediate, Hard, Expert, etc. At a later stage, advanced intelligent learning capabilities could be incorporated to constantly improve the system's end game and scoring capacity by learning frequent patterns and how best to counter them based on previous attempts.

## Conceptual System Description

To achieve learning capabilities and to simply store user information, the system will need to admit both a front end layer and a back end data layer, whether locally stored or externally via web hosted database will be decided upon benchmark results and requirement reassessment.

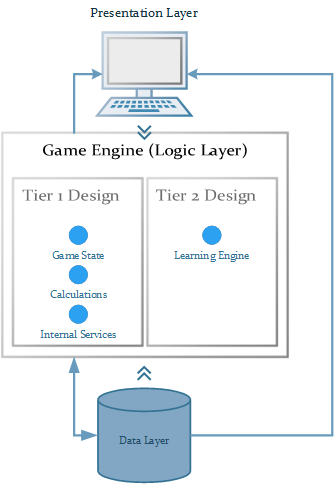


Figure 2.a Conceptual View of the Scrabble Engine System

## Data Layer

Present at this layer, the underlying structured data model will represent the stored user information, a provision for self-learning information, common patterns, all the necessary character pool conditions such as character/tile scores and distribution amounts and efficiency data for reporting purposes. Currently, positive results have been found through functional exploration using a locally hosted SQL Compact Edition .SDF local database file housing words and distributions in terms of read and write times and reliability.

## Core Engine (Logic) Layer

This layer presents the most difficult and experimental functionality of the whole system. All A.I functionality will be present here as will all calculations of scores, tracking, etc. The bulk of research throughout the project is centred on achieving a functional end point of this layer.

## Graphical User Interface Layer

The construction of a responsive interface on which users can compete against the engine will be required. This design will be based off the Hasbro [1] standard design pattern, styles may vary in the project design however.

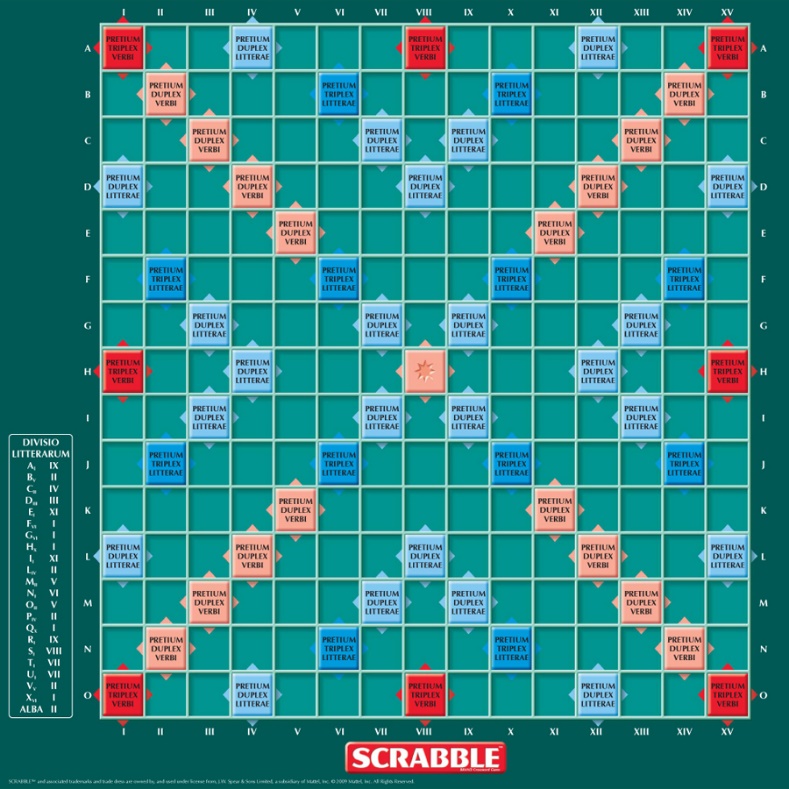


Figure 2.b Hasbro Scrabble Design [1]

# Functional Requirements

## Use Case Diagram

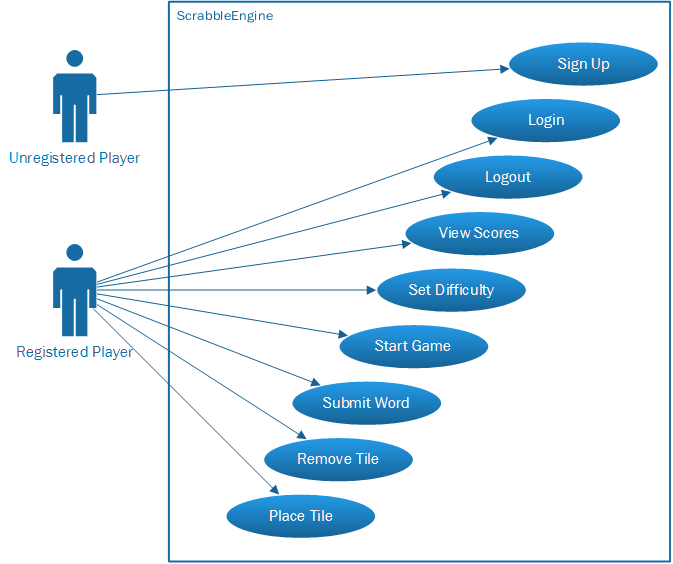


Figure 3.a System Use Case Diagram

## Brief Use Cases

### Set Difficulty

Name: Set Difficulty

Actors: User, System (Core Engine)

Preconditions: A user has been successfully logged into the system.

Description: This Use Case begins when the user requests a change in the session game difficulty. The GUI-layer will transfer the requests to the core engine which will then register the difficulty level with the session.

Post Condition: All moves in the current session will be based off of the select difficulty level. The user will be asked whether they wish to globally set the difficulty level and register this with the data layer.

### View Scores

Name: View Scores

Actors: User, System (Data Layer)

Preconditions: A user has been successfully logged into the system.

Description: This Use Case begins when the user posts a GUI request to view scores from previous games. The system retrieves the data from the data layer and subsequently displays the data in a custom results form.

Post Condition: Non-functional. System waits for user to close dialog

### Login

Name: Login

Actors: User, System (Data Layer)

Preconditions: The system loads the GUI layer and asks user for login credentials

Description: This Use Case begins when the user run the inline client executable. The GUI layer presents the Login form asking for details.

Post Condition: Once the Login button has been pressed, the system checks credentials and displays the initial game board.

### Logout

Name: Logout

Actors: User, System

Preconditions: A user has been successfully logged into the system.

Description: This Use Case begins when the user requests the logout procedure.

Post Condition: User will be asked to close the application or return to the login screen.

### Sign Up

Name: Sign Up

Actors: User, System

Preconditions: User is not logged into the system

Description: This Use Case begins when a user wishes to register a new user account to play the game.

Post Condition: User will be returned to the login screen and asked to enter the details they just entered.

### Submit Word

Name: Submit Word

Actors: User, System

Preconditions: User is in a currently active game session.

Description: This Use Case begins when a user calls to submit a word placed on the game board. The system will check the validity of the word, resulting score with bonus multiplier (if any) and deduce the tiles from the distribution list.

Post Condition: A.I will perform its turn.

### Place Tile

Name: Place Tile

Actors: User, System

Preconditions: User is in a currently active game session.

Description: This Use Case begins when a user selects a tile and attempts to place it in on the game board. The system checks the validity of the placement (that is satisfies the current boundaries of the game board).

Post Condition: System listens for further placements, removals or word submits.

### Remove Tile

Name: Remove Tile

Actors: User, System

Preconditions: User is in a currently active game session and there is at least a single tile on the game board.

Description: This Use Case begins when a user wishes to remove a tile from the board and subsequently have it returned to the current character pool.

Post Condition: System listens for further placements, removals or word submits.

### Start Game

Name: Start Game

Actors: User, System

Preconditions: User has successfully logged in and makes a selection from the main game menu.

Description: This Use Case begins when a user selects the option to begin a new game. The system then loads a random letter ordering and distributes the first round of letters to the game tray.

Post Condition: System prompts user to take the first turn and listens for first placement.

# Detailed Use Cases

## Login

Name: Login

Actors: User, System (Data & GUI-Layer)

Preconditions: Program executes and loads login screen

1. User launches the application
2. System prompts user for login details (username and password)
3. User enters log in details and submits them
4. System requests user’s login details from the external single-sign on directory
5. System validates that user’s entered login details match those from the directory
6. System displays the main menu

Alternative Flow 4a: Incorrect login details

1. System tells user their login has failed
2. Return to Main Flow step 2

Post Condition: User is directed to the main game menu.

## Logout

Name: Logout

Actors: User, System (Data & GUI-Layer)

Preconditions: User is logged in.

1. User activates the logout sequence by pressing the logout button.
2. System removes User information from session memory.
3. Program redirects to the initial login screen.

Post Condition: None

## View Scores

Name: View Scores

Actors: User, System (Data & GUI-Layer)

Preconditions: User is logged in and is not in an active game session.

1. User selects the View Scores selection from the GUI.
2. System retrieves all game history from the database for the user using the unique username.
3. System loads all information from memory onto a display.

Alternative Flow 2a: No data available

1. System informs user that a game must be played before game history can be displayed.

Post Condition: System waits for user to close the locked dialog to continue.

## Set Difficulty

Name: Set Difficulty

Actors: User, System

Preconditions: User is logged in and is in an active game session.

1. User selects the difficulty dialog
2. System present the available difficulties to choose from.
3. User selects their desired difficulty.
4. System registers the difficulty.

Alternative Flow 4a: Difficulty already selected

1. System requests a difficulty level that isn’t the current difficulty level.

Post Condition: System returns to main game screen.

## Sign Up

Name: Sign up

Actors: User, System

Preconditions: System has loaded the login screen.

1. User selects Sign Up on the GUI.
2. System loads the registration form and presents the dialog to the user.
3. User enters form details
4. System checks if unique credentials exist
5. System informs successful submission of user information and stores details in the database.

Alternative Flow 4a: Username exists

1. System informs user that the username has been taken
2. User enters new username
3. System checks validity

Post Condition: System returns to login screen and awaits user login.

## Submit Word

Name: Submit Word

Actors: User, System

Preconditions: System has placed the desired word on the game board, no boundary issues.

1. System loads each character in the block to memory.
2. System checks adjacent words that were extended or bridged using the tiles placed.
3. System checks validity of the words.
4. System calculates the score of the tiles.
5. System add the scores from multiplier bonuses.
6. System posts the score to the GUI and informs player word successfully entered.
7. System randomly refills the player tray with characters from the remaining word pool.

Alternative Flow 2a: Word breaks boundary rules

1. Inform user this placement is not possible
2. Ask for new word

Alternative Flow 3a: Word is not a valid dictionary word

1. Inform user this placement is not possible
2. Request remove tiles until placement is valid

Post Condition: System updates GUI components and prepares the A.I turn.

## Place Tile

Name: Place Tile

Actors: User, System

Preconditions: Player has started a game session.

1. User selects character from tray.
2. User places character on game board.
3. System checks whether tile can be placed at location.

Alternative Flow 3a: Word breaks boundary rules

1. Inform user this placement is not possible
2. Return tile to character tray and publish error prompt.

Post Condition: System waits for next placement

## Delete Tile

Name: Delete Tile

Actors: User, System

Preconditions: Player has started a game session and at least one tile has been placed on the game board.

1. User selects tile and returns it to the tray.
2. System restores the character tile to the tray pool.

Post Condition: System waits for next placement.

# Non Functional Requirements

## Functionality

**Login**

The system will have a secure login functionality for users of the system. All Information will be stored in an SQL database.



Figure 5.a System Log from Alpha Design

**Sign Up**

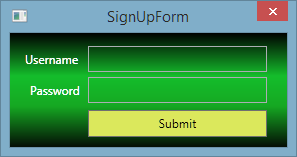


Figure 5.b System Sign Up in Alpha Design

The system will allow new users to create an account in the system. The information of the user will be stored in the SQL database.

**Record scores**

The system will records the scores of the registered user in a SQL database.

## Usability

The Graphical user interface will have a very similar look and feel to the classic Hasbro game of scrabble to make it more familiar to users of the system. The system will use a drag and drop technique to place tiles onto the scrabble board. The system will show the user the current score. The following is a list of requirement that should be completed by a novice user of the system.

* A user of the system will be able to register with the system in under one minute.
* A user of the system will be able to login in 20 seconds.
* A user should be able to place a word in under 30 seconds
* The computer should be able to find the highest scoring word in under 90 seconds

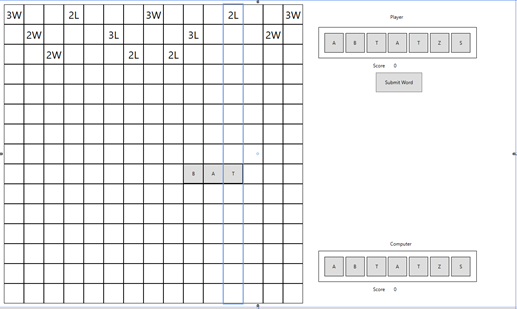


Figure 5.c System Main Game in Alpha Design

## Reliability

For a system like this, the system should be available over 95% of the time. The only issue with the software could possible connection to the database. The maximum time to fix a problem should be no more than three hours. Any longer than this and the software doesn’t become appropriate for a large customer base.

## Performance

Performance is the key working factor for the whole project. Initially, the algorithms will be slow. This is the expected evolution of the project. With each additional character added to the search stack, the number of potential combinations increase exponentially. Due to this, careful micro iterations will be necessary when working with the selected algorithms and the program should be able to compute the single best word placement in less than five seconds. To further combat performance roadblocks the game will force a 9 character maximum selection (8 placed tiles and 1 existing board tile).

## Supportability

As the project expectation is primarily to produce a correctly functioning prototype, supportability will come in the form of A.I change management and feature branching to improve the algorithm and allow for a design that eliminates limits purposely set by the project owners to satisfy time constraints.

# Data Structures

For this project we are going to test the efficiency of two different data structures. The trie data structure and the b-tree data structure. The trie gets its name from retrieval due its ability to retrieve text very efficiently. The B-tree is the main data structure Microsoft SQL server uses to store its data.

## Linear Array

To understand why any particular data structure is efficient, it is necessary to understand why the data structure is inefficient. Array’s, perhaps good for a much smaller set of data, simply does not match the needs of this project. Search times through a list of 200,000+ words would be very slow. User waiting times would be unsatisfactory and beyond acceptable standards.

This applies to unordered arrays using linear searching and with ordered arrays using binary search algorithms that are typically O(log n) due to the fact the algorithm requires a return of many possible words to successfully calculate the best option to ensure maximum point coverage, thus the average case would be O(n).

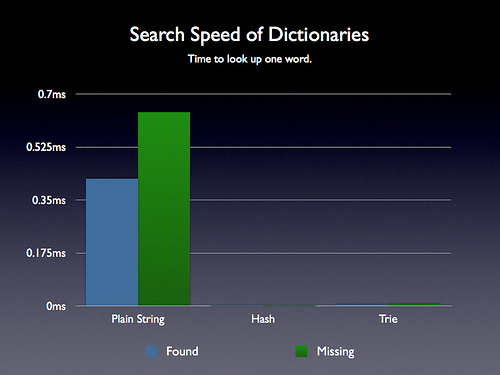


Figure 6.a Benchmark Results of String Operations

## Trie

A trie is an ordered tree data structure that is used to store a set of items, usually strings. Unlike binary search trees, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. Each node contains either an array of nodes of a list of pointers to other nodes. Each of these nodes represent a letter in the alphabet. There is also a flag in each node to say whether it is the end of a word or not.

Looking up if a word is in a trie takes O (n) operations, where n is the length of the word. So for array implementations the lookup speed doesn't change with increasing Trie size. It has been used to store large dictionaries of English words in spelling-checking programs and in natural-language programs. This theory makes it ideal for a word searching algorithm need to work in the scrabble game

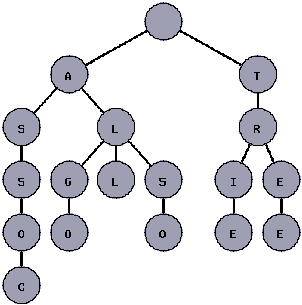
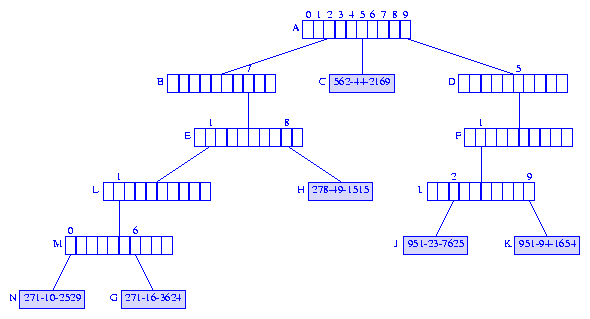


Figure . Illustration of Trie Broad Span Nodes

Figure 6.c Illustration of Trie Nodes

## B-Tree

B-trees are self-balancing tree data structure that keep data sorted. The main advantage of b-trees is that they allow all procedures performed on the tree such as searching, insertion and deletion in logarithmic time. A B-tree has an order (m) number which is the number of entries in a single node. A node can have m+1 children. Nodes can be split or joined depending on the situation. Unlike most tree, B-trees start on the bottom and create nodes above.

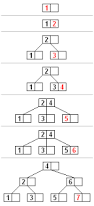


Figure 6.d B-Tree Nodes

As seen from the figure above, when a node becomes full it splits into two node and moves the middle item to the node above or creates a new node if it doesn’t exist. The same principle exist for deleting an item in the tree. Nodes are joined together to balance out. For databases the max number of layers is normally three, so it will never take more than two steps to find any piece of data.

The main disadvantage of a B-tree is that all the nodes will ever be full, therefore wasting space in memory or wherever the data is stored.

## Performance Analysis (Secondary Research)

The following chart is a piece of performance analysis carried out on a word based puzzle game that involves potential word combinations on a grid. Although the game in question is somewhat dissimilar to Scrabble, the underlying operations on String searches are somewhat similar.

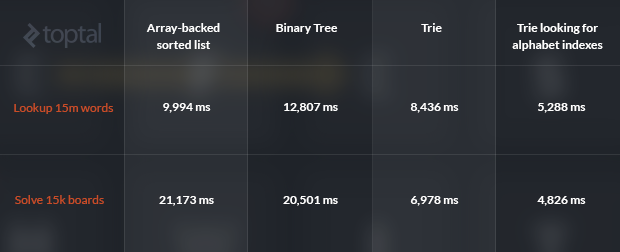


Figure 6.e Timings for Data Structures Tests

# Core Algorithms

The application’s core functionality will be centred on a key set of algorithms that will be involved will the span searching of very large data structures that contain the entire EOWL. Operations will be carried out on subsets of the data structure depending on particular variable aspects such as leading characters, trailing characters, etc.

The algorithm for the B-tree will be simple queries to a database and the databases built in function will deal with the complex functions. The trie algorithms will be coded in the project. The main algorithms will be insert into a Trie and search a Trie. The other main algorithm will be to loop through every possible combination of letters.

Finding possible combinations

For every set of letter there will be an n number possibilities, where n = number of characters in the set. The algorithm will required a huge amount of processing and there will be many different attempts of this algorithm to find the efficient algorithm.

PSEUDOCODE

1. Find combination of letters
2. Send it to the search method.
3. If string is a word then store it
4. Change String and loop back to start

Insertion into a Trie

To insert a key K into a trie we begin as we would to search for the key K, possibly moving down the trie, following the appropriate Children fields of the branch nodes, corresponding to the characters of the key. At the point where the Children of the branch node leads to NULL, the key K is inserted as an information mode.

PSEUDOCODE. Any insertion would ideally be following the below algorithm:

1. Find the place of the item by following bits
2. If there is nothing, just insert the item there as a leaf node
3. If there is something on the leaf node, it becomes a new in inner node depending how the item to be inserted and the item that was in the leaf node differs.
4. Create new leaf nodes where you store the item that was to be inserted and the item that was originally in the leaf node.

Searching a TRIE

To search for a key k in a trie T, we begin at the root which is a branch node. We search through the children of the node to see if the character exist. If it does we move to that child and repeat the process on the next node. This process continues until the end of the string or a node is not found.

PSEUDOCODE. The search algorithm involves the following steps:

1. For each character in the string, see if there is a child node with that character as the content.
2. If that character does not exist, return false.
3. If that character exist, repeat step 1.
4. Do the above steps until the end of string is reached.
5. When end of string is reached and if the marker (NotLeaf) of the current Node is set to false, return true, else return false.

# Functional Specification Data Requirements

## Dimensions

The system will need to store date about particular data dimensions in order to record progression. Heading data objects will be noted as Dimensional indices. The following list of objects will be included in the system data model:

User - D001

Scores (History) - D002

Characters - D003

## Metrics

Metrics will be included in the project as data about dimensions. These will be used for various metriculation purposes.

### D001

* Username
* Password
* Date of Registration
* Game Ranking

### D002

* User Score
* A.I Score
* Date of Game
* Point Difference
* Progressive Deviation
* Outcome Ranking

### D003

* Distribution Count
* Individual Score
* Board Location
* Distribution Frequency

## FS Dimension-Metric Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metric | D001 | D002 | D003 | Priority |
| Username | X | X |  | H |
| Password | X |  |  | H |
| Date of Registration | X |  |  | L |
| Game Ranking | X | X |  | M |
| User Score | X | X | X | H |
| A.I Score |  | X | X | H |
| Date of Game | X | X |  | L |
| Point Difference |  | X | X | M |
| Progressive Deviation | X | X | X | M |
| Outcome Ranking | X | X | X | M |
| Distribution Count |  | X | X | H |
| Individual Score | X | X | X | H |
| Board Location |  | X | X | H |

# Conclusion

The aim of the project is to research and evaluate different types of algorithms to manipulate and control strings. The choice of algorithms in a project can exponentially chance the efficiency of a project. These algorithms will be tested on the common game ‘Scrabble’.

This project will also demonstrate the procedure of implementing artificial intelligence in a program. Artificial intelligence is a major part of current and future computer development and this project show how it can be implemented in every sort of software.

# References

|  |  |
| --- | --- |
| [1] | Unknown, “The Trie Data Structure; Examples in Java,” TopTal, 1 November 2015. [Online]. Available: http://www.toptal.com/java/the-trie-a-neglected-data-structure. |
| [2] | SQLITY, “B-Trees How SQL Server Indexes are Stored,” SQLITY, 2 November 2015. [Online]. Available: http://sqlity.net/en/2445/b-plus-tree/. |
| [3] | “Scrabble - The Pixie Pit,” The Pixie Pit, 01 10 2015. [Online]. Available: http://thepixiepit.co.uk/scrabble/dictions.htm. [Accessed 01 10 2015]. |
| [4] | Hasbro, “Scrabble Hasbro,” 04 10 2015. [Online]. Available: http://scrabble.hasbro.com/en-us. |
| [5] | A. W. A. &. G. J. JACOBSON, “The World's Fastest Scrabble Program,” *Communications of the ACM,* vol. 31, no. 5, 1988. |
| [6] | “Merriam - Webster Scrabble,” Merriam - Webster, 04 10 2015. [Online]. Available: http://www.merriam-webster.com/game/scrabble.htm. [Accessed 04 10 2015]. |
| [7] | R. Bell, “A Beginner's Guide to Big O Notation,” Rob Bell, 28 October 2015. [Online]. Available: https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/. |

# Table of Figures

[Figure 3.a Conceptual View of the Scrabble Engine System 5](#_Toc435546913)

[Figure 3.b Hasbro Scrabble Design 6](#_Toc435546914)

[Figure 4.a System Use Case Diagram 7](#_Toc435546915)

[Figure 6.a System Log In Alpha Design 14](#_Toc435546916)

[Figure 6.b System Sign Up in Alpha Design 14](#_Toc435546917)

[Figure 6.c System Main Game in Alpha Design 15](#_Toc435546918)

[Figure 7.a Benchmark Results of String Operations 16](#_Toc435546919)

[Figure 7.b Illustration of Trie Broad Span Nodes 17](file:///C:\Users\kmjb\Dropbox\CS%20Project\Documentation\Functional%20Research%20Report.docx#_Toc435546920)

[Figure 7.c Illustration of Trie Nodes 17](#_Toc435546921)

[Figure 7.d B-Tree Nodes 18](#_Toc435546922)

[Figure 7.e Timings for Data Structures Tests 18](#_Toc435546923)