

2021

**Student Name:** Jaynik Parsotomo

**Student Number:** 199009725

A Scrabble Computer Player

Interim Report

Contents

[1. Introduction 3](#_Toc88742065)

[1.1. Introduction 3](#_Toc88742066)

[1.2. Aims 3](#_Toc88742067)

[1.3. Objectives 3](#_Toc88742068)

[2. Survey of Literature 4](#_Toc88742069)

[2.1. Introduction 4](#_Toc88742070)

[2.2. Requirements of Gaming Algorithms 4](#_Toc88742071)

[2.3. Gaming Algorithms 4](#_Toc88742072)

[2.3.1 Cross Checks and Anchors 4](#_Toc88742073)

[2.3.2. Backtracking 4](#_Toc88742074)

[2.4. Dictionary Storage 5](#_Toc88742075)

[2.4.1. DAWG 5](#_Toc88742076)

[2.4.2. GADDAG 5](#_Toc88742077)

[2.5. Technical Specification 7](#_Toc88742078)

[2.6. Conclusions 7](#_Toc88742079)

[3. Architecture and Design 8](#_Toc88742080)

[3.1. Software Architecture 8](#_Toc88742081)

[3.2. Software Design 9](#_Toc88742082)

[3.2.1. Use Case 9](#_Toc88742083)

[3.2.2. Components Diagram 10](#_Toc88742084)

[3.3. Interface Design 12](#_Toc88742085)

[4. Planning and Timescales 14](#_Toc88742086)

[Bibliography 16](#_Toc88742087)

Text, letter

Description automatically generated

# 1. Introduction

## 1.1. Introduction

Artificial intelligence, AI for short, is a phrase used to describe the ability of a machine or a program to think and learn. AI is being used in many different industries to help perform tasks and one of the most prominent industry where AI is being used is, the gaming industry. The gaming industry has used AI to enhance the user experience to deliver realistic interactions to provide a more human-like intelligence, like non-player characters.

Therefore, for this project I will be implementing an AI to play the classic word game, Scrabble, invented by Alfred Mosher Butts in 1933 [1].

## 1.2. Aims

The primary aim of this project is to create a computer player that plays Scrabble, against a human. This aim is expanded further by using gaming algorithms to enhance it.

## 1.3. Objectives

The main objectives of this project are:

* To investigate game playing algorithms that can be used with scrabble.
* To investigate dictionary storage data structures.
* To implement an efficient computer player that plays scrabble.
* To implement the dictionary data structure.
* To create a good UI.

# 2. Survey of Literature

## 2.1. Introduction

The first research conducted in this project, was to familiarise myself with Scrabble, so I can implement this knowledge in the design and code. I used the current Scrabble owner Hasbro, main site [2] to learn about the rules and components needed.

Next, I learnt about general points about gaming algorithms, to ensure when researching I am looking at the correct material. This allowed me to conduct the main research concerning this literature review which are algorithms that can be used in Scrabble.

## 2.2. Requirements of Gaming Algorithms

The main three things the AI needs to determine the best move, are steering, pathfinding and goal orienting according to a publication on towards data science [3].

Steering refers to the control movement of entities in the game. In the case of Scrabble, it would be the movement of the tiles, for this to work it would need to know the rules of the game.

Pathfinding refers to finding the shortest route. This is important for gaming AIs as; you need a good pathfinding algorithm to quickly find what the AI needs. In Scrabble the pathfinding would be used to find the best legal word, hence it needs to know which moves are legal to play.

Goal orienting means to provide knowledge on what it is trying to achieve, which is basically the terms to winning or losing a game. In Scrabble the AI needs to know how to win, which is to get the most points from words.

## 2.3. Gaming Algorithms

### 2.3.1 Cross Checks and Anchors

In Scrabble the computer player must scan the entire board to locate the position a word can be played. From this it checks if it can be legally placed, meaning any connecting letters are also legal words. As such, algorithms I have found to aid with this are cross checks and anchors as stated in a journal paper by Appel and Jacobson [4].

Anchors refer to a position on the board that is empty or at least one adjacent non-empty square, and it must be adjacent to an already placed letter. This allows the computer player to focus on that row, to which it can place the word to the left part of the anchor or the right of an already played word thus connecting the word.

Cross-checks are when placing a word parallel to another word, the letters in the new word need to be a valid word with the letters perpendicular to it. In other words, it checks each column of the letters to check if they are valid.

### 2.3.2. Backtracking

To search the board for valid places to place the letters, the authors in the journal paper [4] showcase a backtracking algorithm. Essentially, what the algorithms does is, once the human player has placed a word, it searches the word to see if it can extend the word from the anchor. First it searches the left part of an anchor and checks if the word can be extended by checking the dictionary.

Text, letter

Description automatically generatedA picture containing text, receipt

Description automatically generatedIf it cannot find a prefix, then it searches the right anchor to check if it can append it by checking the DAWG (section 2.4.1). This is combined with cross checks to place the word.

Figure 1: Pseudo-code to find extension of right part of anchor. Source [4].

Figure 2: Pseudo-code for searching the left part of anchor. Source [4].

## 2.4. Dictionary Storage

The main objective of Scrabble relies on a dictionary to find valid words. Therefore, I researched algorithms that can be used to by the computer player to find words efficiently.

### 2.4.1. DAWG

Diagram

Description automatically generatedDiagram

Description automatically generatedDAWG, short for Directed Acyclic Word-Graph, published in a journal paper by Andre W. Appel and Guy J. Jacobson [4], is a way to represent the dictionary to create a fast move generation algorithm. The algorithm is used with a vast dictionary which represents the dictionary in a graph structure and allows for less nodes to be used, hence less memory then a trie. The difference is shown in figures 3 and 4.

Figure 4: A DAWG for the given lexicon. Source [4].

Figure 3: A trie the given lexicon. Source [4]

### 2.4.2. GADDAG

GADDAG published in a journal paper by A. Gordon [5] is an improved version of the DAWG. The GADDAG can be noted as a two-way DAWG since this solves a disadvantage of the DAWG which is, it is unable to search in both directions at once. Gordon states the GADDAG is finite automaton which avoids non-deterministic prefix generation unlike the DAWG. This allows the GADDAG to search bidirectional, which means it is a faster algorithm.

GADDAG is basically a DAWG, but with extra additions which allows to accommodate substring matches. The extra additions are that for every prefix of the word, which is also reversed, a separator marker to differentiate the 2, is added to the remainder of the word. The prefix of a word refers to the break down, for example BIKE -> prefix: B -> BI > BIK -> BIKE.

In the paper the author uses the example of CARES.

Represents the separator.

C -> C ARES

CA -> AC RES

CAR -> RAC ES

CARE -> ERAC S

CARES -> SERAC

Diagram

Description automatically generatedThe use of this is, if the RE is on the board, then reversing that we get ER. Then searching the GADDAG it will find ERAC+S. What this means is that we can add the CA at the start and append S to RE to get CARES.

Figure 5: Subgraph of unminimized GADDAG for the word CARE. Source [5]

Diagram

Description automatically generatedText

Description automatically generatedUnlike the DAWG, the GADDAG is storing a lot more words since it is searching bi-directional from any letter of a word. This makes the GADDAG a lot larger than the DAWG for the same dictionary. Therefore, a way Gordon suggests decreasing the memory used is by minimising the GADDAG data structure. This is done by merging common prefix and suffix paths.

Figure 7: Semi-minimised GADDAG for the word CARE. Source [5]

Figure 6: Pseudo-code for GADDAG construction. Source [5].

## 2.5. Technical Specification

In this section I will discuss the programming technology that I will be using to create this game. Playing games has never been easier because of the internet, which allows anyone to play games on the web without any downloads. Thus, I have decided to create Scrabble as a web application.

There are many ways to build a web application, and the most popular language to build them is to use JavaScript with HTML and CSS. For this reason, I will be using JavaScript as it is compatible on most browsers, hence any browser that supports JS will be able to load this game.

However, for this project I will not be using vanilla JavaScript, instead I will use React. React is a frontend JavaScript library, which is used to build UI components [6]. This will be very useful in the creation of my Scrabble game as the main factor an end user cares about are the UI, thus React will allow me to build a good UI. Also, React can create SPA, single page applications, easily which will be helpful in this project since the game will only require a single page.

Furthermore, although I do not require a database as the only data, I will be storing is a dictionary text file, React can be connected to one. If I need to connect one or use another language for the non-UI functions, I can easily connect react to a Java backend [7] if required.

## 2.6. Conclusions

This literature review investigated algorithms used in previous Scrabble implementation. From this I can state which algorithms will be helpful to implement in my own Scrabble game. I will be incorporating the algorithms like anchors and cross checks, to make the search for valid words more efficient. In addition, I intend to use the GADDAG by Steven Gordon, as it is specifically related to Scrabble which allows the fastest move generation compared to other algorithms I have seen. I have also decided upon how I will create this application, which will influence my design choices in the next section.

# 3. Architecture and Design

In this section I will be discussing the software architecture I have decided upon and the design of the software. This section will allow me to distinguish what components the system needs and to create a design concept that will help me in the implementation.

## 3.1. Software Architecture

The software architecture is what defines the structure of the systems, as to how each element interacts with each other. The game requires a good GUI, section to process the data and a dictionary of scrabble words, to support this I will be using a 3-layer architecture.

The 3-layer architecture consists of:

* Presentation layer
  + Where user will interact with the system.
* Logical layer
  + This layer will be used by the other layers to communicate with each other. It will receive data from the presentation layer and apply the game logic to it. It also retrieves data from the data layer to be used with the GADDAG algorithm.
* Data layer
  + Where the dictionary words will be stored.

Diagram

Description automatically generated

Figure 8: Diagram demonstrating how my 3-layer architecture will work

## 3.2. Software Design

For this project I have decided to use React therefore, the class diagram has been adapted. Instead of the conventional class diagram, since React uses components I have adapted the diagram to fit this criterion. See section 3.2.2.

### **Diagram Description automatically generated**3.2.1. Use Case

Figure 9: Use case diagram

The use case diagram showcases all the features each user will use. For this system I have 2 actors, the computer player and human player. I have only defined 2 actors for the system since I am focusing on an AI that can play against a human, hence only 2 users.

The computer player use cases are all related to “moves”. The focus of the AI is to generate a move and to ensure the move is valid, hence the uses cases defined for it. The generate move use case has three prerequisites it must also use which are scanning the board to find the valid positions, looking up the dictionary to find matches and then deciding on which move. This will link into the validate move use case to ensure the move is legal, using the anchors and cross checks. If the move is valid, it can calculate the score.

The human player use cases are all the interactions the user will have with the system. The user will move tiles around the board to form a word, hence the move tiles use case. The player will then interact with the system to submit the word. When submitting the word, a validation check will be done, if it passes the score will be calculated. The reason why I have included this in the play word use case is because the user can use this precondition to check if a word, they are unsure about, is valid or not, hence becoming a used feature. The user will also have the option of passing their turn or swapping tiles. Both use cases contain the optional choice of cancelling this process hence the extension.

### 3.2.2. Components Diagram

A picture containing text, receipt

Description automatically generated

Figure 10: Components Diagram

This is my adapted class diagram which uses components instead of classes. React uses components which are essentially the classes and functions, but the difference being these components are building blocks to render the UI. Therefore, I have created a components diagram, with each component being represented by a blue box. Each component contains the following data: name, child components, helpers, variables, and functions. The green boxes are the entities I will need to consider, since they each require different features. The red boxes in the diagram are helpers, who will basically contain the game logic or keep track of data, they will not render any UI aspects.

I laid out the diagram in a sort of tree manner, since in react all the rendering will happen in a single html file. I have structured it to resemble the way it will appear in the file. A basic overview is the game component will render the game and will decide when to switch the players turns and end the game. The board component is the main UI element, which is created using squares, where the tiles will be placed. Players is split into 2 entities since they require different functionality. Both will be using the rack to form words and move the tiles around. The computer player will use the helper generate move to form its words. Then the moves will be validated, and scores will be calculated if validation passes. The links between each component are subjective to change, if during the implementation I find a better way to do it.

## 3.3. Interface Design

Chart, treemap chart

Description automatically generatedChart, treemap chart

Description automatically generatedChart, treemap chart

Description automatically generatedThe interface designs I have created will help me in knowing what the final product should look like. I can use it a guide for the layout and aesthetics when implementing the game. I created them using Gravit [8] designer application. This allowed me to produce a medium fidelity prototype.

Figure 11: UI before game is started

Figure 11: UI before game is started

Figure 12: User moves tiles on to the board

Figure 12: User moves tiles on to the board

Chart, treemap chart

Description automatically generated

Figure 13: After a turn is completed

Figure 13: After a turn is completed

Figure 14: Computer players turn

Figure 14: Computer players turn

Figures 11, 12, 13, 14 showcases what the UI will look like for the majority duration of the game. Figure 11 shows the UI before the game has started. This figure shows how the Scrabble board is the main attraction with the other UI components around it. When the user moves a tile from the rack to the board shown in figure 12, it shows how the button pass will be changed to submit. Figure 13 indicates what happens when it is the computer players turn. It will display a pop-up message with an overlay to prevent the user from interacting with the system. Once a word has been played, the scores will be calculated, and the UI will be updated as shown in figure 14. The scores, last word played, and tile bag count has been updated.

Chart, treemap chart

Description automatically generated

Figure 15: Swap tiles

Figure 15: Swap tiles

Figure 15 displays the UI when the user wants to swap tiles. A tray will pop-up with slots for the tiles to be swapped to be placed in. The buttons will change accordingly for cancellation and confirming the swap.

# Graphical user interface, application Description automatically generatedA picture containing chart Description automatically generated4. Planning and Timescales

Figure 16: View of Gantt chart for completed tasks

My time on this project has been focused on researching and designing, meeting the first 2 objectives. This is shown in the above Gantt chart, where I spent the majority of October researching algorithms. By spending this time, I was able to explore several algorithms by reading journal papers of previous Scrabble implementations, discovering the different algorithms and data structures. By doing so I can use these algorithms in my own implementation without having to reinvent the wheel, thus saving time during the implementation stage.

Graphical user interface, application, table, Excel

Description automatically generatedFor the rest of the time, I spent it on designing the system. The reason I spent most of my time on this is so when I move on to the implementation, I already have a well design to follow and clear idea as to how the system components should work together. Therefore, it minimises the chance of logical errors when coding and no time wasted on deciding on the layout and look.

Figure 17: Gantt chart for November 27th, 2021, to December 28th, 2021

Figure 17: Gantt chart for the rest of this semester

For the rest of this semester, I will be starting the implementation stage. I will begin by setting up the react application, creating the skeleton. Then because the focus of this project is the computer player, I will be focusing my time on implementing this. Furthermore, since I will be working on other coursework’s I have been realistic with the time scale, ensuring I leave time for the other work.

Graphical user interface, application, table, Excel

Description automatically generated

Figure 18: Gantt chart for semester 2

Figure 7: Minimised GADDAG for the word CARE. Source [5]Figure 18: Gantt chart for semester 2

For semester 2, for the first half I will still be working on implementing features, however because I do not know the schedule for semester 2, I have only put some of the more important features to implement. I have made sure to allocate enough time to do these features and made sure to leave gaps, for other work and time to prepare for the interview.

## Bibliography

1. Hasbro, “Scrabble history: Making of the classic American Board Game,” *scrabble*. [Online]. Available: <https://scrabble.hasbro.com/en-us/history>. [Accessed: 4-Oct-2021].
2. Hasbro, “Scrabble rules: Official Word Game Rules: Board Games,” *scrabble*. [Online]. Available: <https://scrabble.hasbro.com/en-us/rules>. [Accessed: 11-Oct-2021].
3. S. Carryer, “The brains in games: Video game ai,” *Medium*, 10-May-2019. [Online]. Available: <https://towardsdatascience.com/the-brains-in-games-video-game-ai-d0f601ccdf46>. [Accessed: 15-Oct-2021].
4. A. W. Appel and G. J. Jacobson, “The world's fastest Scrabble program,” *Communications of the ACM*, vol. 31, no. 5, pp. 572–578, 1988.<http://www.cs.cmu.edu/afs/cs/academic/class/15451s06/www/lectures/scrabble.pdf>
5. S. A. Gordon, “A faster Scrabble move generation algorithm,” *Software: Practice and Experience*, vol. 24, no. 2, pp. 219–232, 1994. <https://ericsink.com/downloads/faster-scrabble-gordon.pdf>
6. “React – a JavaScript library for building user interfaces,” *– A JavaScript library for building user interfaces*. [Online]. Available: <https://reactjs.org/>. [Accessed: 24-Nov-2021].
7. B. Bachina, “How to develop and build react app with Java backend,” *Medium*, 27-Nov-2020. [Online]. Available: <https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-react-app-with-java-backend-c1e6c5c93ae>. [Accessed: 24-Nov-2021].
8. *Gravit Designer*. [Online]. Available: <https://www.designer.io/en/>. [Accessed: 24-Nov-2021].