Designing and Evaluating Performance in Computer Based Word Game: The Nigerscrab Experience

Y. O. Folajimi
Department of Informatics & Centre for Computational Intelligence
DeMontfort University
Leicester, United Kingdom
yetunde_folajimi@yahoo.com, yfolajimi@dmu.ac.uk

ABSTRACT
Computer games are important elements that have significantly approbated our culture, being a primary driving force for social and economic development. One of the most popular and most interesting games is Scrabble, a board game in which players compete by forming words on a common 15-by-15 grid. The essence of this paper is to discuss the implementation of an improved model for evaluating the rack contents and generating moves in scrabble using probability and heuristics search. Accordingly, we present experimental results of the implemented model and justify the fact that the model actually improved the game strength and gamers’ experience

Keywords: Computer game, Scrabble & NigerScrab

1. INTRODUCTION
Scrabble is a popular crosswords game played and embraced by millions of fans all over the world. Contestants play the game by forming words on a board of 15 x 15 squares, just like in crossword puzzles. Each player at any point in time has a rack containing seven tiles that are randomly picked from a bag that initially contains 100 tiles. To achieve high scoring words, the player uses strategies that strike a balance between maximising his score and managing his tiles for the purpose of achieving high scores in future. Scrabble is clearly a game of imperfect information since a player neither has the knowledge of what is contained in the opponent’s tiles nor is able to identify what tiles he would select next from the bag. It is important to note that Scrabble, unlike chess or checkers, is a proprietary game, the rights of which are owned by Hasbro, Inc.

NigerScrab, a Scrabble-like game, follows closely the rules of Scrabble by looking at millions of possible moves in each turn, guided by probability with heuristics for an evaluation function that considers a large number of conditional circumstances which it uses to decide on which move to make out of numerous available options. These considerates are so massive that they cannot all be completely mentioned but some comprehensive details are given in subsequent sections of this work.

The past successes of AI researchers at Scrabble games, the methodologies and techniques used have revealed encompassing solutions of divergent degrees and procedures, such as finding the best word from a seven-tile rack, or computing the most eminent scoring play available from a given game situation. All these have motivated investigations into the workings of these programs, in which research has clearly revealed that the “brute-force” algorithms assumes that the highest-scoring play is always the best, and is not in the interest of Scrabble game as a whole [1], [2], [3], (Edley, 1997; Sheppard, 1999; Richards and Amir, 2007).

This paper presents our efforts in designing and implementing a variant of the popular Scrabble game, NigerScrab. The game engine can make use of AI tools at its disposal to manipulate all aspects of game-play so well that it is able to defeat competing human opponents. The paper further presents experimental results on the implemented model and a justification that the model has actually improved the game strength and gamers’ experience.

1.1 Overview of Scrabble
The conventional board game, Scrabble (copyright Hasbro, Inc.) is a word game for 2, 3 or 4 players whose aim is to form interlocking words in the style of a crossword, in which words can only be positioned vertically or horizontally on a 15X15 grid board whose rows are numbered 1 to 15, and columns are numbered A to O. This gives ease of referencing, in which each grid is identified by the row and column number; for instance, 1A represents the square on row 1, column 15. At the outset of the game, each player begins his/her turn by drawing seven tiles each from
a bag initially containing the 100 tiles, including 2 blanks that can represent any letter.

To start the game, the initial word to be placed on the board must connect to the centre star square in position 8-H; all subsequent words placed must use at least one square adjacent to a tile already on the board. In every turn, the player has the option of placing a word, exchanging tiles or passing. Exchanging tiles allows a player to replace between one and all of the tiles on the player’s rack and lose a turn. A player may pass at any time and when both players pass consecutively, the game ends. When a player places tiles on the board, that player draws new tiles from the tile bag, accumulating more tiles until that player’s number of tiles equals 7, or until there are no more tiles in the bag.

If a player is able to place all the seven tiles contained in his/her rack on the board at the same time, in a single move, that player receives an extra 50 point bonus score, commonly called a bingo. The game ends when all of the tiles have been taken from the bag and one player has used all of the tiles in his rack. On ending the game, the total points for tiles remaining in the rack of each of the players is subtracted from the final score; the player that finishes the game has an additional points equivalent to that subtracted from the opponent’s rack. The point values of the letters, including the letters associated with those point values are shown in Table 1:

Table 1: Scrabble Letters Point Values

<table>
<thead>
<tr>
<th>Tile</th>
<th>Score</th>
<th>Tile</th>
<th>Score</th>
<th>Tile</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>J</td>
<td>8</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>K</td>
<td>5</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>L</td>
<td>1</td>
<td>U</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>M</td>
<td>3</td>
<td>V</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>N</td>
<td>1</td>
<td>W</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>4</td>
<td>O</td>
<td>1</td>
<td>X</td>
<td>8</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>P</td>
<td>3</td>
<td>Y</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
<td>Q</td>
<td>10</td>
<td>Z</td>
<td>10</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>R</td>
<td>1</td>
<td>Blank</td>
<td></td>
</tr>
</tbody>
</table>

Apart from the fact that the score of the first player is doubled, the squares on the board upon which the tiles fall actually determine the final score of the word formed because some of the squares represent letter or word multipliers and are designated double-letter square, triple letter square, double letter word or triple letter word. The score of any letter that falls on 2-letter square or 3-letter square will be multiplied by 2 or 3, respectively, while the entire word will be multiplied by 2 or 3, respectively, if any part of it falls on a 2 or 3 word square. This is only applicable to the first player to utilize the extra point square and not to another player who utilizes them. A detailed rule of playing the game is contained in Hasbro Website, http://www.hasbro.com/.

Figure 1. Conventional Scrabble board game in progress.
Source: http://fredzone.files.wordpress.com/

2. RELATED WORKS

Most of the ultimate dreams of Artificial Intelligence have been realised by studies in game playing, including a chess program that succeeded in defeating the world champion in a 1996 tournament [4]. In another game one year afterwards [5] and [6] also confirmed CHINOOK as the first checkers program to overcome a human world champion in any game. Over the years, scrabble research has revealed that the underlying factor in Scrabble AI is that a successful Scrabble engine consists of many components, like full width search, evaluation functions, move generator, move evaluator, self-play, feedback, indirect comparisons, knowledge engineering, perfect information search techniques, statistical look-ahead, opponent modelling / opponent search and so on.

In the early days of computer gaming, Stuart Shapiro and others of SUNY Buffalo implemented several SCRABBLE-playing programs in SIMULA and Pascal on a PDP-10 [5], [6]. They represented their lexicon as a tree-structure of letters where each path down the tree has an associated list of words that can be formed using exactly those letters on the path. Peter Turcan later wrote a Scrabble-player which generates moves by iterating over the words in his lexicon in reverse order of length in 1981 [7]. As reported by [8], Turcan’s program does not make use of any adversary
search, but rather uses an evaluation function more sophisticated than the score of the prospective move. It takes the score and conditionally adds terms depending on simple strategic features of the new position and tiles left in the rack. However, one major drawback of this program is poor endgame performance.

Steve Gordon [10] reports on Peter Weinberger’s Scrabble-playing program that was originally designed to work on a PDP-11. Weinberger’s move generator first constructs a set of position descriptors, one for each place on the board where a legal move might be made. Weinberger’s program has no concept of strategy, and simply chooses the highest-scoring play available. Subsequently, Appel and Jacobson [8] then published their own move generator which was a landmark in Scrabble move generation and gave rise to more eminent interests in Scrabble programming. James Cherry [9] invented AcBot, a unix-based Scrabble-like robot that has an internal representation of the board and its rack, which it uses in conjunction with its dictionary to decide on which move to make a move based on a set of rack-leave rules derived by Steve Gordon [10]. Gordon [10] identifies some shortcomings with the rack leave scores and endgame of ACBot, and thus derived a set of rack-leave rules which was used in conjunction with James A Cherry’s dictionary [9] to come up with an improved game, in which some of those shortcomings were addressed although a solution was yet to be found to the endgame ambiguity.

Other interesting Scrabble bots have also evolved. One program that stands out uniquely in the recent trends in the world of Scrabble innovation and competitions is Maven, developed privately by Brian Sheppard and concluded in the early 1990s [10]. Since 1988, when Maven resolutely defeated human World Champion Ben Logan in a 1998 exhibition match, the National Scrabble Association has used Maven as a standard for championship games. According to Brian Sheppard [11], where a restrained description of Maven’s workings was given, the program is divided into three distinct game states: mid-game, pre-endgame and endgame.

Although Maven has got to the level at which it can defeat world champions; its design did not take opponent modelling into account because, Sheppard [2] mentioned that it is not considered a factor in modelling strong Scrabble. However, recent research has indicated that opponent modeling is another strategy that has great influence in better performance of Scrabble engine [3]. Katz-Brown and O’Laughlin [12] used the same architecture as Sheppard’s Maven to implement Quackle, an open source Scrabble-like program in March 2006. Their design made use of a static evaluation function to categorize a list of potentially beneficial moves by the candidate and then decide on which move is the best based on simulation results.

3. METHODOLOGY

Figure 2 demonstrates a progression stipulated by Sturm [13] which we adopt for the purpose of this study. It reveals the models that enable straightforward and realistic progression from one type of diagram to another in Visual Basic programming, that is use case diagrams, sequence diagrams, class diagram and activity diagrams, which were carried on to NigerScrab game programming. The methodology follows an iterative and incremental lifecycle. Consequently, our implemented system evolved by going through several iterations, each of which consists of all the primary tasks of analysis, design, and implementation / construction.

The design phase involved detailed requirements definition for each class to include its attributes and methods. Consequently, the objects collaborate with one another in order to perform the various scenarios described by the use cases. Objects that represent the same entities are grouped together to form classes. We documented these classes and their various relationships from the problem domain by class diagrams. These diagrams as well as the representations of the relationships among the classes are included in the Unified Modelling Language (UML) which is now the reference point for the OOAD methodology. We, thus, built the system by using combinations of classes that have been developed previously (reuse) in addition to developing new ones using the specifications defined during the design phase. Finally we evaluated the resulting system by testing as suggested by Booch in [14].

![Figure 2. UML Process for visual Basic](Source:Sturn 1999)
Using the existing rules for the game, together with practical experience of playing it, we determined the use case for a human player making a move description. The summary is that a complete move by a human player involves the letter-by-letter construction of a new word or words on the board. The move is checked for validity when it is completed, the board is permanently updated if it is valid, and the player's score is incremented. Figure 3 depicts NigerScrab with the associated use cases.

![Diagram](image_url)

**Figure 3. NigerScrab with Associated Use Cases**

### 4. DESIGN ARCHITECTURE

Our design partitioned the system into a GUI, functional core, and data repository. The GUI depends on the core because it invokes operations of the core, and the core depends on the GUI by invoking GUI operations to display the result of computations and other activities. The core also depends on the repository to generate log files, move history and consult the dictionary. Figure 4 shows the architectural design for NigerScrab.

The GUI is designed to manage the display of the board and racks, bearing in mind the constraint that only the rack of the current turn player should be visible). The design also includes a module to check the correctness of moves (move validation module) and a module to generate moves automatically (move generation modules). There are also data management modules, to hold dictionary data and game history data.
5. RESULTS AND DISCUSSION
Despite having a nice structure and game logic, the game interface presentation should adequately represent what it does, and the graphical user interface should contain clear and easily distinguished facilities for interacting with the game engine. Figure 5.2 presents the graphical user interface (GUI) of NigerScrab. The following utilities and information are available for the user to interact with the game:

- Game board (contains 15 x 15 grid labelled A to O and 1 to 15)
- Letters racks (human tiles are visible while NigerScrab’s tiles are hidden)
- Remaining letters
- Duration
- Game history (contains words played and score)
- Toolbar: Offers the following utilities:
  - Cancel (tiles are withdrawn from board back to rack)
  - Pass (allows for skipping turn; player scores zero for that turn and tiles are exchanged)
  - Done (played tiles are committed)
  - Help utilities

![Figure 5. NigerScrab Graphical User Interface](image-url)
As shown in figure 5, the board and game view very much resemble the standard Scrabble game with the board positioned at the left and tiles and rack positioned side by side below the board. Players’ scores are positioned within the window containing the players’ tiles and rack. The game history is positioned by the right side of the board, while the dictionary is positioned at the extreme right.

5.1. Move Placement Algorithm
Once NigerScrab has performed all the clerical tasks of situation inspection, in most cases, it will be ready to attempt tile placements on the board by deconstructing the method by which human players search for moves. The search starts at the rack, where players pore over their assortment before directing their eyes to the board and their thoughts to placement options. Note that a placement is considered as any set of open positions that may accept tiles from a player’s rack, while a move is considered as a placement that may produce a valid, scoring play (in other words, a placement passes all checks preceding lexicon checks). As a matter of fact, NigerScrab’s algorithm should attempt all “plays” by iterating through all possible placements, skipping over those known immediately to be invalid. The placements procedure for a given arrangement A in a given row R is shown in figure 6:

```plaintext
If row_position < (15 – a.size):
  next row_position
Else
  Letter = InStr(A)
  X = Remaining_Pos – Remaining_letter
  IF X < 0 then
    End if
  Else If row_position < > 0
    next row_position
  else
    row_position = row_position + letter
    a = a - letter
    if a.size = 0
      end if
  else:
    next row_position
  else:
    exit row
```

Figure 6. The placements procedure for a given arrangement

The algorithm guarantees contiguous primary words, because each played word must connect to at least an anchor square (that is, an existing square upon which a tile has already been played before), simplifying the validation step. Every time the algorithm finds a play (that is, succeeds in placing each tile for the current arrangement) it passes the play to a validation subroutine. The same function that processes user “plays” (valid placements) can be used for NigerScrab’s attempts; compliance with the data structure provides this ease of functionality.

The play processing function, NeedLett returns a Play object, packed with information about placement attempt. First, the PassPos subroutine is checked for validity of word. If it passes, the score is compared with the current best play (initially an empty Play object with score 0), and if it is greater, this Play object becomes the new best play against which future iterations will check. This methodology guarantees that the highest scoring play possible will be identified.

5.2 Initial Performance of Permutation Algorithm
By trying each arrangement at each viable position, the best play should be found by validating each placement and saving the one with the highest score. This gives rise to the question of how long it would take. Earlier experiments disclosed that it will take over a minute in all, which is definitely too long [15]. It is important to observe that the time spent on each arrangement is directly proportional to the number of arrangements of that size, as expected. Moreover, the computations above do not take the two blanks into consideration. With two blank tiles included in the bag, there is a risk that the computer could possibly draw both, increasing the number of attempts to a computationally intractable value. As a matter of fact, if the game engine could adequately make use of blank tiles, it will likely guarantee additional bingo and expose NigerScrab to greater opportunities. However, to include wild tiles (blanks) in both computer and human selections, a special set of decision-making routines were proposed [16].

Table 2 gives a summary of game performance using the permutation algorithm as against Appel and Jacobson’s DAWG algorithm. In the permutation algorithms, we carefully avoided the use of blanks with permutation algorithm, as it leads to computationally untraceable time. The table indicates that the Appel and Jacobson’s DAWG algorithm clearly performs better in the area of CPU time and words spanned, especially with the ability to utilize the blank (spanning 1005.31 words per second without blanks and 513.1 words per second with blanks) as against permutation algorithm which was unable to utilize blanks and spanned only 156.45 words per second.
Table 2: Game performance using the permutation algorithm as against Appel and Jacobson’s DAWG algorithm

<table>
<thead>
<tr>
<th></th>
<th>Permutation Algorithm (without blank)</th>
<th>Appel and Jacobson’s Move generation algorithm (without blank)</th>
<th>Appel and Jacobson’s Move generation algorithm (with blank)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary size</td>
<td>80,000 (approx.)</td>
<td>80,000 (approx.)</td>
<td>80,000 (approx.)</td>
</tr>
<tr>
<td>CPU time (seconds)</td>
<td>79.5</td>
<td>4.31</td>
<td>18.12</td>
</tr>
<tr>
<td>Words spanned</td>
<td>12438</td>
<td>4333.67</td>
<td>9236.43</td>
</tr>
<tr>
<td>Per sec</td>
<td>156.45</td>
<td>1005.31</td>
<td>513.1</td>
</tr>
</tbody>
</table>

5.3 Algorithm Performance

The permutation algorithm, which does not employ the use of cross check set runs with a time complexity of $O((n+1)!(n-k)!)$, where $n$ is the number of letters and $k$ is the size of tiles in rack. However, with the introduction of the cross set check, ensuring reduction in the number of tree spanned. Since the hash function used naturally has a time complexity of $O(n)$, and in the worst case, the number of cross check to be done is $15 \times 15 = 225$ (there are 15 rows and 15 columns on the board. Therefore the cross-set-check algorithm has a time complexity of:

$$O((n+1-225)!(n-225-k)!)/O((n-224)!(n-225-k)!$$

Using a dictionary set of approximately 80,000 words, Table 3 gives the relative performance of our move generation algorithm in approximately 100 games and playing on HP, a dual core processor of 4gb RAM, when the algorithm was used with and without cross-check set.

Table 3: Move generation algorithm performance

<table>
<thead>
<tr>
<th></th>
<th>Permutation Algorithm (Without cross-check set)</th>
<th>Move Generation Algorithm (With cross-check set)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Per play</td>
<td>Per move</td>
</tr>
<tr>
<td>CPU time</td>
<td>31.54</td>
<td>0.0034</td>
</tr>
<tr>
<td>Time efficiency</td>
<td>n.a.</td>
<td>$O((n+1)!(n-k)!)$</td>
</tr>
<tr>
<td>letters spanned</td>
<td>68,253</td>
<td>5.46</td>
</tr>
<tr>
<td>Per second</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No of moves</td>
<td>73,256</td>
<td>38.5</td>
</tr>
<tr>
<td>Per play</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Suffix searched</td>
<td>62,443</td>
<td>31,201</td>
</tr>
<tr>
<td>With blank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Without blank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average score</td>
<td>19.4</td>
<td>381</td>
</tr>
<tr>
<td>Per game</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results indicate that the algorithm has ensured reduction in the number of words and letters transverse.

6. CONCLUSION AND FUTURE STUDIES

This work is built on a research into discovering ways of increasing the strength of a Scrabble game engine and has led us to the implementation of a specific variant of Scrabble game, named NigerScrab, which was derived from the game model. Through Design, testing, maintenance and experimentations, it was found that the strength of NigerScrab has actually improved and has players’ experience have also increase.

More than anything else, the techniques employed in this research has improved the quality of NigerScrab and we will not hesitate to adopt the same approach to new projects. Adapting NigerScrab for other dictionaries or other languages, especially Nigerian languages (NigerScrab adapted her name from Nigeria) to test if it is also
superhuman in other languages is an interesting task, though a great challenge. A good number of avenues can be explored so as to increase the features such as points per game, bonuses and so on. Extending studies on NigerScrab design to include an intelligent agent capable of enhancing the ability of players to adaptively learn vocabulary as they play with the game engine has been proposed in [17]. It is hoped that further research in this direction will propel NigerScrab into a an intelligent game based learning System

REFERENCES


ACKNOWLEDGEMENT:
The author gratefully acknowledge the Commonwealth Scholarships Foundation for providing the needed support towards the success of this publication

Author’s Brief

Dr Yetunde O. Folajimi is a Lecturer of Computer Science at University of Ibadan. She specializes in Artificial Intelligence with special interests in Serious Games and E-Learning, including digital game based learning, adaptive learning and Social Learning. She is the founding Director of the Initiative for Technology Empowered Education in Africa and currently a Commonwealth Research Fellow at De Montfort University United Kingdom. She can be reached at yetunde_folajimi@yahoo.com or yfolajimi@dmu.ac.uk. Phone +44 (0) 116 2076757