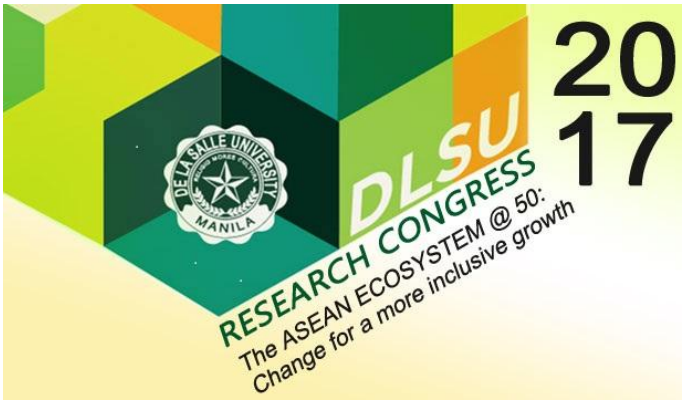

Presented at the DLSU Research Congress 2017
De La Salle University, Manila, Philippines
June 20 to 22, 2017

Word Games as Context in a Freshman Programming Subject

Florante R. Salvador*
De La Salle University, Manila
**Corresponding Author: florante.salvador@dlsu.edu.ph*

Abstract: We explore the use of word games as context for students to learn, design and implement basic data structures and algorithms for representing, storing and searching words. We describe how a public domain word list was used in explaining key concepts, and in formulating exercises, assignments and term projects. For the result, we discuss representative projects submitted and some misconceptions that were identified when projects were tested and demonstrated in front of the class.

Key Words: Introductory computer science; Learning in context; Basic data structures and algorithms; Word games; Dictionary search

1. INTRODUCTION

An emerging practice in teaching an introductory computer science is to contextualize abstract concepts based on real world data and activities such as playing games or solving puzzles. To wit, recent papers from the ACM Special Interest Group on Computer Science Education (SIGCSE) annual conference include cases based on board game strategies (Bezakova, Heliotis, Strout, 2013), pencil puzzles (Butler, Bezakova, and Fluet, 2017), and "diverse real-world data sets" (Bart, Whitcomb, Kalura, Shaffer, and Tilevich, 2017). It is hypothesized from these representative works that a real-world context would keep the student grounded and engaged in the learning activities. There are some studies, however, such as (Bouvier, et al., 2016) that argue that providing context may add to cognitive load.

We explored word games as context for teaching a freshman programming subject. We think that word games are suitable based on the fact that students have had experience playing or solving word games on just plain paper such as a crossword puzzle, a board game such as Scrabble or Boggle, a computer program such as the traditional Hangman or TextTwist, or in more recent years, a mobile app such as 4 pics 1 word. Even if this was not the case, a portion of the class activities can be devoted to learning/playing and enjoying a word game or two. Our assumption is that the context will not negatively affect the cognitive load when there is prior experience; (Griggs and Cox, 1982, page 417) states that the "presentation of the task allows the subject to recall past experience with the content of the problem".

In the following sections, we describe how a public domain word list was used in explaining key concepts, and in formulating exercises, assignments and term projects. For the result, we discuss representative machine projects submitted by students, including some misconceptions that were

identified when the projects were tested and demonstrated in front of the class.

2. PROGRAMMING at DLSU CCS

COMPRO2 is the 2nd programming subject taken by freshman BS in Computer Science students at De La Salle University, College of Computer Studies. In a regular offering, it is usual to have eight to ten sections of COMPRO2, each having around 20 students. The big idea that we need to learn in COMPRO2 is how to represent, store and manipulate a group of elements in a computer. To this end, topics covered include (a) dynamic memory allocation, (b) arrays, (c) strings, (d) structures, (e) linked lists and (f) file processing. The C programming language is the implementation language used in writing the source codes.

Students are assessed based on their output: (a) exercises and assignments, (b) long exams and (c) machine project/problem (MP). The scope of the MP is larger, and requires more time and effort to accomplish compared with a simple programming exercise or assignment. The student would need to perform iterative activities such as think, design and implement the data structures and algorithms, code, test, debug, analyse and document the solution to a programming problem or task. Thus, the MP serves as a major evidence for an authentic assessment of the expected learning outcomes of the course. The students are given "enough time", typically, four weeks, to solve and turn in milestones of their MP.

Based on the COMPRO2 syllabus, we list below the specific learning outcomes (LOs) that were targeted using word games as context in the learning activities:

1. Analyze, design and implement algorithmic solutions for problems requiring the use of data structures, individually or in a team, using appropriate C constructs revolving around data representation and acquisition, data processing, and output representation.
2. Develop syntactically and semantically correct and modularized programming codes that exhibit code readability.
3. Communicate the rationale for employing the specific data structure and the logic behind the formulated solution to a programming problem.
4. Assess correctness of programming solution by identifying test cases to evaluate solution.

3. METHODOLOGY: WORD GAMES AS CONTEXT

We started exploring word games as context in COMPRO2 in Term 2, AY2016-2017. The problem specifications for several exercises, assignments, and MP were written using the Enhanced North American Benchmark Lexicon Millennial edition, abbreviated as ENABLE2K, as the input word list. ENABLE2K provides real-world data comprising of 173,528 English words mainly of American English spellings. It is in public domain and used by several word games available on the internet.

We describe below how word games were used as context in the learning activities in three steps.

3.1 First Step: Getting Acquainted

The data stored in ENABLE2K are words which can be represented and stored as 1D array of characters, abstracted as strings. In one of the laboratory activities, the students were initiated in the context by asking them to explore and experiment searching for words using some prescribed string patterns in `www.morewords.com` (Hoare, 2017).

It is during this step that the students were instructed to download a text file named "enable2k.txt" which contains the words in ENABLE2K from `www.morewords.com`. All words are encoded in lower case letters, and listed in alphabetical order. They were then asked to open, browse and study the contents of the file. To encourage critical thinking, initial questions were posed such as:

- Q1: What are the first and last words in the file?

- Q2: What is the shortest word? What is its length?
- Q3: What is the longest word? What is its length?
- Q4: How many 5-letter words are there?
- Q5: How many words start with 'q' (or another letter)?
- Q6: How many words end with 'q' (or another letter)?

Q1 can readily be answered by looking at the contents of "enable2k.txt" file (answer: "aa", "zyzzyvas"). Q3 to Q6, can also be answered by brute force manual inspection of the text file contents. The real intent behind these questions is actually for the students to realize, on their own, that answers to such kind of questions are better formulated computationally by applying concepts learned in the subject.

3.2 Second Step: Solving Small Problems

Having been acquainted with the word list (and hopefully with the students' interest piqued), the next learning activity proceeded to formulating the computational steps, and implementing programs that answer questions similar to those posed above. Q3, for example, is typical of a class of problems that finds a minimum or maximum value, in this case the longest word. Q4 requires counting, Q5 and Q6 involve searching for a character in a string.

To answer all the questions above, the program would need to input, i.e., read, the words from "enable2k.txt" file. It should be noted that during the time that this learning activity was conducted, the topic on file processing was not yet covered. The workaround to this problem is to run the executable program in the command line interface with input redirection, for example:

```
c:\> readwords < enable2k.txt
```

where `readwords.exe` is the executable file for the program that will read the words, and the < symbol represents a redirection which means that the input data will come "enable2k.txt" instead of a default keyboard input. A partial source code for the `readwords` program is shown in following:

```
#define NWORDS 173528
// ...some other lines of code...

   int i;
   char word[51];

   for (i = 0; i < NWORDS; i++){
      scanf("%s", word);
      printf("%s\n", word);
   }
```

The choice of 51 as the array size for the `word[]` string is actually arbitrary as we shall see later on. This code was then used by the students as the starting point for implementing their own respective programs as solutions for Q2 to Q6. For example, some student solutions that answer Q3, i.e., the longest word, were formulated as follows:

```
  int i;
   char word[51];
   char longest[51];

   strcpy(longest, "");
   for (i = 0; i < NWORDS; i++){
      scanf("%s", word);
      if (strlen(word) > strlen(longest))
         strcpy(longest, word);
   }

   printf("%s %d\n", longest,
          strlen(longest));
```

The longest word is "ethylenediaminetetraacetates" with a length of 28 characters. Having found the longest word necessitates a change in the declaration of the `word[]` variable to

```
 char word[29];
```

to reduce waste in memory space.

Programming problems were then assigned to answer other questions such as:

- Q7: How do you convert the words in lower case letters to upper case letters?
- Q8: How many palindromes are there?
- Q9: How many words contain "???" as substring? (where "???" are specified letters)

It was observed that the majority of the students were able to turn in correct solutions to the assignments. This simple activity helped assessed if the students learned arrays, strings and string related functions. Indirectly, a positive side effect is that the students also learned new words that add to their vocabulary.

Solving these small problems served as preparations for the students to gain the necessary understanding and confidence prior to solving a larger problem specified in the next step.

3.3 Final Step: Solving the MP

Students were asked to submit a one page proposal describing the word game that they would want to implement for their MP. This is in contrast to the traditional practice in the past where the instructor gives a MP specification and all students would solve the same problem. The change is quite important here, because the students now have some form of freedom to choose what they would want to work on, hopefully, something that is really interesting to them.

Essentially all the initial proposals were accepted and the students proceeded to designing and implementing their data structures and algorithms for their respective word games. The students were also made aware that they have the option to change the game that they would like to implement at any point in time before the actual MP final submission deadline. It should be noted here that GUI was not part of the MP requirements.

A starter kit was provided to the students so that they do not have to start from scratch. The starter kit includes the following:

- a project file
- file containing the main() function
- a header file containing the #defines and function prototypes related to initializing, printing and freeing up the memory for the word list
- an object file that contains the binary implementation of the functions declared in the given header file

The use of a project file allowed the students to experience programming where functions are grouped into modules and stored in multiple source code files (see LO2 in section 2).

When the MP specifications were given, the students have yet to learn file processing which is the last topic required in the syllabus. It should also be noted that in a word game, the player would need to input words via keyboard. Thus, the input redirection technique described in the previous subsection can no longer be used. As a workaround, an object file was provided as part of the starter kit which implements the functionality for reading the words from "enable2k.txt" using text file processing functions, specifically, `fopen()`, `fscanf()` and `fclose()`. This allows the students to progress with their MP without being hampered by lack of necessary file processing know-how. Once the students get to learn text file processing, they were required to remove the object file from the project file, and replace it with their own source code implementation as part of the final MP deliverables.

The MP specifications pointed out that the use of a naïve simple linear search on the entire ENABLE2K word list is discouraged. As case in point, assume that the search key is "zyzzyvas". A naïve linear search would require that the key be compared with all the words in the dictionary, i.e., there will be 173,528 comparisons (which is the total number of words in "enable2k.txt")! The actual objective of the MP was for the students to think of and come up with a method, i.e., data structure and algorithm, for word search which is faster than a naïve linear search (see LO1 in Section 2).

Two milestones were set both in the form of a software demonstration. The first demonstration was not graded since the intents are: to find out how the student progressed, provide a venue for student consultation to address problems or issues encountered, and provide feedback on the current

status of the project. The second demonstration was one graded. It also required the students to present in front of the class with selected peers as testing participants (beta testers).

The final MP deliverables are: the project file with the corresponding source codes, and a document that describes the word game implemented, the data structure for representing the words, and the word search algorithm (see LO3 in Section 2).

4. RESULTS AND OBSERVATIONS

A total of 26 projects were submitted coming from three sections (about 9 projects per section). Since GUI is not required, the user-interface is primarily keyboard and text based. There were 8 groups who submitted MPs based on TextTwist, 5 groups implemented Shiritori, 3 groups on guessing game (given the letters, guess the word), 1 group on Hangman, 1 group on memory game (show some words, the players memorizes them, screen clears, the user types in the words he remembers), and other groups submitted some other games that belong to a different category. One group submitted an interesting game that was inspired by Formula 1 racing. The idea is for the user to input a word, and based on the word length advances the virtual car by a certain number of laps, i.e., longer words would move the car in more laps thus reaching the final lap faster. There were also some groups that implemented functionalities for storing and reading leaderboard data using text file processing.

The MPs were demonstrated by the students in front of the class. Their peers served as testers. After testing the MP, the students were asked to explain to the class the data structure they used for representing and storing the words and the algorithm they used for searching.

We describe some of the ideas that came up in the submitted MPs. Some simply used binary search instead of linear search. While it was acceptable as a MP solution, the students were made aware that it was "trivial" solution.

One way to speed up the search is to start the linear search not with "aa" (the 1st word in the dictionary), but with the 1st word that starts with the same letter as the search key. For example, if the key is "zyzzyvas", then linear search will start comparison with "zabaglione" which is the first word that starts with letter 'z'. This radically reduces the number of comparisons from 173,528 to 543 (where 543 is the number of words that starts with 'z'). The supporting data structure used in this approach is a 1D array of integers such that each array element contains the index where the 1st word of a given letter can be found.

An alternative data structure for representing words used an array of linked lists. Since there are 26 letters in the English alphabet, there will be 26 linked lists. That is, the first array element contains a pointer to the linked list of words starting with 'a', and the last array element contains the pointer to the linked list of words starting with 'z'. Searching for a word then proceeded using a linear search on the associated linked list.

Another idea grouped the words based on their string lengths. For example, all words with a length of 2 will be in the 1st group, those with a length of 3 will be in the 2nd group, those with a length of 4 will be in the 3rd group, and so on. The words in each group were stored in alphabetical order. Using the example above, since the length of the search key "zyzzyvas" is 8, it is only logical to search only in the group of words with a length of 8. This idea was used by some of the MPs that implemented TextTwist.

Some MPs required the word game such as Shiritori to check and disallow words that have already been entered before by the player. For example, assume that the current Shiritori word sequence is "jargon", "notes", "song", "games". If the player inputs "song" then it will be rendered as an invalid input since it was already entered before. This game rule necessitates that the MP be implemented with another data structure for checking repetitive input words.

On another note, the MP demo also served as a learning opportunity to point out improvements, and for correcting misconceptions committed by students. The following were observed during the demonstration:

- Some MPs do not test if `malloc()` returns a NULL value thus exposing the program to a possible run-time error when memory is no longer available.
- Some MPs do not `free()` up the dynamically allocated memory.
- As described above, an integer array, say A[] of 26 values can be used to store the indices of the first word that starts with a given letter of the alphabet. In at least two MPs, a binary search was used with low and high limits set based on the starting letter of the search key. For example, if the word is "bugle", the low index is A[1] and the high index is A[2] – 1. The idea is correct per se', but the misconception is that the index value was determined using a linear search instead of taking advantage of the mapping of the 1st character via simple assignments, i.e.,

```
index = word[0] – 'a';
low = A[index];
high = A[index + 1] – 1;
```

- One MP implemented binary search correctly. However, the low and high limits were obtained by incorrectly doing two separate linear searches on the entire word list to find the first and last words that starts with the same letter as the search key.

5. CONCLUSION AND FUTURE WORK

Based on the initial explorations, we think that word games can serve as useful and realistic context for learning and applying the abstract concepts in programming. All the topics that the students learned in COMPRO2 (i.e., dynamic memory allocation, arrays, strings, structures, linked list and file processing) can be applied and the learning outcomes can be assessed based on the deliverables.

We still need to investigate and verify via a qualitative survey if the chosen context was really able to engage the student and if it was able to maintain the student's interest in learning the subject. We also would like to check if word games serve as better context compared with the contexts used in previous offerings.

6. REFERENCES

Bart, A. C., Whitcomb, R., Kafura, D., Shaffer, C. and Tilevich, E. (2017). Computing with CORGIS: diverse, real-world data sets for introductory computing. In Proceedings of the ACM SIGCSE '17, pages 57-62.

Bezakova, I., Heliotis, J., & Strout, S. (2013). Board game strategy in introductory computer science. In Proceedings of the ACM SIGCSE '13, pages 17-22.

Bouvier, D., Lovellette, E., Matta, J., Alshaigy, B., Becker, B., Craig, M., Jackova, J., McCartney, R., Sanders, K., Zarb, M. (2016). Novice programmers and the problem description effect. In Proceedings of the ACM SIGCSE '16, pages 103-118.

Butler, Z., Bezakova, I., & Fluet, K. (2017). Pencil puzzles for introductory computer science: an experience and gender-neutral context, In Proceedings of the ACM SIGCSE'17, pages 93-98.

Griggs, R. and Cox, J. (1982). The elusive thematic-materials effect in Wason's selection task. British Journal of Psychology, 73 (3): pages 407-420.

Hoare, R. (2017). Morewords.com, accessed, April 18, 2017.