WORD SENSE DISAMBIGUATION USING HUMAN COMPUTATION

Project Report submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology.

In

Computer Science & Engineering

By

PRANAV JAIN (121222)

Under the Supervision of

MR. ARVIND KUMAR

To

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Word Sense Disambiguation Using Human Computation"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2015 to May 2016 under the supervision of Mr. Arvind Kumar (Assistant Professor, Dept. of CSE & IT).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Pranav Jain
121222

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Date: 30-05-2016

ARVIND KUMAR
Assistant Professor
Dept. of CSE & IT

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my guide Mr. Arvind Kumar for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by his time to time shall carry me a long way in the journey of life on which I am about to embark.

The in-time facilities provided by the Department of Computer Science & Engineering and Information Technology throughout the project development are also equally acknowledgeable.

At the end I would like to express my sincere thanks to all my friends and everyone else who helped me directly or indirectly during this project work.

Pranav Jain
121222

Table of Contents

List of Abbreviations

WSD – Word Sense Disambiguation

HCOMP – Human Computation

NLP – Natural Language Processing

CV – Computer Vision

HBGA – Human Based Genetic Algorithm

List of Figures

List of Tables

Abstract

Words have different meanings based on the context of the word usage in a sentence. Word sense is one of the meanings of a word. Human language is ambiguous, so that many words can be interpreted in multiple ways depending on the context in which they occur.
Word sense disambiguation (WSD) is the ability to identify the meaning of words in context in a computational manner. WSD is considered an AI complete problem, that is, a task whose solution is at least as hard as the most difficult problems in artificial intelligence.

WSD can be viewed as a classification task: word senses are the classes, and an automatic classification method is used to assign each occurrence of a word to one or more classes based on the evidence from the context and from external knowledge sources. WSD heavily relies on knowledge. Knowledge sources provide data which are essential to associate senses with words. The assessment of WSD systems is discussed in the context of the Senseval/Semeval campaigns, aiming at the objective evaluation of systems participating in several different disambiguation tasks. Here, some of the knowledge sources used in WSD, different approaches for WSD (supervised, unsupervised and Knowledge-based) and evaluation of WSD systems are discussed. The applications of WSD are also seen.

In our project, we shall use Human Computation (HCOMP) to achieve the desired disambiguation for a word. Using HCOMP will decrease the processing overhead in our project as compared to that of WSD using AI algorithms. Also, the HCOMP component of our project will be achieved by a fun multiplayer game which will urge players to choose alternate words for certain words in a phrase or a sentence.

CHAPTER 1: INTRODUCTION

1.1 Word Sense Disambiguation

In computational linguistics, word sense disambiguation, more commonly known by its abbreviation WSD, is an open problem of natural language processing and ontology [1].

One of the first problems that any natural language processing (NLP) system encounters is lexical ambiguity, syntactic or semantic. The resolution of a word's syntactic ambiguity has been solved in language processing by part-of-speech taggers with high levels of accuracy. The problem of resolving semantic ambiguity is generally known as word sense disambiguation (WSD) and has been proved to be more difficult than syntactic disambiguation. Human language is ambiguous, so that many words can be interpreted in multiple ways depending on the context in which they occur the identification of the specific meaning that a word assumes in context is only apparently simple. Unfortunately, the identification of the specific meaning that a word assumes in context is only apparently simple.

Consider the following two sentences,

(a) I can hear bass sounds.

(b) They like grilled bass.

The occurrences of the word bass in the two sentences clearly denote different meanings: low frequency tones and a type of fish, respectively. Here, the process WSD assigns correct meaning to the word bass in the above two sentences as

(a) I can hear bass / low frequency tone sounds.

(b) They like grilled bass / fish.

WSD is one of the central challenges in Natural Language Processing (NLP). Many tasks in NLP require disambiguation. Word Sense Disambiguation is needed in Machine Translation, Information Retrieval, Information Extraction etc. WSD is typically configured as an intermediate task, either as a stand-alone module or properly integrated into an application.

WSD can be easily understood via the following figure:

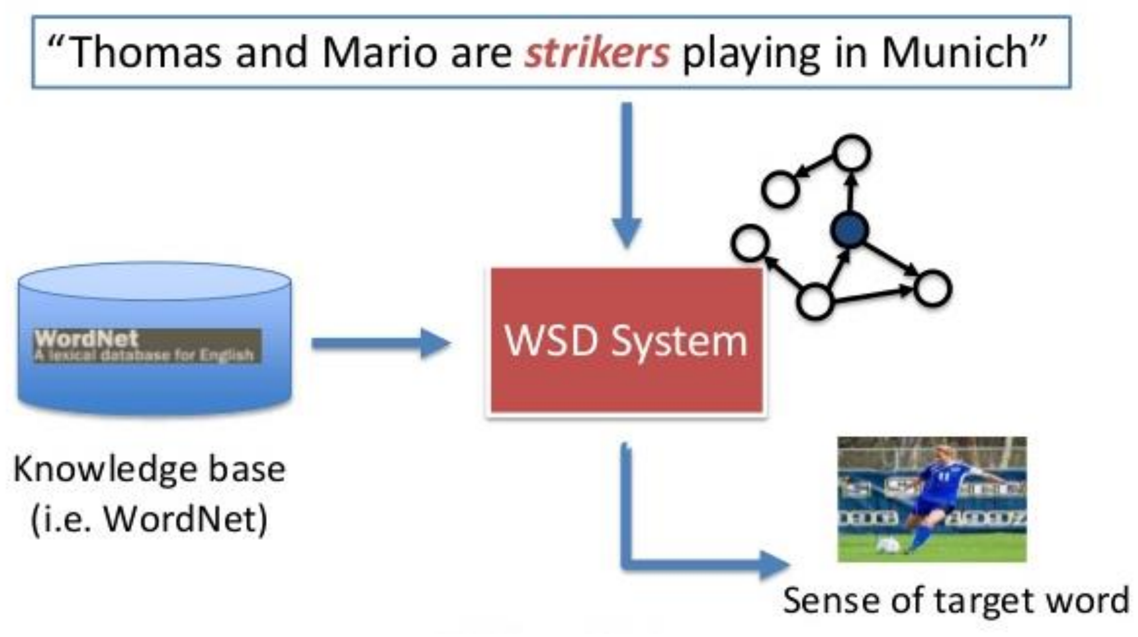

"Thomas and Mario are *strikers* playing in Munich"
WordNet
A lexical database for English
Knowledge base
(i.e. WordNet)
WSD System
Sense of target word

Fig. 1.1.1: WSD in a Nutshell

1.2 Human Computation

Human Computation (HCOMP) is a technique in which a computer performs its function by outsourcing certain steps to humans, usually to enhance a program's heuristics [2].

This approach uses differences in abilities and alternative costs between humans and computers to achieve symbiotic human-computer interaction. HCOMP reverses the roles of a human and a computer to some extent as the computer asks a person to solve a problem, and then collects and interprets their solutions. Manual creation of data set for WSD is an arduous and monotonous job. And also it will be difficult for a small number of programmers to create a strong and reliable data set.

The following figure explains the overwhelming advantage that the human brain has over its digital counterpart:

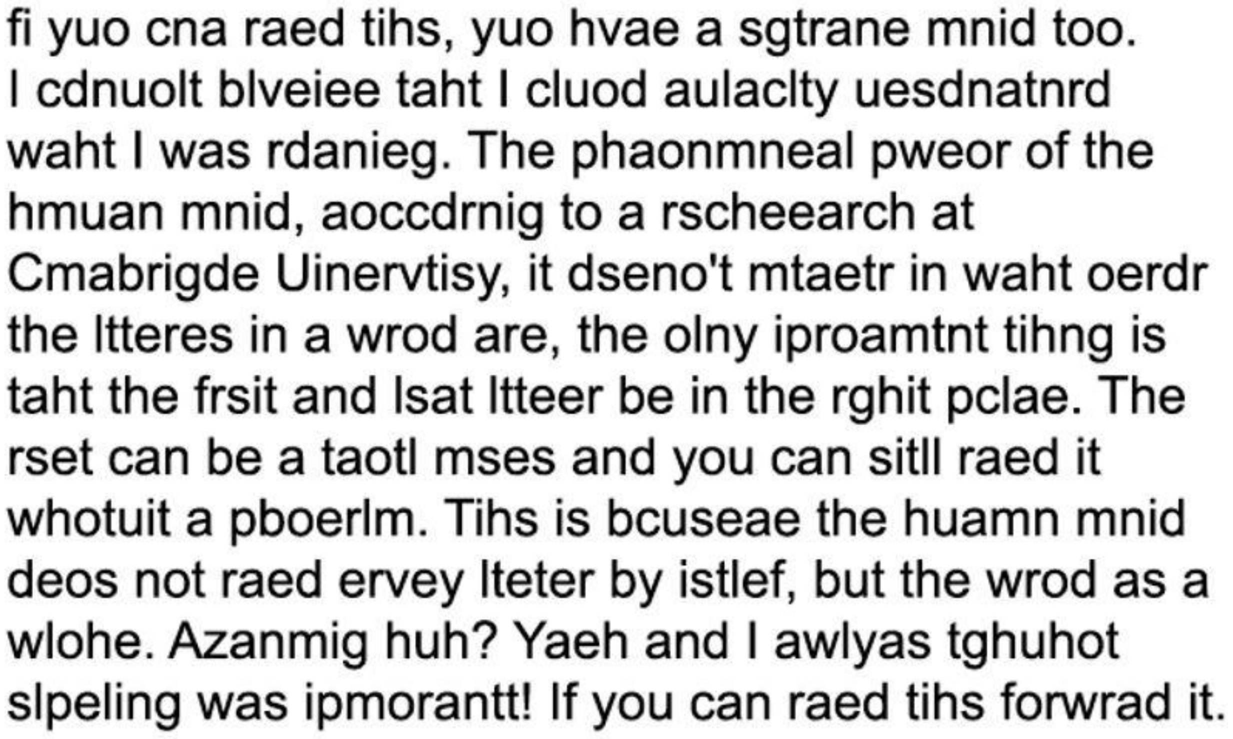

fi yuo cna raed tihs, yuo hvae a sgtrane mnid too. I cdnuolt blveiee taht I cluod aulaclty uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuan mnid, aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it dseno't mtaetr in waht oerdr the ltteres in a wrod are, the olny iproamtnt tihng is taht the frsit and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it whotuit a pboerlm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Azanmig huh? Yaeh and I awlyas tghuhot slpeling was ipmorantt! If you can raed tihs forwrad it.

Fig 1.2.1: Text Readable By Humans But Not By Computers

A computer in no way imaginable can read and understand the paragraph, but a human brain can, and the reasons are still debatable. This explains the clear advantage that the human brain has over a computer – a computer has to be programmed and trained for it to be able to understand the human language, but our brains are automatically programmed to do so.

Therefore to make this process easy and fun, HCOMP is the best way to create the data set for WSD.

There are four types of HCOMP systems:

- Voluntary
- Incentive By Money
- Incentive By Fun
- Mandatory

We have chosen to go ahead with Incentive by Fun HCOMP for our project because it serves our purpose in the best possible manner out of the four HCOMP options available.

1.3 Problem Description

In this project, we are going to tackle the problem faced in creation of datasets used for WSD algorithms, which is also known as corpora acquisition bottleneck. Manual creation of data set for WSD is an arduous and monotonous job since it is very difficult for a small number of programmers to create a strong and reliable data set manually or through NLP programs.

Therefore to make this process easy, efficient and fun, human computation is the best way to create the dataset for WSD.

1.4 Objective and Methodology

Our main objective is to create a reliable dataset for feeding various WSD algorithms via human computation through a fun game designed to produce valuable output by engaging human players in what they perceive to be a cooperative task of guessing the same word as another player.

We are going to create a two player game in which players will be given a sentence or paragraph and a highlighted word within that context. The players will be given a certain amount of time to provide as many alternate suggestions as they can for the highlighted word within the given context. The main aim of the players would be to provide alternate suggestions such that they match those provided by their opponent, which will earn them points. Therefore, the players will involuntarily provide us with a dataset just by playing a fun word guessing game.

CHAPTER 2: LITERATURE REVIEW

2.1 Title: Word Sense Disambiguation via Human Computation [3]

One formidable problem in language technology is the word sense disambiguation (WSD) problem: disambiguating the true sense of a word as it occurs in a sentence (e.g., recognizing whether the word "bank" refers to a river bank or to a financial institution). This paper explores a strategy for harnessing the linguistic abilities of human beings to develop datasets that can be used to train machine learning algorithms for WSD. To create such datasets, we introduce a new interactive system: a fun game designed to produce valuable output by engaging human players in what they perceive to be a cooperative task of guessing the same word as another player. Our system makes a valuable contribution by tackling the knowledge acquisition bottleneck [8] in the WSD problem domain. Rather than using conventional and costly techniques of paying lexicographers to generate training data for machine learning algorithms, we delegate the work to people who are looking to be entertained.

The relevance of WSD is becoming clear as advancing information/web technologies are catalysts for the production of enormous amounts of textual data, including articles, blogs, status messages, digitized books, etc. There is a growing need to introduce structure to this data in order to make it consumable and manageable by machines. Current WSD algorithms use collections of data and machine learning algorithms to create models that determine the sense of the target word in the sentence. Generally supervised algorithms perform better than unsupervised algorithms. These facts made human computation an ideal technique for this problem – with sufficient knowledge supervised algorithms can be used for almost all applications. Currently knowledge acquisition for WSD is very expensive. Manual creation of a training dataset for a WSD system involves taking a large set of textual data, isolating words to disambiguate, and hand labelling each of these words with their gold label word sense. This process is an arduous and consequently an expensive one. But what if we make this labelling process a pleasant one? This paper explores a new system: a game that is designed to capture human knowledge in a distributed fashion via an enjoyable game. Our study involves assessing the effectiveness of this game in tackling the knowledge acquisition bottleneck [8].

In this paper we described a game, named Jinx, which is designed to generate word sense disambiguation (WSD) datasets. These datasets can be used to train high quality machine learning algorithms for the WSD problem. The game accomplishes this task in a low cost, distributed, fashion by employing human beings to consider a word in sentence and generate guesses for synonyms for the word in the context of the sentence. The game uses a point system to provide utility to users and uses a cooperative, paired player, structure to make the game fun and to control the quality of the guesses. We populated the game with ten words and multiple sentences from a widely recognized word sense evaluation dataset. We then had people play the game for approximately an hour. In postgame interviews, most everyone reported that the game was fun.

Thus we are confident that the game is capable of attracting a large, sustainable, audience. Preliminary analysis of the guesses of the game indicates that many of the guesses correspond directly to synonym sets for words in that context. Thus the game generates a set of synonyms to a particular word in a sentence as perceived by the general public. This dataset in itself contains interesting linguistic data. With respect to WSD, however, many guesses corresponded to more than one word sense or corresponded to incorrect word senses. We are currently exploring more sophisticated data analysis methods to extract a high quality WSD dataset.

2.2 Title: Word Sense Disambiguation Corpora Acquisition via Confirmation Code [4]

Word Sense Disambiguation (WSD) is one of the fundamental natural language processing tasks. However, lack of training corpora is a bottleneck to construct a high accurate all-words WSD system. Annotating a large-scale corpus by experts costs enormous time and financial resources. Human Computation is a novel idea for integrating human resources behind the Web, which has been wasted, to solve practical problems that are difficult for computers. Based on human computation, we design a confirmation code system, which can not only distinguish between human beings and computers (the function of normal confirmation code system), but also annotate WSD corpora. The preliminary experimental result shows that the proposed method can annotate large-scale and high-quality WSD corpora within a short time. To the best of our knowledge, this is the first attempt to use confirmation code in natural language processing for corpora acquisition.

A WSD confirmation code includes two questions. Each question consists of a sentence and a highlighted ambiguous word in the sentence. All senses of the ambiguous word are provided as optional answers. The system only knows the answer for one of the two questions, which is named as known question and the other is unknown question. A user needs to choose a word sense for each ambiguous word. The user can pass the confirmation stage if and only if his answer to the known question is correct. Like in reCAPTCHA, users do not know which one is known question. They must choose each word sense carefully in order to pass confirmation stage. Therefore, they provide the correct sense for the ambiguous word of unknown question. If WSD confirmation code system is widely used by lots of Web sites, we can easily collect large-scale corpora.

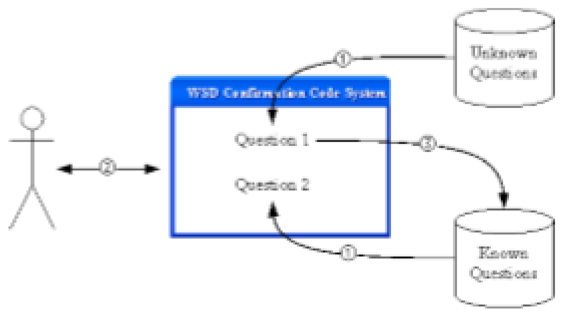


Fig. 2.2.1: The data flow chart of WSD confirmation code system

The data flow chart of the WSD confirmation code system is shown in Figure 2.2.1. Two questions are randomly selected from known and unknown question databases respectively - the two questions are asked to a user and the user needs to answer them. Once the user's answer is correct, i.e. it is equal to the answer of known question, he can pass the confirmation stage. Then we can know the answer of the unknown question, which becomes a new known question and can be added into the known question database. Otherwise, the confirmation stage cannot be passed and the user has to answer another pair of questions.

To address the lack of WSD corpora, we propose a human computation based method. When users successfully input a confirmation code, they annotate a WSD example incidentally. The preliminary experiments show that the novel method can annotate large-scale and high-quality WSD corpora within a short time. As far as we know, there is no work done to annotate natural language processing corpora with confirmation code. In the future, we plan to improve the annotation speed and reduce the complexity of confirmation process by showing two sentences with the same ambiguous words. Thus, users can easily compare the two sentences. More importantly, they only need to read the options once, which can save confirmation time further. We also can use unsupervised clustering method which determines senses that are very similar and displays only one of the alternatives. Secondly, we will apply this method to other languages. Our method is general enough and can be applied to any languages as long as the language has a thesaurus and some initial WSD corpora. Of course, a particular language WSD confirmation code system can only be used in Web sites of the same language because it is difficult for a normal user to perform WSD task on the foreign language that they are unfamiliar with. Thirdly, we can apply this method to other natural language processing tasks which need corpora acquisition such as co-reference resolution, named entity recognition, and parsing. Finally, we will use the corpora annotated by the confirmation code method to train a more effective WSD model.

2.3 Title: Determining the Difficulty of Word Sense Disambiguation [5]

Automatic processing of biomedical documents is made difficult by the fact that many of the terms they contain are ambiguous. Word Sense Disambiguation (WSD) systems attempt to resolve these ambiguities and identify the correct meaning. However, the published literature on WSD systems for biomedical documents report considerable differences in performance for different terms. The development of WSD systems is often expensive with respect to acquiring the necessary training data. It would therefore be useful to be able to predict in advance which terms WSD systems are likely to perform well or badly on.

This paper explores various methods for estimating the performance of WSD systems on a wide range of ambiguous biomedical terms (including ambiguous words/phrases and abbreviations). The methods include both supervised and unsupervised approaches. The supervised approaches make use of information from labelled training data while the unsupervised ones rely on the UMLS Metathesaurus. The approaches are evaluated by comparing their predictions about how difficult disambiguation will be for ambiguous terms against the output of two WSD systems. We find the supervised methods are the best predictors of WSD difficulty, but are limited by their dependence on labelled training data. The unsupervised methods all perform well in some situations and can be applied more widely.

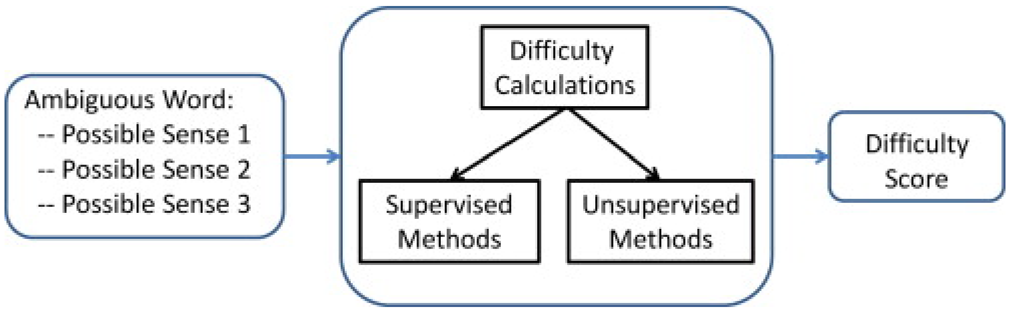


Fig. 2.3.1: WSD Difficulty Flowchart

Word Sense Disambiguation (WSD) is the task of automatically identifying the appropriate sense of an ambiguous word based on the context in which the word is used. For example, the term cold could refer to the temperature or the common cold, depending on how the word is used in the sentence. Automatically identifying the intended sense of ambiguous words improves the performance of biomedical and clinical applications such as medical coding and indexing; applications that are becoming essential tasks due to the growing amount of information available to researchers. This paper explores approaches to estimating the difficulty of performing WSD on ambiguities found in biomedical documents. By difficulty we mean the WSD performance that can be obtained for the ambiguity since, in practise, performance is the most important factor in determining whether applying WSD to a particular ambiguity is likely to be useful. Ambiguities for which low WSD performance is obtained are considered to be difficult to disambiguate while those for which the performance is high are considered to be easy to disambiguate.

A wide range of approaches have been applied to the problem of WSD in biomedical and clinical documents. Accurate WSD can improve the performance of biomedical text processing applications, such as summarization, but inaccurate WSD has been shown to reduce an application's overall performance. The disambiguation of individual terms is important since some of those terms are more important than others when determining whether there is any overall improvement of the system. The importance of WSD is likely to depend on the application and research question. For example, Weeber et al. found that it was necessary to resolve the ambiguity in the abbreviation “MG” (which can mean “magnesium” or “milligram”) in order to replicate the connection between migraine and magnesium identified by Swanson.

Some of the methods applied in this paper are supervised since they are based on information derived from a corpus containing examples of the ambiguous term labelled with the correct sense. Other methods do not require this resource and only require information about the number of possible senses for each ambiguous term which is normally obtained from a knowledge source, such as the UMLS Metathesaurus.

The accuracy of WSD systems for biomedical documents varies enormously across ambiguous terms. It would be useful to be able to predict the difficulty of a particular term for WSD systems in order to determine whether applying WSD would be useful. In this paper, we explore a range of approaches to estimating WSD difficulty. Some of these are based on information extracted from sense-labelled corpora while others make use of information from knowledge sources. Evaluation was carried out by comparing the predictions made by these measures with the actual accuracy of two different WSD systems on three data sets.

Results show that the supervised methods are good predictors of WSD difficulty in some cases, but that their results are not consistent across different data sets. These methods also require labelled training data, limiting their usefulness. The unsupervised approaches do not have this limitation and can be applied to a wider range of ambiguities. Our experiments showed that these approaches were also good predictors of WSD difficulty. The best performance was obtained using the relatedness measure proposed by Lesk and aggregating the scores using the mean similarity metric. This method obtained a statistically significantly higher negative correlation than the other measures when compared to both the supervised and unsupervised

WSD systems. The performance of this measure was also reasonably consistent across different data sets and types of ambiguity (terms and abbreviations). The methods explored in this paper are useful tools for estimating the performance of a WSD system that can be computed without the need for labelled data.

In the future, we plan to explore other relatedness measures that use contextual information about the senses rather than (or in conjunction with) their placement within a taxonomy. We would also like to explore semantic groups of the terms to determine if some types are easier to disambiguate than others.

2.4 Early Work in Human Computation [9]

Human-based computation (apart from the historical meaning of "computer") research has its origins in the early work on interactive evolutionary computation. The idea behind interactive evolutionary algorithms is due to Richard Dawkins. In the Biomorphs software accompanying his book The Blind Watchmaker (Dawkins, 1986) the preference of a human experimenter is used to guide the evolution of two-dimensional sets of line segments. In essence, this program asks a human to be the fitness function of an evolutionary algorithm, so that the algorithm can use human visual perception and aesthetic judgment to do something that a normal evolutionary algorithm cannot do. However, it is difficult to get enough evaluations from a single human if we want to evolve more complex shapes. Victor Johnston and Karl Sims extended this concept by harnessing the power of many people for fitness evaluation (Caldwell and Johnston, 1991; Sims, 1991). As a result, their programs could evolve beautiful faces and pieces of art appealing to public. These programs effectively reversed the common interaction between computers and humans. In these programs, the computer is no longer an agent of its user, but instead, a coordinator aggregating efforts of many human evaluators. These and other similar research efforts became the topic of research in aesthetic selection or interactive evolutionary computation (Takagi, 2001), however the scope of this research was limited to outsourcing evaluation and, as a result, and it was not fully exploring the full potential of the outsourcing.

A concept of the automatic Turing test pioneered by Moni Naor (1996) is another precursor of human-based computation. In Naor's test, the machine can control the access of humans and computers to a service by challenging them with a natural language processing (NLP) or computer vision (CV) problem to identify humans among them. The set of problems is chosen in a way that they have no algorithmic solution that is both effective and efficient at the moment. If it existed, such an algorithm could be easily performed by a computer, thus defeating the test. In fact, Moni Naor was modest by calling this an automated Turing test. The Imitation Game described by Alan Turing (1950) didn't propose using CV problems. It was only proposing a specific NLP task, while the Naor test identifies and explores a large class of problems, not necessarily from the domain of NLP that could be used for the same purpose in both automated and non-automated versions of the test.

Finally, Human-based genetic algorithm (HBGA) encourages human participation in multiple different roles. Humans are not limited to the role of evaluator or some other predefined role, but can choose to perform a more diverse set of tasks. In particular, they can contribute their innovative solutions into the evolutionary process, make incremental changes to existing solutions, and perform intelligent recombination. In short, HBGA allows humans to participate in all operations of a typical genetic algorithm. As a result of this, HBGA can process solutions for which there are no computational innovation operators available, for example, natural languages. Thus, HBGA obviated the need for a fixed representational scheme that was a limiting factor of both standard and interactive EC. These algorithms can also be viewed as novel forms of social organization coordinated by a computer.

2.5 Luis von Ahn [10]

Fig. 2.5.1 Luis Von Ahn

Luis von Ahn (born 1979) is a Guatemalan entrepreneur and an associate professor in the Computer Science Department at Carnegie Mellon University. He is known as one of the pioneers of crowd sourcing. He is the founder of the company reCAPTCHA, which was sold to Google in 2009, and the co-founder and CEO of Duolingo, a popular language learning platform. As a professor, his research includes CAPTCHAs and human computation, which has earned him international recognition and numerous honours. Von Ahn's early research was in the field of cryptography. With Nicholas J. Hopper and John Langford, he was the first to provide rigorous definitions of steganography and to prove that private-key steganography is possible. In 2000, he did early pioneering work with Manuel Blum on CAPTCHAs, computer generated tests that humans are routinely able to pass but that computers have not yet mastered. These devices are used by web sites to prevent automated programs, or bots, from perpetrating large-scale abuse, such as automatically registering for large numbers of accounts or purchasing huge number of tickets for resale by scalpers.

Von Ahn's Ph.D. thesis, completed in 2005, was the first publication to use the term "human computation" that he had coined for methods that combine human brainpower with computers to solve problems that neither could solve alone. Von Ahn's Ph.D. thesis is also the first work on Games with a Purpose, or GWAPs, which are games played by humans that produce useful computation as a side-effect. The most famous example is the ESP Game, an online game in which two randomly paired people are simultaneously shown the same picture, with no way to communicate. Each then lists a number of words or phrases that describe the picture within a time limit, and are rewarded with points for a match. This match turns out to be an accurate description of the picture, and can be successfully used in a database for more accurate image search technology. The ESP Game was licensed by Google in the form of the Google Image Labeller, and is used to improve the accuracy of the Google Image Search. Von Ahn's games brought him further coverage in the mainstream media. His thesis won the Best Doctoral Dissertation 2010.

In 2007, von Ahn invented reCAPTCHA, a new form of CAPTCHA that also helps digitize books. In reCAPTCHA, the images of words displayed to the user come directly from old books that are being digitized; they are words that optical character recognition could not identify and are sent to people throughout the web to be identified. ReCAPTCHA is currently in use by over 100,000 websites and is transcribing over 40 million words per day. As of 2014, von Ahn is working on Duolingo, a company that aims to coordinate millions of people to translate the Web into every major language.

2.6 Title: Human Computation by Luis von Ahn [6]

Tasks like image recognition are trivial for humans, but continue to challenge even the most sophisticated computer programs. This thesis introduces a paradigm for utilizing human processing power to solve problems that computers cannot yet solve. Traditional approaches to solving such problems focus on improving software. I advocate a novel approach: constructively channel human brainpower using computer games. For example, the ESP Game, introduced in this thesis, is an enjoyable online game —many people play over 40 hours a week — and when people play, they help label images on the Web with descriptive keywords. These keywords can be used to significantly improve the accuracy of image search.

People play the game not because they want to help, but because they enjoy it. I introduce three other examples of "games with a purpose": Peekaboom, which helps determine the location of objects in images, Phetch, which collects paragraph descriptions of arbitrary images to help accessibility of the Web, and Verbosity, which collects "common-sense" knowledge. In addition, I introduce CAPTCHAs, automated tests that humans can pass but computer programs cannot. CAPTCHAs take advantage of human processing power in order to differentiate humans from computers, an ability that has important applications in practice. The results of this thesis are currently in use by hundreds of Web sites and companies around the world, and some of the games presented here have been played by over 100,000 people. Practical applications of this work include improvements in problems such as: image search, adult-content filtering, spam, common sense reasoning, computer vision, accessibility, and security in general.

Construction of the Empire State Building: 7 million human hours. The Panama Canal: 20 million human hours. Estimated number of human-hours spent playing solitaire around the world in one year: billions. A problem with today's computer society? No, an opportunity. What if this time and energy could be channelled into useful work? This thesis presents a general paradigm for doing exactly that: utilizing human processing power. We focus on harnessing human time and energy for addressing problems that computers cannot yet tackle on their own. Although computers have advanced significantly in many respects over the last 50 years, they still do not possess the basic conceptual intelligence or perceptual capabilities that most humans take for granted. By leveraging human skills and abilities in a novel way, we hope to solve large-scale computational problems that computers cannot yet solve and begin to teach computers many of these human talents. In this paradigm, we treat human brains as processors in a distributed system, each performing a small part of a massive computation. Unlike computer processors, however, humans require an incentive in order to become part of a collective computation. We propose online games as a means to encourage participation in the process.

We argue that games constitute a general mechanism for using brain power to solve open computational problems. Each problem requires the careful design of a game developed to be enjoyable and, at the same time, guarantee that game-play correctly solves instances of the problem. We argue that designing such games is much like designing computer algorithms: the game needs to be proven correct and enjoyable; its efficiency can be analysed; more efficient games can supersede less efficient ones, etc. Instead of using a silicon processor, these "algorithms" run on a processor consisting of ordinary humans interacting with computers over the Internet. We refer to these games as "human algorithm games."

We have presented a general paradigm for harnessing human computation power to solve problems that computers cannot yet solve. Some open problems that could be solved using this technique include:

• Language Translation. Imagine a game in which two players that do not speak the same language work together to translate text from one language to the other.

• Monitoring of Security Cameras. With cameras becoming less expensive over time, it is now feasible to have security cameras everywhere. Imagine a game in which people watch the security cameras and alert authorities of illegal activity.

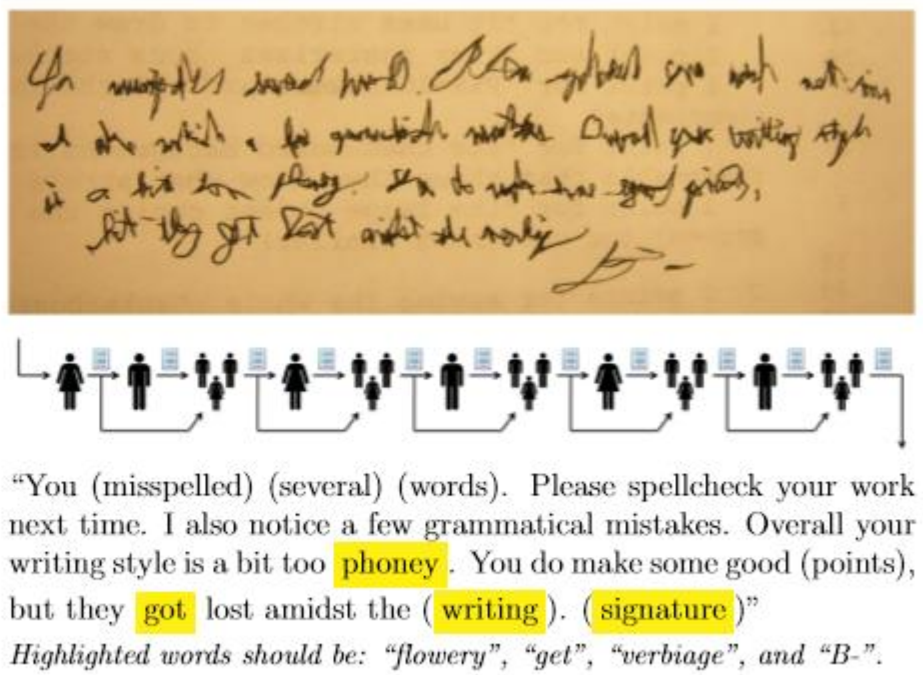
• Improving Web Search. Different people have different levels of skill at searching for information on the Web. Imagine a game in which the players perform searches for other people.

• Text Summarization. Imagine a game in which people summarize important documents for the rest of the world. (The biggest challenge in solving this problem is that it would require an intelligent way to break it up into small “bite-size” chunks.)

We believe there is an immense amount of work to be done in the continued development of this paradigm. Indeed, we hope researchers will use and improve upon the method and metrics presented for development and evaluation of human algorithm games. We believe the techniques in this thesis present an opportunity for researchers and game designers to contribute to the progress of Artificial Intelligence.

2.7 Title: Programming With Human Computation [7]

Amazon’s Mechanical Turk provides a programmatically accessible micro-task market, allowing a program to hire human workers. This has opened the door to a rich field of research in human computation where programs orchestrate the efforts of humans to help solve problems. This thesis explores challenges that programmers face in this space: both technical challenges like managing high-latency, as well as psychological challenges like designing effective interfaces for human workers. We offer tools and experiments to overcome these challenges in an effort to help future researchers better understand and harness the power of human computation. The main tool this thesis offers is the crash-and-rerun programming model for managing high-latency tasks on MTurk, along with the TurKit toolkit which implements crash-and-rerun. TurKit provides a straightforward imperative programming environment where MTurk is abstracted as a function call. Based on our experience using TurKit, we propose a simple model of human computation algorithms involving creation and decision tasks. These tasks suggest two natural workflows: iterative and parallel, where iterative tasks build on each other and parallel tasks do not. We run a series of experiments comparing the merits of each workflow, where iteration appears to increase quality, but has limitations like reducing the variety of responses and getting stuck in local maxima. Next we build a larger system composed of several iterative and parallel workflows to solve a real world problem that of transcribing medical forms, and report our experience. The thesis ends with a discussion of the current state-of-the-art of human computation, and suggests directions for future work.



"You (misspelled) (several) (words). Please spellcheck your work next time. I also notice a few grammatical mistakes. Overall your writing style is a bit too phoney. You do make some good (points), but they got lost amidst the (writing). (signature)"

Highlighted words should be: "flowery", "get", "verbiage", and "B-".

Fig. 2.7.1: A passage of poor handwriting is transcribed using a human computation algorithm. Workers put parenthesis around words they are unsure about.

This thesis is about programming with human computation. The sequence in Figure 2.7.1 is an example. The example starts with a passage of practically indecipherable handwriting. This passage is fed into a program to decipher the text. This program is special — it can hire people. That is, it uses human computation. The program hires a series of workers in a human computation market, namely Amazon's Mechanical Turk, to make guesses about the words in the passage. The program hires additional workers to vote on the guesses made by other workers. This process, an automated collaboration between human and machine, correctly deciphers 90% of the words. The human computation algorithm above is just one possible algorithm solving one possible problem. Similar algorithms may solve a wide range of problems. In fact, it may be possible to write sophisticated algorithms that can perform knowledge work tasks like writing or programming as well as experts working alone. This thesis will not go that far, because programming with human computation presents a unique set of programming challenges that need to be understood and overcome first. However, given MTurk — an on-demand source of small-scale labour — the time is ripe to start building systems and running experiments that lay the groundwork for writing complex human computation algorithms.

This thesis makes three main contributions. The first contribution is the crash and-rerun programming model, along with the TurKit toolkit which uses this model. Crash-and-rerun is suited to the unique challenges of programming with human computation. In particular, hiring humans introduces latency. A simple process like the one above can take hours to complete. This is a problem for developers writing a new algorithm, since they may need to rerun the algorithm many times to debug it and figure out how it should work. The crash-and-rerun model helps by memorizing the results from human workers so that they can be replayed without cost or delay the next time a program runs. We implement the crash-and-rerun programming model inside TurKit, and offer an API for programming on Mechanical Turk using this programming model.

The second contribution of this thesis is a set of experiments that explore the trade-offs between two basic human computation algorithms: iterative and parallel. Fig. 2.7.1 demonstrates the iterative algorithm, where a series of workers iteratively improve on each other's work. In the parallel version of this algorithm, we would show each worker a blank slate upon which to start transcribing the passage, and we would use the same voting tasks to determine the best transcription. This thesis applies each 20 algorithm to several problem domains, including describing images, brainstorming company names, and transcribing blurry text. We find that iteration increases the average quality of responses from each worker, but also decreases the variance. We also find that iteration can get stuck in a local maxima, similar to a gradient ascent algorithm.

The third contribution is a case study of a human computation process designed to solve a real-world problem, that of digitizing handwritten forms. We present the design of the process which involves three phases. Each phase is a simple iterative or parallel algorithm. The first phase locates fields on a blank form; the second phase produces labels for each field; and the third phase transcribes the handwritten information in each field of each form. We execute the process on a sample of real world data, and discuss the results. We also make suggestions for researchers or practitioners working on end-to-end systems of this sort.

The contributions above are motivated by a grander vision of where human computation can take us. It may be possible to take many knowledge work tasks that are traditionally carried out by single experts, and break them down into algorithms which orchestrate the efforts of many individuals. These algorithms may perform tasks more efficiently and reliably than experts, doing for knowledge work what the industrial revolution did for manufacturing. In a sense, each of the contributions of this thesis is a stepping stone toward exploring this possibility. This thesis embarked along an ambitious path to create complex human computation algorithms capable of writing essays, designing software, and creating presentations. I had even hoped that this thesis would be written using such an algorithm — alas. Still, this thesis makes several contributions which I hope will help others advance the field of human computation.

First, we presented a toolkit for writing human computation algorithms on MTurk. The high latency of human computation presents a unique set of design constraints for human computation programming. The TurKit toolkit helps manage this high latency using the crash-and-rerun programming model, which remembers costly results between runs of a program so that they do not need to be redone. In particular, the system remembers the results of HITs posted on MTurk so that they do not need to be re-posted. This allows programmers to maintain a relatively fast iterative design cycle as they prototype new algorithms, while still programming in a familiar imperative style. We focused on trying to maintain a familiar programming style. We wanted to maintain the illusion of introducing a human procedure call to a conventional programming paradigm. However, our efforts come with some caveats. For instance, the crash-and-rerun programming model differs in some subtle ways from traditional programs, which are not always clear to new programmers in this model. Also, the crash-and-rerun model does not scale to continuously running processes, since it will take longer and longer for the program to get back to where it was after rerunning.

Finally, the parallel programming capabilities of the system are difficult to discover and use. There are many decent alternatives to crash-and-rerun programming that fix the scalability and usability issues. However, the crash-and-rerun programming style introduces some compelling advantages difficult to find elsewhere. Most importantly, crash-and-rerun programs can be modified between reruns. This allows a programmer to write the first part of an algorithm, make sure it works, and then write the next part of the algorithm without needing to wait for the first part to execute on MTurk. In fact, it is possible to modify parts of an algorithm which have already executed, in order to collect additional information into variables for later use, or print debugging information. The purpose of TurKit is to promote exploration in the field of human computation, and we site several case studies where people have used it toward that end. TurKit is open source, and available at http://turkit-online.appspot.com.

We used TurKit ourselves to implement and compare iterative and parallel human computation processes. In the iterative approach, each worker sees the results from previous workers. In the parallel process, workers work alone. Our experiments apply each algorithm to a variety of problem domains, including writing, brainstorming, and transcription. We use TurKit to run several instances of each process in each domain. We discover that iteration increases the average quality of responses in the writing and brainstorming domains, but that the best results in the brainstorming and transcription domains may come from the parallel process. This seems related to the greater variance of responses generated in the parallel condition. We also see that providing guesses for words in the transcription domain can lead workers down the wrong path, like a gradient descent toward a local minimum. More experiments like this will need to be run in order to really chart the space of low-level human computation algorithms, but this is a start. We need a foundation upon which to build larger algorithms, and hopefully these experiments contribute to that foundation.

Next, we examine a case study of building a larger human computation system, composed of several phases, each of which uses an iterative or parallel algorithm to achieve some goal. The system performs transcription on a set of hand-completed forms, with an effort not to reveal too much information from the forms to the human workers. We decompose the problem into three phases: the first phase identifies information regions on the form, the second phase provides labels for each region, and the final phase has humans transcribe what is written in each region. We ran this system on a real-world set of forms, and discovered a number of pros and cons. For the most part, the system worked well, including the system for dynamically partitioning the region selection task between workers. No worker needed to draw all the regions, but we also did not need to divide up the form upfront.

On the other hand, we discovered a number of risks related to showing transcription workers only a small window of a form, since sometimes the transcription would be aided by seeing other parts of the form. After getting our hands dirty building tools, systems and running experiments, we come away with an impression of the state-of-the-art of human computation. We express our own view of where we are at, what problems we face, and where we think human computation is headed. I think the most pressing problem today is getting more people to use human computation — the more people use it, the more brains will be motivated to fix all the problems. The biggest hurdle to getting more people to help explore human computation is getting them to understand what can be done, making it easy for them to do it, and ensuring high quality results. Also, we should not just think about this from a bottom-up approach, getting people to use micro tasks on MTurk. We should also consider exploring top-down approaches, looking at how modern knowledge workers go about their tasks, and finding possible pieces to outsource as part of their workflow.

One thought to leave with is this. Humanity has achieved many great things over the course of human existence, even though individual humans only live for 100 years. Hence, a 100 year window is enough for a person to learn the current state of-the-art in humanity, at least in some narrow domain, and make a contribution in that domain that allows the next person who comes along to get even further. We also see companies with achievements surpassing any individual worker, where each employee only works perhaps 30 years at the company. Today, many people move between companies every 5 years. How small can we make this window? If any one person can only contribute to a company for a month, can it achieve arbitrarily great things? If we design a human computation algorithm with clever protocols for focusing workers on different sub-problems within a system, and present them with just the most relevant information, can we make this window as small as a day, or an hour? If so — a big if — can we parallelize the algorithm? And if so, what can nearly 7 billion people achieve in a day using this algorithm?

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 PROJECT DESIGN

In this project, we need to solve the problem of WSD using human computation. The best way to incorporate human computation in any problem ethos is through an interactive game. Therefore, we are going to base our human computation component on a fun game in which players have to provide alternate words for the highlighted word in a phrase or sentence. Additionally, they have to suggest the alternate word such that it matches with the alternate suggestion of their opponent. The opponent in the game would be more of a team partner than an opponent because both players are aiming to guess the same word as the other, and thus will contribute to each other's scores. However, the person who submitted the alternate word first will get more points than their opponent. This part is necessary because it will give added incentive to players to guess more words in the allotted time. The words provided by the users will be stored in the database along with the count of the number of times that the users have suggested those words during the game. We shall use that count to determine whether the alternate suggestion of the users is valid or not. This will help our corpora increase in volume and accuracy automatically, purely based on how the users have played the game. Therefore, more the number of users who play the game, the bigger our corpora will be.

The game will be programmed in Java because of its great availability of frameworks and libraries for client-server based programs, and also because our programming expertise lies in this language. Also, database connectivity with Java is relatively simple as compared to other programming languages while being sufficiently secure. Another advantage of using Java will be the easy portability of our game. We need not program our game differently for different platforms. Also our game will be very light on the memory as well as the processor because it will be compiled by the JVM which is already present on the user's machine.

3.2 GAME DESIGN

The game will be a client-server program written in Java using the NetBeans IDE. We shall use the JFrame container for containing our design elements.

3.2.1 Register

The register interface looks like this:

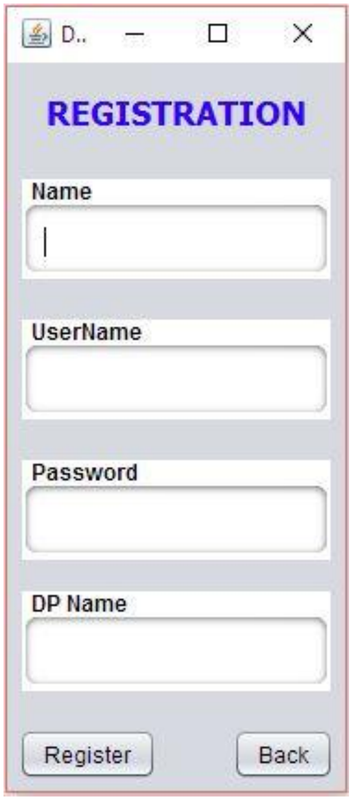


Fig. 3.2.1.1: Register Interface

In this JFrame, the user has to register, so that their stats and login information can be stored in the database. Here the various fields that the user has to fill out are represented by JTextField.

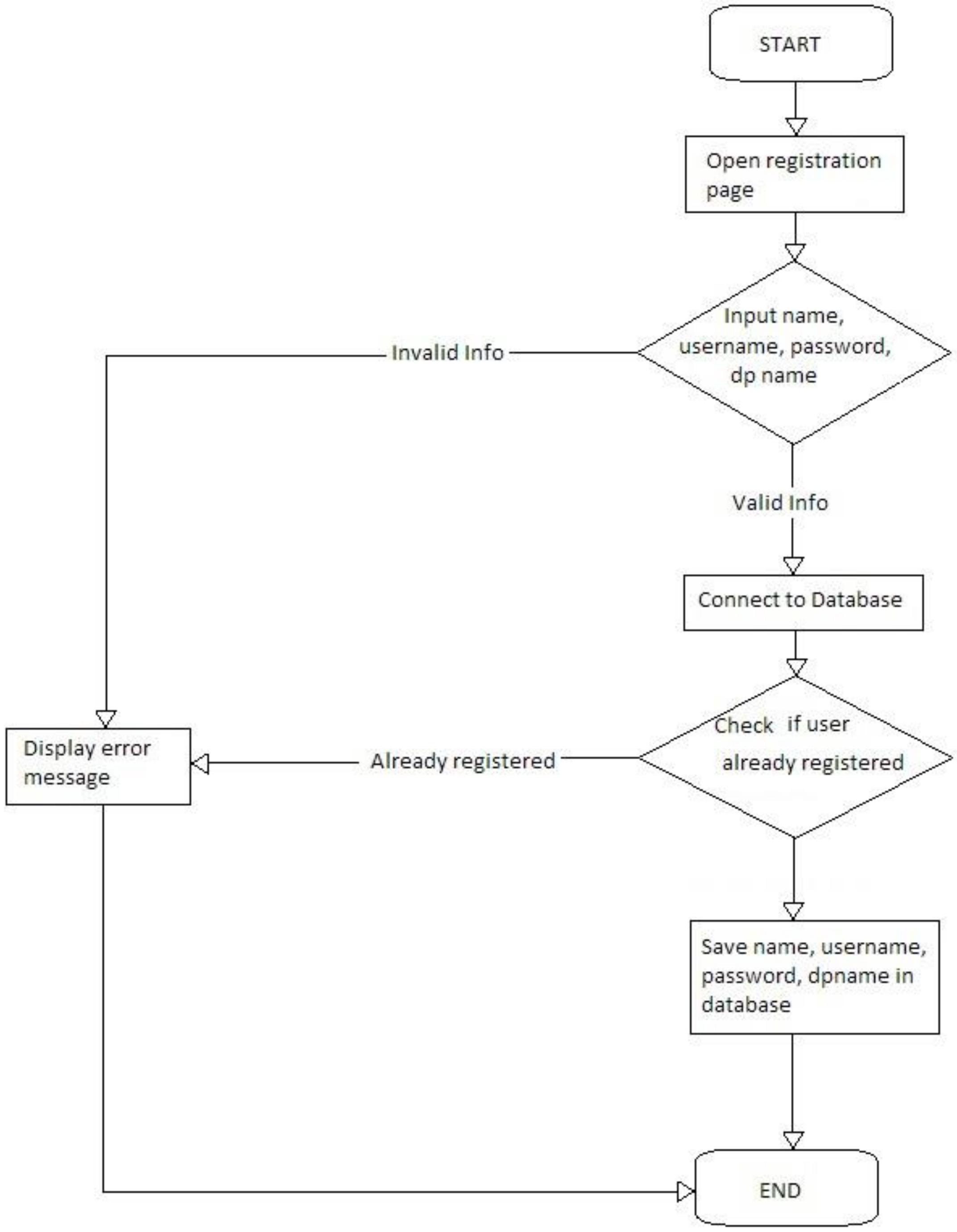

START
Open registration page
Input name, username, password, dp name
Invalid Info
Valid Info
Connect to Database
Check if user already registered
Already registered
Display error message
Save name, username, password, dpname in database
END

Fig. 3.2.1.2: Register Interface Flowchart

The register interface is shown in the above flowchart. The user needs to provide their name, a username and password, and the name of the file which contains their profile picture. Once the user hits the register button, the data gets checked against the database for any redundancy in username, and if there is no redundancy, the user gets registered.

3.2.2 Login

Once the user has registered themselves, then they can login using the login interface. Here the username and password fields have been represented by JTextField. The user needs to provide a username and a password. Once the user hits the login button, the username and password are checked against the database. If they are a match, the user is logged in.

The login interface looks like this:

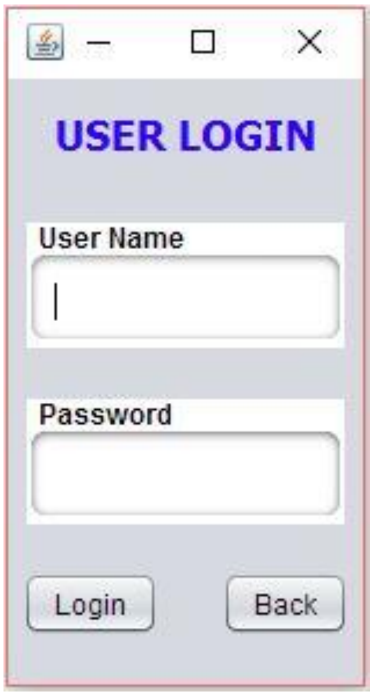


Fig. 3.2.2.1: Login Interface

The login interface can be better understood through the following flowchart:

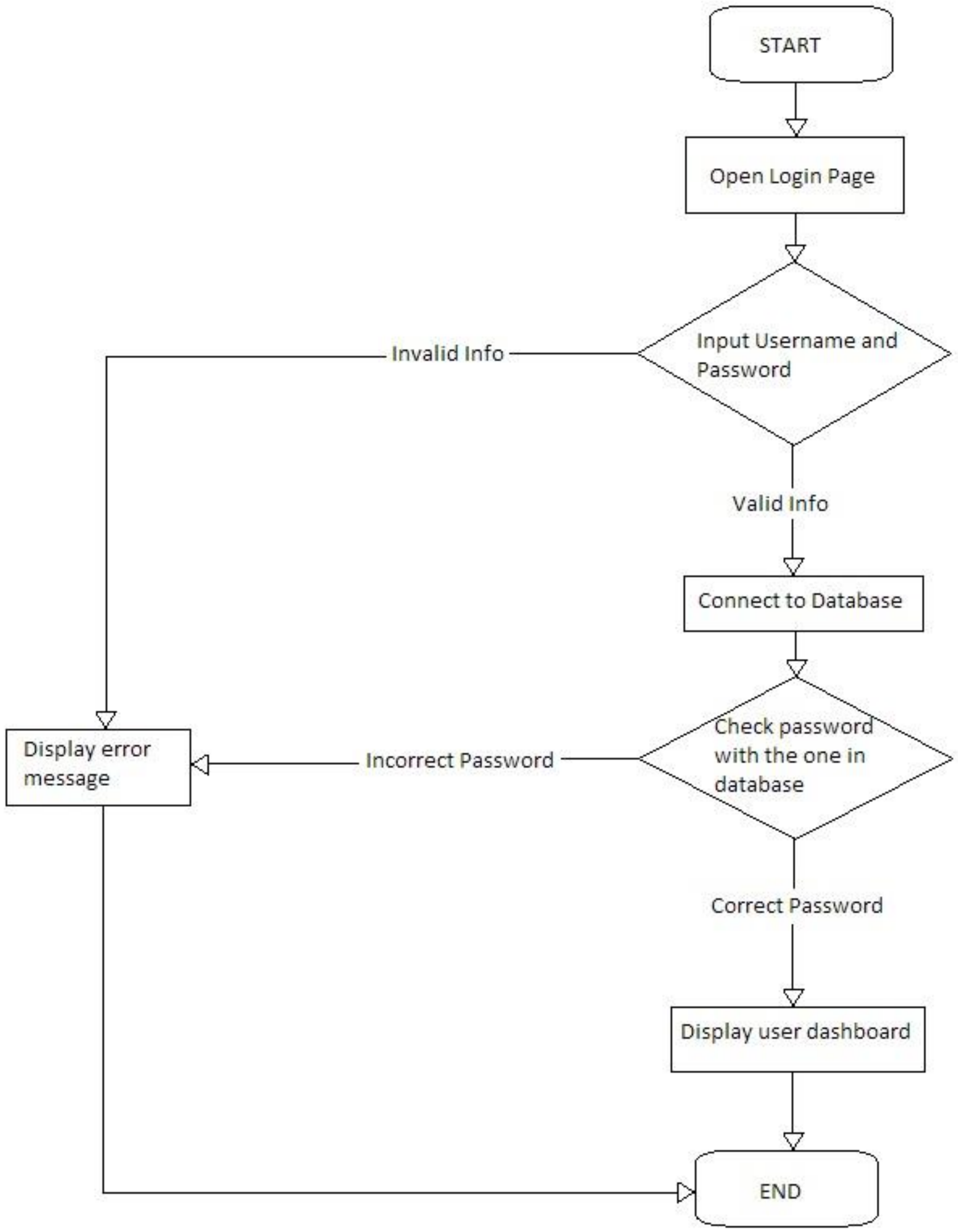


Fig. 3.2.2.2: Login Interface Flowchart

3.2.3 User Profile

The user profile interface displays essential user information and is the central pivot point from where the user can see how to play the game, can play the game, can logout, or can exit. The interface also displays the user's name at the top left corner using a JLabel and also displays the user's profile picture on the right using ImageIcon.

The user profile interface looks like this:

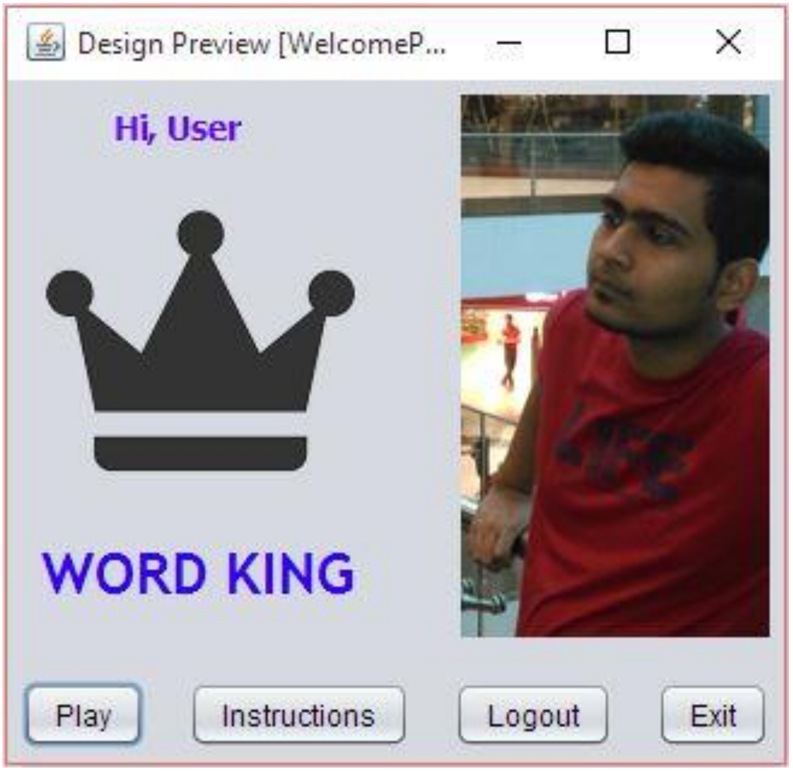


Fig. 3.2.3.1: User Profile Interface

The user profile interface can be better understood through the following flowchart:

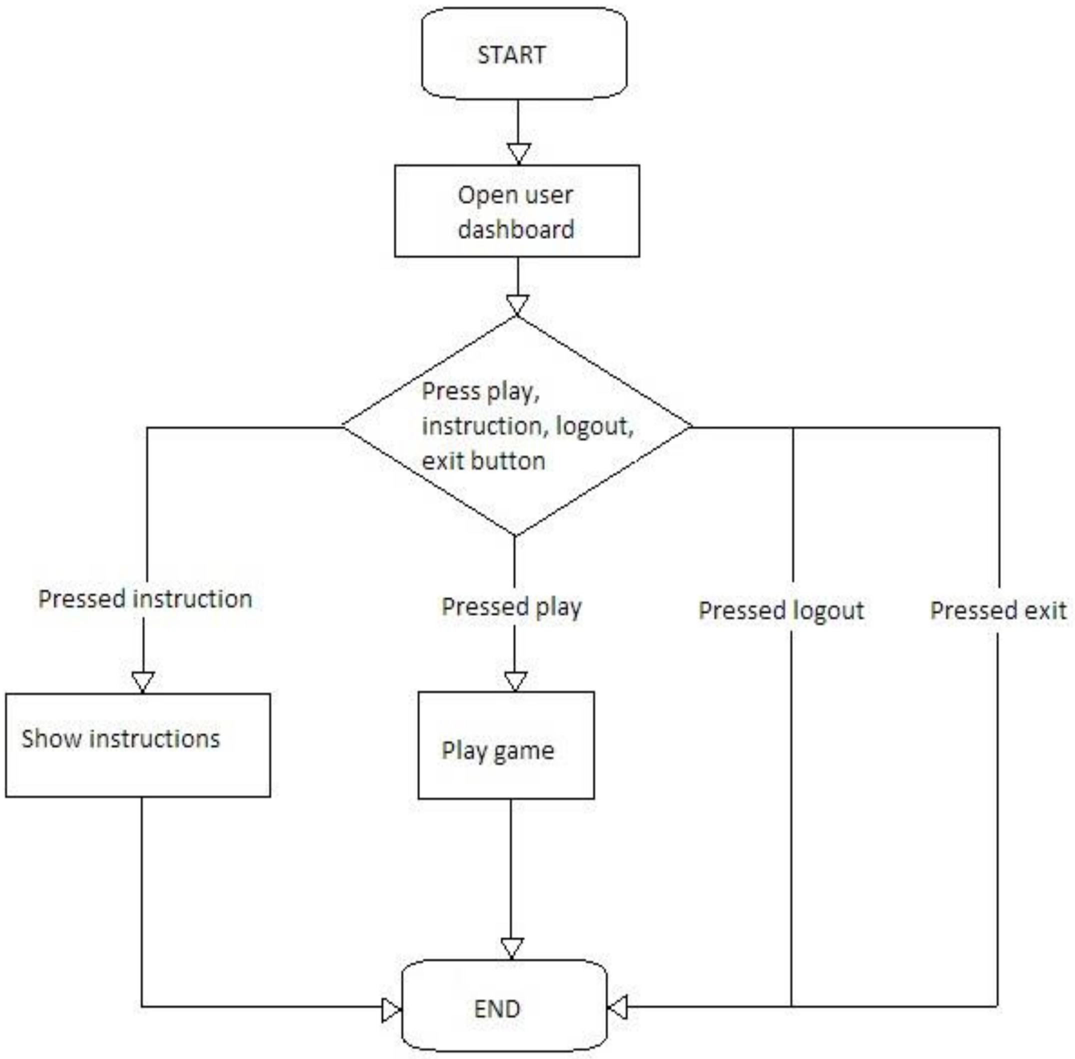


Fig. 3.2.3.2: User Profile Interface Flowchart

3.2.4 Game Interface

Once the user hits the play button in the user profile interface, the game interface pops up. In the game interface, a JTextArea has been used to display the sentence or phrase in which there will be a highlighted word for which the player has to suggest an alternative word.

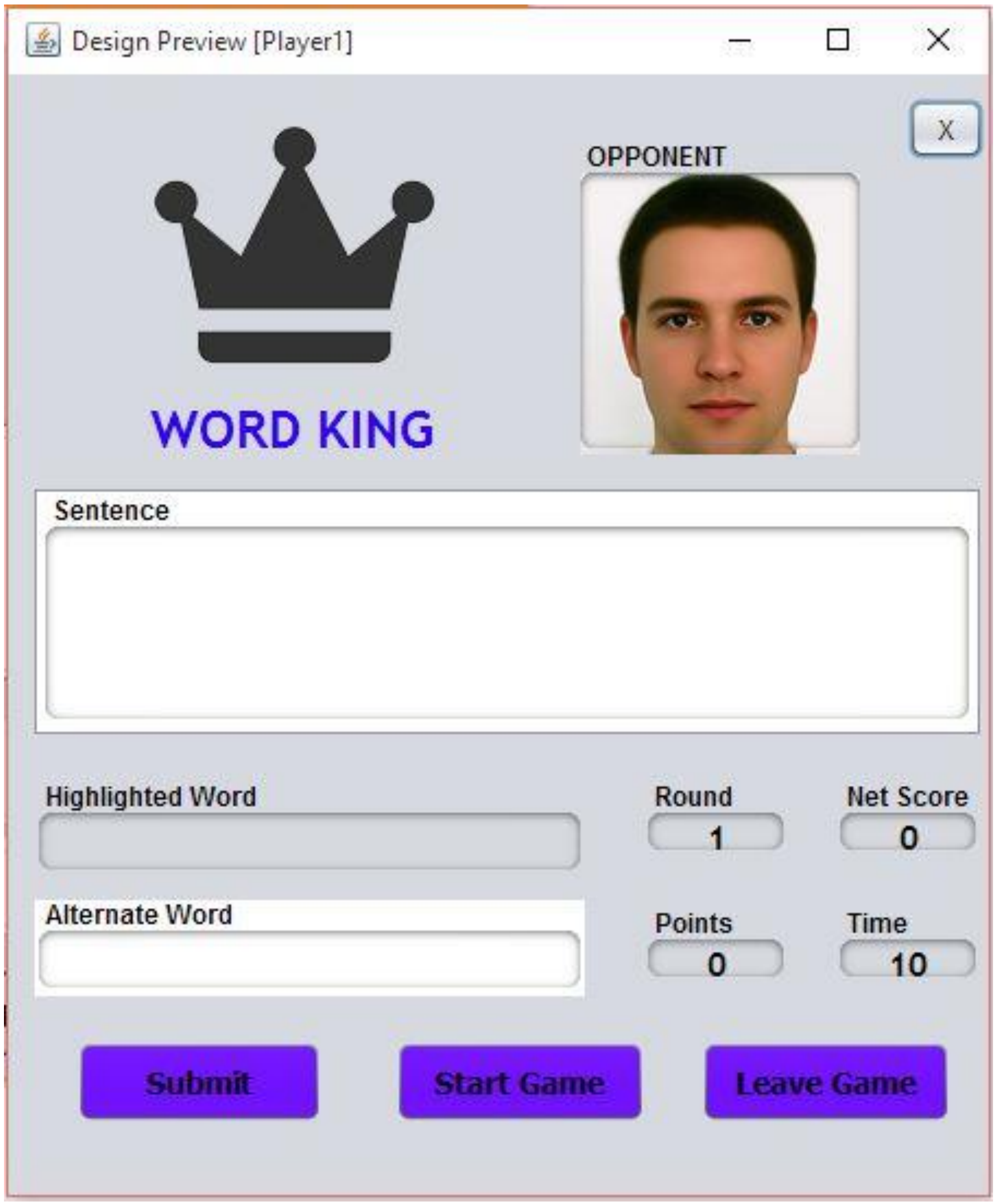


Fig. 3.2.4.1: Game Interface Design

The highlighted word for which an alternate word has to be provided is represented below the sentence text area using a JLabel. We have provided the player with an editable JTextField for them to enter their alternate word. The player can then submit the word using the submit button. Just right to the alternate word text field, there are four JLabels, for round number, points for current round, net score and time remaining. The player can then see if their alternate word suggestion matched that of their opponent in the points label. The player can also check the time remaining for the round in the time label. The interface also shows the profile picture of the opponent in the top right corner using ImageIcon.

3.2.5 Game Play

The two players will be anonymous and will be paired randomly to prevent any form of cheating between the two players. At any given time, both players view the same round, where a round is defined by a context (sentence/paragraph) and a highlighted word within that context.

The players are encouraged to rapidly type replacement words for the highlighted term. They are given incentive to type words that their partner is likely to type and are given 30 seconds to do so.

If the players' guesses are a match then they are awarded points. But both the players are not awarded equal points. The player that first entered the matching alternate word gets more points than the other player who entered the matching alternate word. In the case where an agreement cannot be reached, and the guess of the player matches any word corresponding to the highlighted word in the database, then the player is awarded some points, but not as much as they would be getting on a perfect match with the other player. But if the players' guesses are altogether incorrect, then they are awarded 0 points for that round. One round lasts for 30 seconds, and the players are encouraged to provide as many words as they can in that time frame.

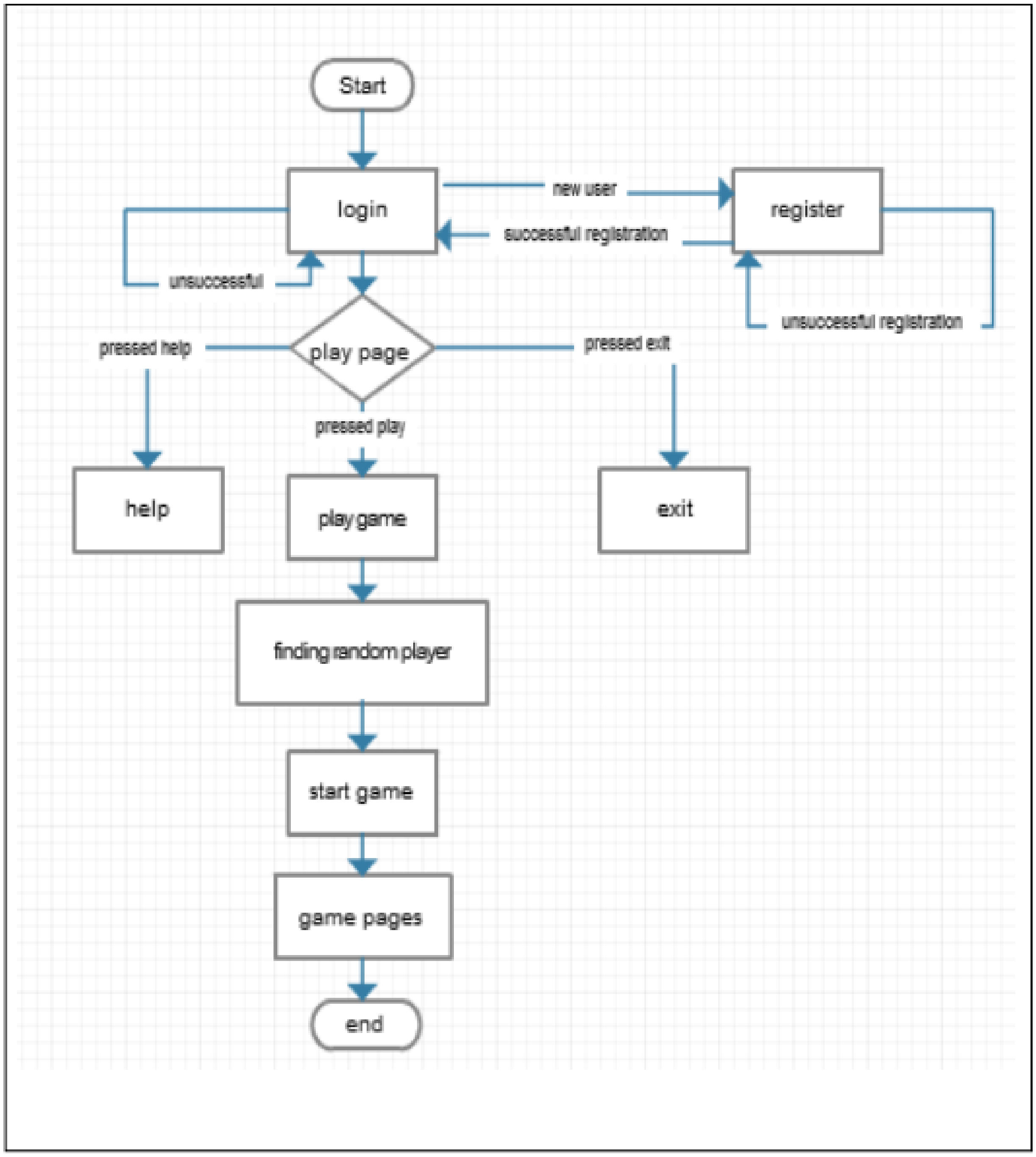


Fig. 3.2.5.1: Flowchart of the Game

A game lasts for 7 rounds, with the last round being a bonus round. In the bonus round, the amount of points that each player gathers is doubled. At the end of the game, the player with the most number of points wins. After the game is completed, both the players' scores are checked against their high scores that are stored in the database. If their score is more than their previous high score, then that score is now counted as the high score for that particular player.

3.2.6 Database Manager

The DB Manager JFrame can be used to add words and sentences for the game. The interface looks like this:

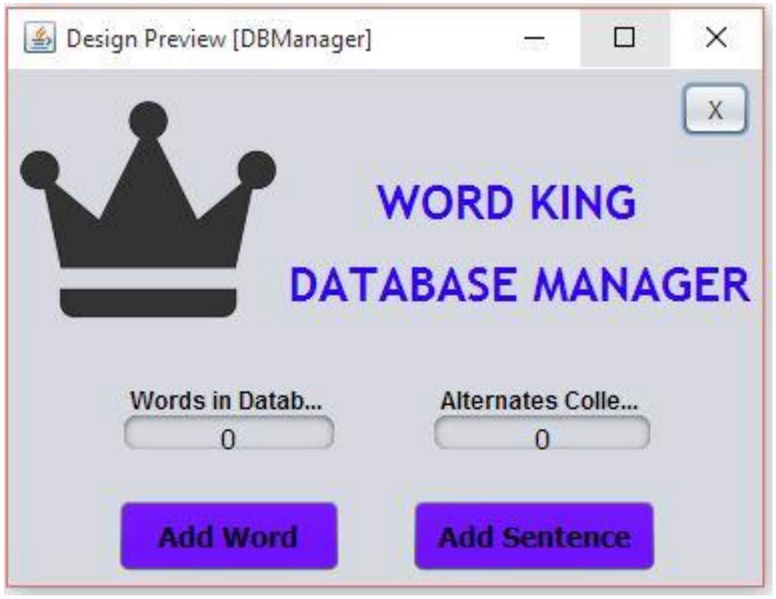


Fig. 3.2.6.1: Database Manager Interface

User has options to add both words and sentences to the database. But if the user wants to add a word or a sentence, they will have to provide the admin password to be able to successfully add a word or a sentence, otherwise their request will be denied. The database manager interface can be better understood through the following flowchart:

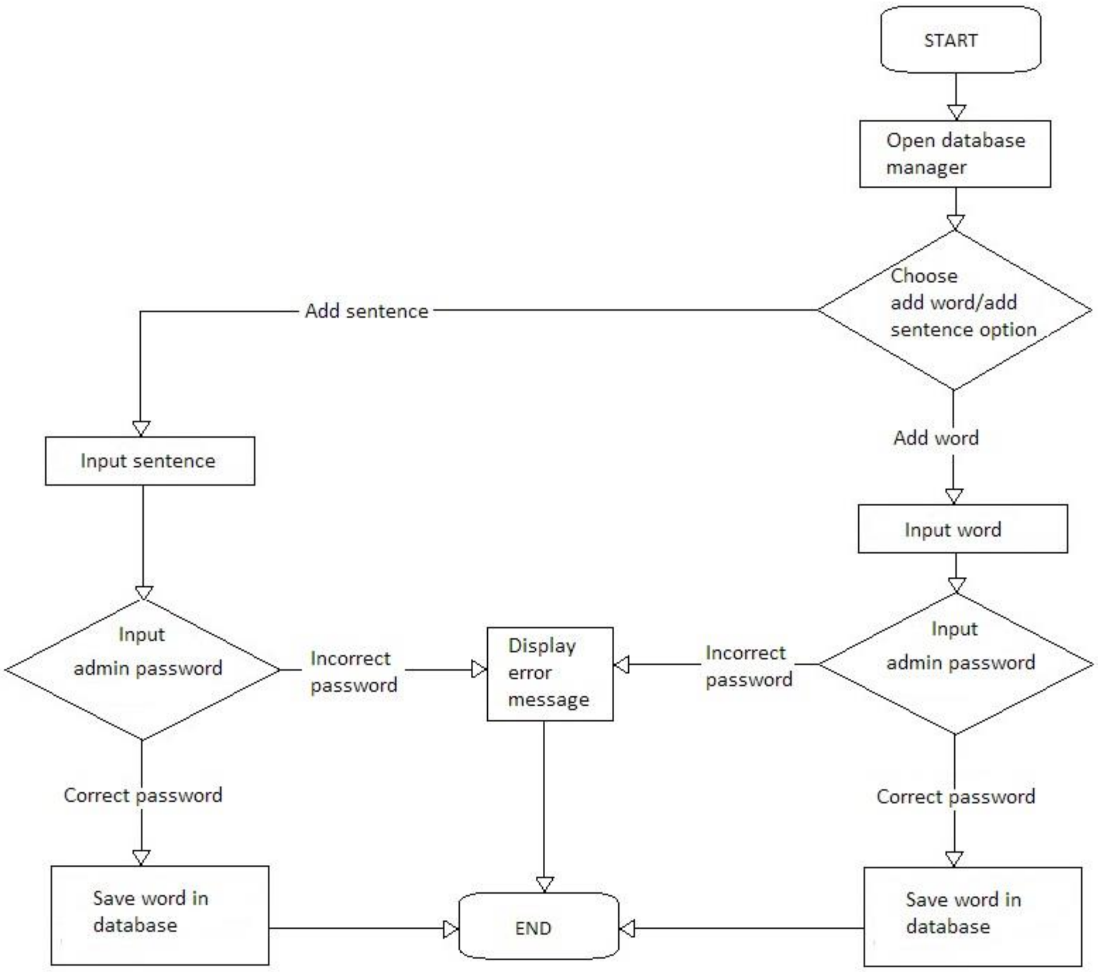


Fig. 3.2.6.2: Database Manager Interface Flowchart

3.2.7 Corpora Acquisition

Once the players enter their alternate suggestions for a highlighted word, their answers are checked. If they are a match, then we store the word in our database and set its count to 1. If another set of players suggest the same alternative word for another game, then we increase the count of the word to 2, and so on.

If the count of the word crosses a certain threshold, say 20, which means that 40 people think that that word is the correct alternative to the highlighted word in the sentence. So therefore, then we treat the word as being a correct alternative to the word that was highlighted in the sentence. In this way, we can build a database of words having similar meanings, and words whose meanings might be different in different contexts. Thus, the more players play our game, the better our corpora will become.

We store our words and sentences in a database and whenever the application is launched, we create a trie from all the words in the database. We also store the alternate words in a separate database and upon the application launch, we create hashmaps from all the corresponding alternate words in the database. During the game, the alternate word count is changed dynamically in the hashmap and it reflects in the database only after the game has been completed. Periodically we check the alternate word database, if any alternate word count is above 20, we shift the alternate word to the main word database so that it is part of the trie and not the hashmap.

Thus our technique will be a good solution to the corpora acquisition bottleneck, which has been an everlasting problem in Natural Language Processing (NLP).

CHAPTER 4- PERFORMANCE ANALYSIS

4.1 INTRODUCTION

In computer science, the performance analysis is the determination of the amount of resources necessary to execute algorithms. Most algorithms are designed to work with inputs of arbitrary length. The efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps which is known as time complexity or storage locations which is known as space complexity.

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimates of the resources needed by any program which solves a given computational problem. These estimates provide an insight of search for efficient algorithms into reasonable directions

In theoretical analysis it is common to estimate their complexity in the asymptotic sense which means to estimate the complexity function for arbitrarily large input. For this purpose Big O notation, Big-omega notation and Big-theta notation are used. Exact (not asymptotic) measures of efficiency can sometimes be computed but they usually require certain assumptions concerning the particular implementation of the algorithm, called model of computation.

Time efficiency estimates depend on what we define to be a step. For the analysis to correspond usefully to the actual execution time, the time which is taken by any program to perform a step must be guaranteed to be bounded above by a constant. One must be careful here, for instance, some analysis count an addition of two numbers as one step. This assumption in certain contexts may not be warranted. For example, if the numbers involved in a computation may be arbitrarily large, the time required by a single addition can no longer be assumed to be constant.

Run-time analysis is a theoretical classification that estimates and anticipates the increase in running time of an algorithm as its input size increases. Run-time efficiency is a topic of great interest in computer science. A program can take seconds, hours or even years to finish executing, depending on which algorithm it implements.

4.2 PERFORMANCE ANALYSIS RESULTS

So, upon performance analysis of the application and its algorithms, following time complexity was calculated for the various tasks happening during the application:

S.No	Task	Time Complexity	Remarks
1	Creating dataset	O(n*m)	n = number of words, m = length of longest word
2	Inserting word into dataset	O(k)	k = length of word
3	Inserting hashmap at the leaf node of a word	O(k)	k = length of word
4	Inserting alternate word into hashmap	O(1)	
5	Searching word in dataset	O(k)	k = length of word
6	Searching alternate word in hashmap	O(k+1)	k = length of word in whose leaf node the particular hashmap is stored

Table 4.2.1: Performance Analysis Results

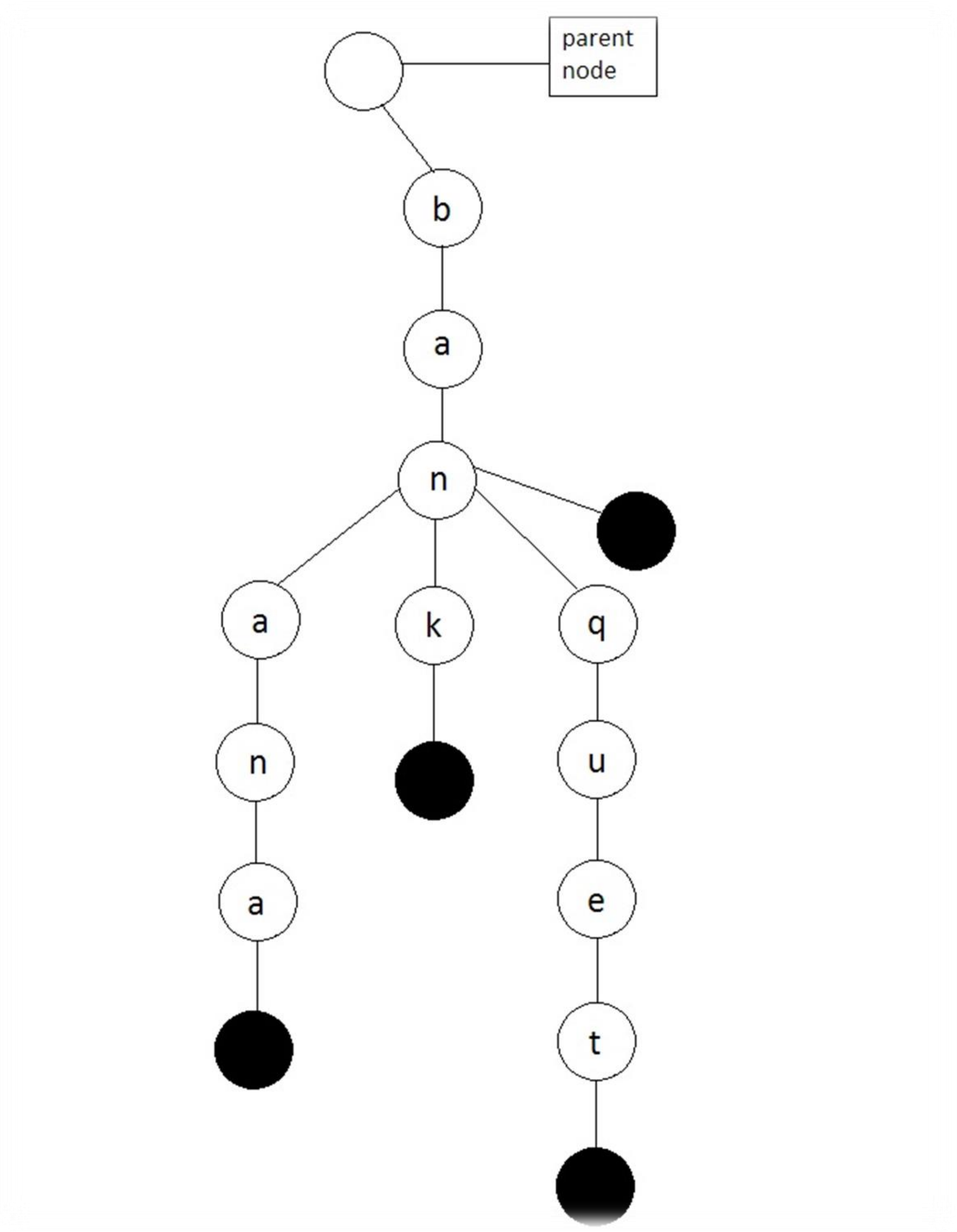


Fig. 4.2.1: Normal Trie

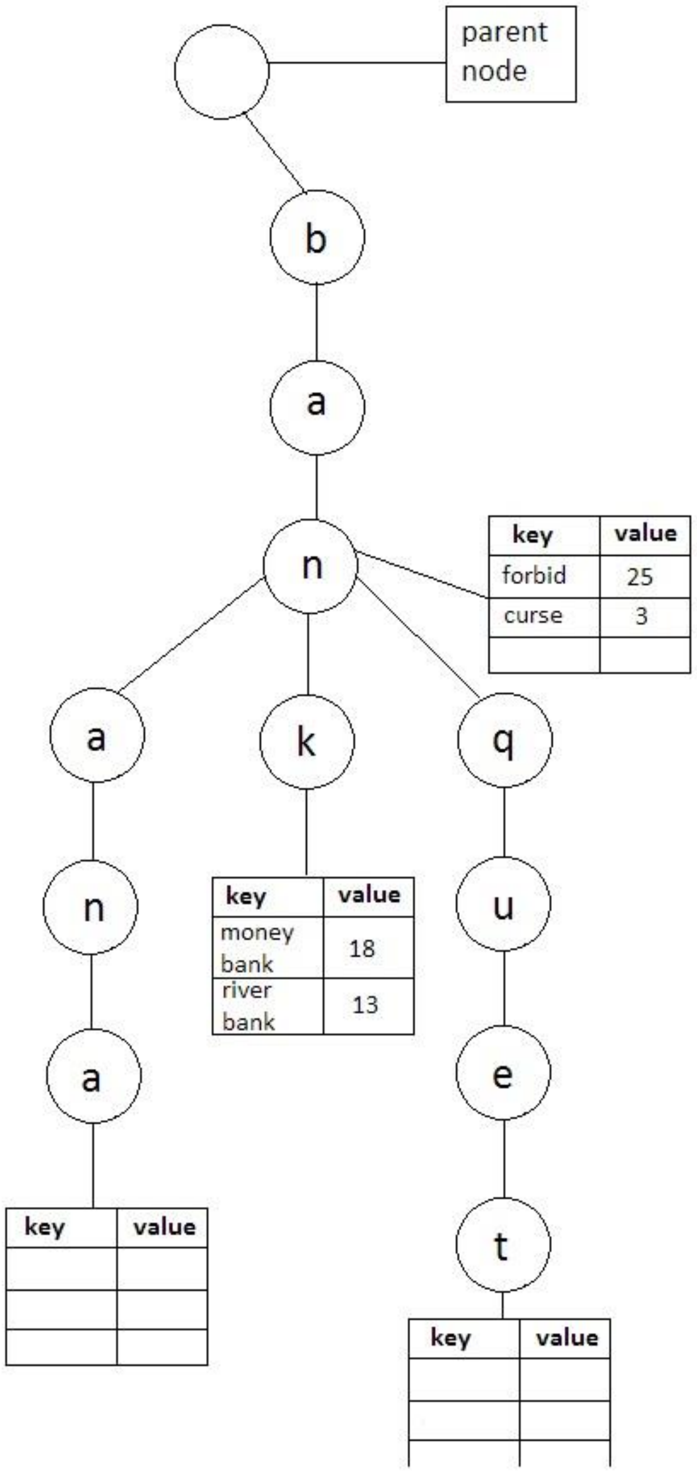


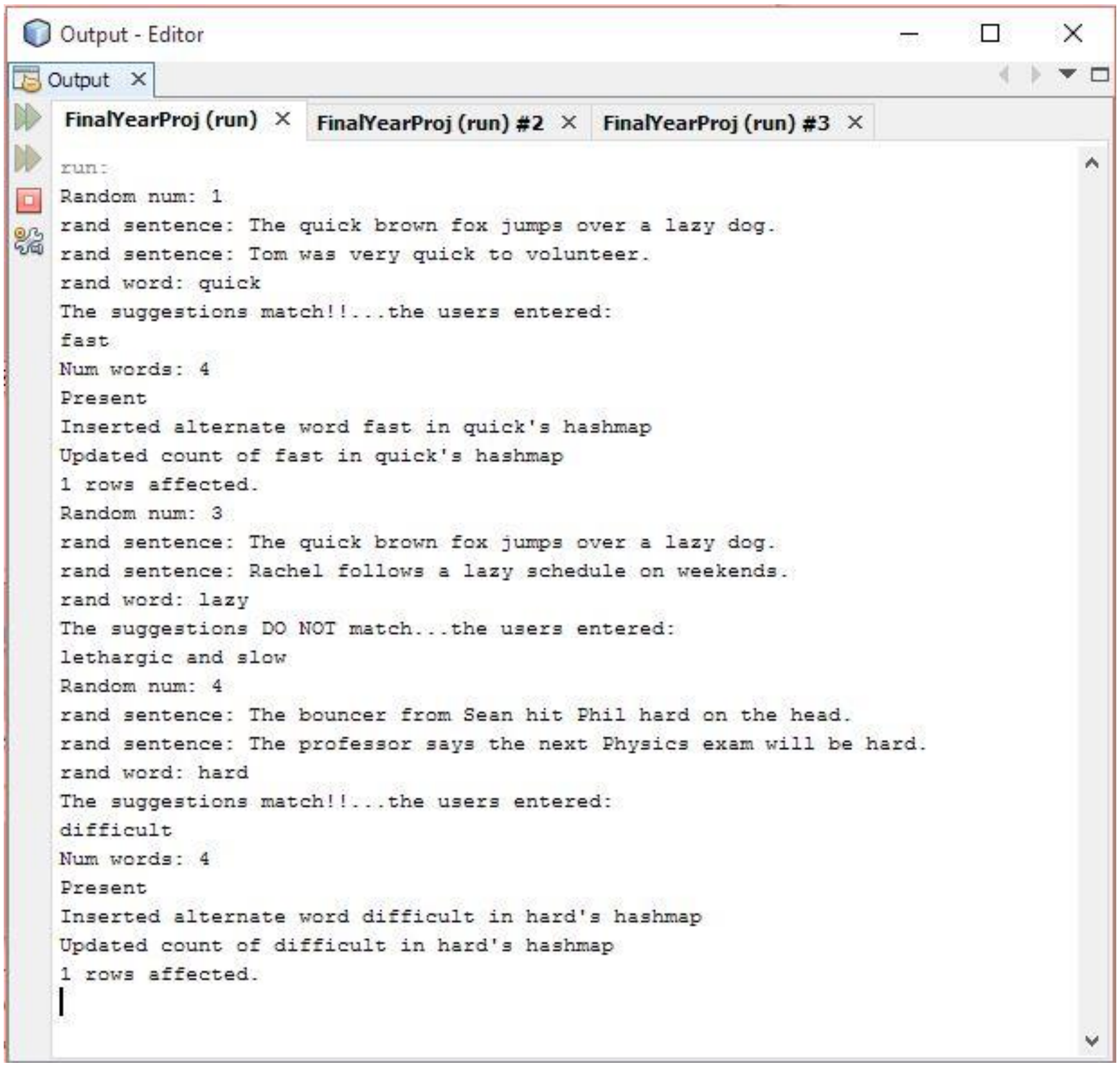
Fig. 4.2.2: Modified Trie with Hashmaps

The main thing to notice here is the time complexity of the creation of dataset from words stored in database. The time complexity here is O(n*m) where n is the number of words and m is the length of the longest word. This is because we have used trie data structure (fig. 4.2.1) to store the words and the time complexity mentioned is the complexity of creation of trie. Furthermore, we have modified the trie data structure to give us the ability to store the ‘word sense’, i.e., in which context the word can be used. We have used hashmaps (fig. 4.2.2) to store the alternate words to a given word, so that context can be derived from those alternate words whenever needed. Now since the time complexity of both insertion and searching in hashmap is O(1), the hashmap adds negligible overhead to the already efficient trie dataset, which makes our dataset very light and fast.

4.3 OUTPUT

4.3.1 Server Output

The server is the place where the trie and hashmaps are stored, and words are randomly selected from the dataset and corresponding sentences are shown to the players. Then the players submit alternate suggestions for the given words. The players’ suggestions are matched at the server, and if they match then the server updates the corresponding hashmap with the alternate suggestion’s count. For a simulation game with 3 rounds, the following sample output was recorded on the output console of the NetBeans IDE:



```
run:
Random num: 1
rand sentence: The quick brown fox jumps over a lazy dog.
rand sentence: Tom was very quick to volunteer.
rand word: quick
The suggestions match!!...the users entered:
fast
Num words: 4
Present
Inserted alternate word fast in quick's hashmap
Updated count of fast in quick's hashmap
1 rows affected.
Random num: 3
rand sentence: The quick brown fox jumps over a lazy dog.
rand sentence: Rachel follows a lazy schedule on weekends.
rand word: lazy
The suggestions DO NOT match...the users entered:
lethargic and slow
Random num: 4
rand sentence: The bouncer from Sean hit Phil hard on the head.
rand sentence: The professor says the next Physics exam will be hard.
rand word: hard
The suggestions match!!...the users entered:
difficult
Num words: 4
Present
Inserted alternate word difficult in hard's hashmap
Updated count of difficult in hard's hashmap
1 rows affected.
```

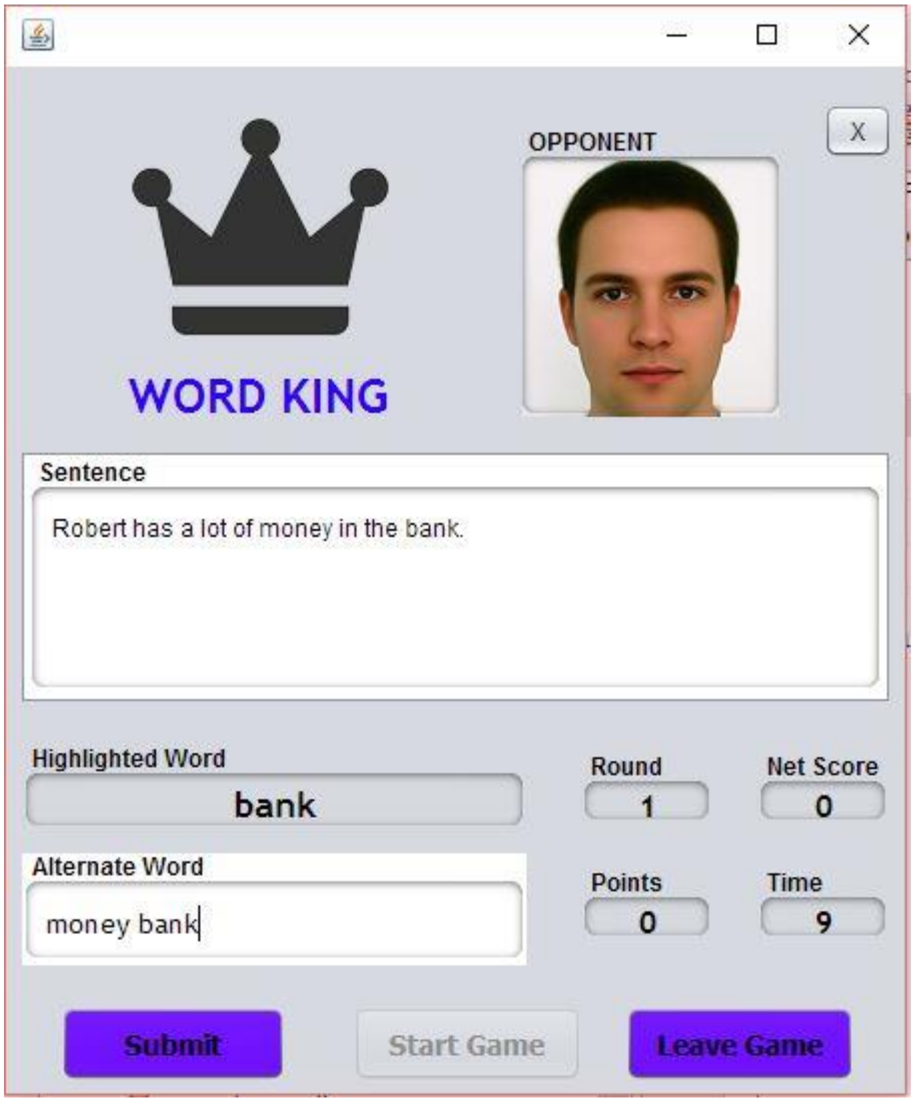
Fig. 4.3.1.1: Server Output

In the output, a random number is shown. This random number is responsible for selection of word from the dataset. Once a word is selected, a sentence associated with that particular word is selected at random.

4.3.2 Client Output

The client is the place where the game is played, and all the players see is the front end game interface. The players see a sentence and a highlighted word in the sentence for which they have to provide an alternate suggestion.

Here is a screenshot of the front end game interface during game play:



Sentence: Robert has a lot of money in the bank.

Highlighted Word: bank

Alternate Word: money bank

Round: 1

Net Score: 0

Points: 0

Time: 9

Fig. 4.3.2.1: Client Side during Gameplay

Here is the output on the NetBeans IDE's output console when the players provide alternate suggestions that do not match:

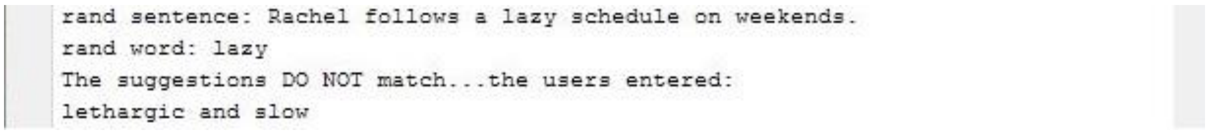

```
rand sentence: Rachel follows a lazy schedule on weekends.
rand word: lazy
The suggestions DO NOT match...the users entered:
lethargic and slow
```

Fig. 4.3.2.2: Client Output When Suggestions Do Not Match

Although the players' alternate suggestions do not match, the program still checks whether any player's suggestion is present in the hashmap of the given word. If it is, then the player is awarded points, but the count of that particular alternate suggestion is not increased.
Here is the output on the NetBeans IDE's output console when the players provide the same alternate suggestion:

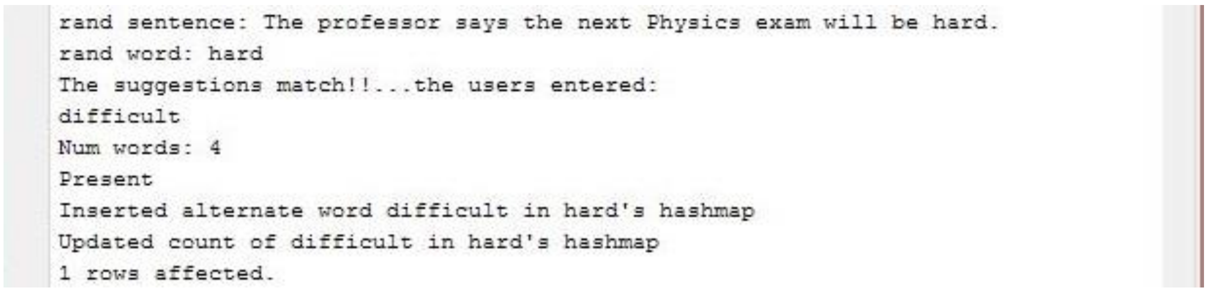
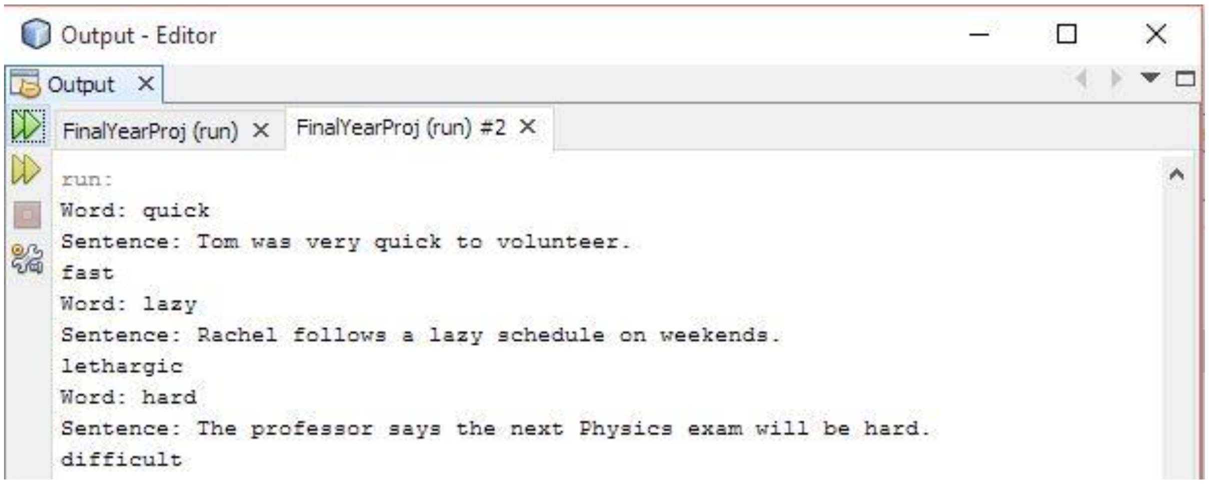

```
rand sentence: The professor says the next Physics exam will be hard.
rand word: hard
The suggestions match!!...the users entered:
difficult
Num words: 4
Present
Inserted alternate word difficult in hard's hashmap
Updated count of difficult in hard's hashmap
1 rows affected.
```

Fig. 4.3.2.3: Client Output When Suggestions Match

Here Num words: 4 is the number of alternate suggestions present in the word hard's hashmap. Then it is checked whether the word difficult is present in the hashmap, if it is, then its count is updated and increased by 1. Otherwise, the word difficult is added to hard's hashmap with count 1.
The program also prints a trace of all the sentences, the highlighted words and the player's alternate suggestions on the output console, so that the player can keep track of all the previous sentences, the highlighted words and their alternate suggestions. Here is a sample output:



```
run:
Word: quick
Sentence: Tom was very quick to volunteer.
fast
Word: lazy
Sentence: Rachel follows a lazy schedule on weekends.
lethargic
Word: hard
Sentence: The professor says the next Physics exam will be hard.
difficult
```

Fig. 4.3.2.4: Client Output Trace

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

An everlasting problem in NLP is the word sense disambiguation (WSD) problem: disambiguating the true sense of a word as it occurs in a sentence (e.g., recognizing whether the word "bank" refers to a river bank or to a financial institution). This project explores a strategy for harnessing the linguistic abilities of human beings to develop a dataset that can be used to train machine learning algorithms for WSD. To create such a dataset, we use Human Computation through a fun game designed to produce valuable output by engaging human players in what they perceive to be a cooperative task of guessing the same word as another player.

Players being unaware of the true purpose behind the game, will involuntarily help us in our corpora acquisition by trying to score maximum points by guessing as many words as possible in 30 seconds.

5.2 FUTURE WORK

Once a large enough data set has been created through human computation, the dataset can be fed to any WSD machine learning algorithm which works with the English language. We can also create similar types of games which can be used to create data sets for languages other than English. There hasn't been much research on corpora acquisition for Hindi, which is definitely an area to explore and will help in corpora acquisition for Asian languages. A similar type of game can be created which urges the players to translate the highlighted word to another language. This game can help improve machine based language translation.

REFERENCES

[1] Word Sense Disambiguation, Wikipedia (https://en.wikipedia.org/wiki/Word-sense_disambiguation)

[2] Human Computation, Wikipedia (https://en.wikipedia.org/wiki/Human-based_computation)

[3] Nitin Seemakurty, Jonathan Chu, Luis von Ahn, Anthony Tomasic: Word Sense Disambiguation via Human Computation, HCOMP '10 Proceedings of the ACM SIGKDD Workshop on Human Computation, 2010, Pages 60-63

[4] Wanxiang Che and Ting Liu: Word Sense Disambiguation Corpora Acquisition via Confirmation Code, Proceedings of the 5th International Joint Conference on Natural Language Processing, Chiang Mai, Thailand, November 8 – 13, 2011, pages 1472–1476

[5] Bridget T. McInnes and Mark Stevenson: Determining the difficulty of Word Sense Disambiguation, Journal of Biomedical Informatics, October 28, 2013, Pages 83-90

[6] Luis von Ahn: Human Computation, December 7, 2005, CMU-CS-05-193

[7] Greg Little: Programming with Human Computation, June 2011, Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

[8] Knowledge Acquisition Bottleneck, Quora (https://www.quora.com/What-is-knowledge-acquisition-bottleneck)

[9] Alexander J. Quinn, Benjamin B. Bederson, “Human Computation: A Survey and Taxonomy of a Growing Field”, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2011. pp 1403-1412

[10] Luis von Ahn, Wikipedia (http://en.wikipedia.org/wiki/Luis_von_Ahn)