ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION

Paper ID #49651

Learning Languages through Interactive Gaming

Mr. Colby Edward Kurtz, Houston Christian University

Learning Languages through Interactive Gaming

[1]Colby E. Kurtz, [2]Matthew Z. Blanchard, [3]Marian K. Zaki
[1,2] Undergraduate Cyber Engineering Students, [3]Assistant Professor of Computer Science
College of Science and Engineering
Houston Christian University
kurtzce, blanchardmz, mzaki @hc.edu

Abstract

This paper demonstrates the design and implementation of an innovative gamified software application for learning human-spoken languages. The game serves as an interactive and enjoyable supplement to aid the learning process of different languages for elementary-aged children. At its core, the application uses a translation Application Programming Interface (API) to process text and output translations in the target language chosen by the learner. Additionally, it is AI-enabled, allowing the utilization of APIs such as OpenAIs's ChatGPT to enhance the translation capabilities. Provided is a basic proof of concept that was developed as part of the Final Pi Project in the Intermediate Computer Programming (COSC 1352) course. The gamified program was written in Python, utilizing a modular design along with Google Translate[1] to generate progressively translated sentences. The game runs on a Raspberry Pi that is connected to a mini touch LCD screen, creating a portable handheld gaming console. The graphical adventure game interacts with the language learner through text commands. In its current implementation, it can handle translation of both full sentences and individual words within a sentence. This dynamic adjustment of the difficulty level allows the game to cater to a wide variety of language learning needs effectively.
The results of the initial version of the game are very promising and demonstrate that the program is highly scalable and capable of evolving in future iterations into a fully interactive gamified language-learning tool. Additionally, the modular design of the program provides an easy to integrate with alternative translation APIs and customization of parameters for various applications. Our proof-of-concept demo won the "Best Technical Award" at the Final Pi Project Design Expo as evaluated by faculty, industry professionals and advisory board members. These results highlight the potential of our innovative approach to redefine language learning for young learners.

Introduction

Learning a new language often presents a significant challenge, particularly in maintaining engagement, especially for young learners. Traditional learning tools frequently rely on repetitive tasks, such as writing, speaking, or reading the same translated texts multiple times. However, these methods can lead to boredom and eventually the abandonment of the learning process all together. The solution provided in this paper addresses this issue by gamifying the entire language-learning experience using a software application that is designed to keep the learner engaged and motivated.

While gamification may initially seem like a novel or untested concept, many recent studies suggest an increase in engagement from students when presented with gamified learning. For instance, Huseinovic[2], emphasize that, "In the field of language learning, gamification is a crucial element in facilitating language practice and knowledge acquiring through the use of language learning applications." Similarly, researched conducted by D. López-Fernández el al.[3] at the Universidad

Politécnica de Madrid in Spain highlights the effectiveness of teacher designed games in comparison to conventional lectures. The study shows that students taught via games performed just as high as those subject to a traditional lecture. However, students showed an overwhelming preference for learning via games rather than a traditional lecture.

As demonstrated by these studies, there is much room in the education sphere for gamified learning programs that can spur an interest in students. Such programs are particularly well-suited to foreign language education, where frequent interaction with the language – beyond mere memorization – is essential for meaningful learning.

To facilitate an engaged effective language learning environment, we developed a gamified application using the Python programing language, incorporating two distinct modules that form the foundation of the program. The first module, the Translation Module (TM), processes in-game text as input and outputs a translated version tailored to the user's skill level. The second module, the Game Module (GM), handles game's logic and graphics.

The game, titled *LangQuest*, immerses players in an adventure where they must escape from an ominous mansion. To succeed, players must gather specific items in each room as they navigate the mansion avoiding a mysterious entity that is seemingly always one step ahead. The text-based game nature allows the TM to interface with the in-game Graphical User Interface (GUI) elements. Moreover, the TM's design enables compatibility beyond this game, facilitating integration into other applications. Because of its gamified nature, player attention is better retained, thus achieving the goal of user retention for the purpose of learning a foreign language.

Figure 1 is a schematic illustration of both modules of the game along with the components of each module. This paper delves into the intricacies of *LangQuest*, exploring the design and functionalities of both the TM and GM modules. The modular design of the TM and how it works to achieve the desired outcome as well as the potential for future adaptations to address the evolving challenges in gamified language learning will be highlighted along the way.

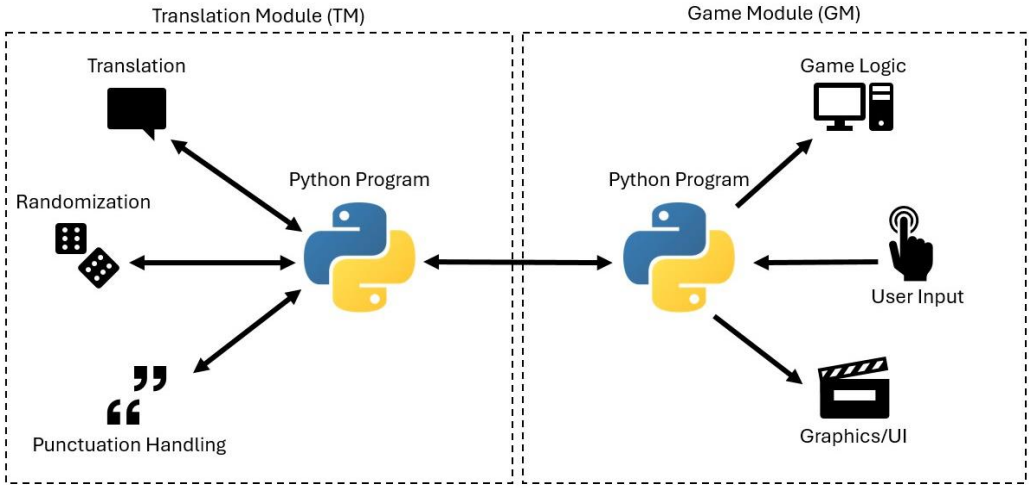

Figure 1. Basic schematic modular diagram of *LangQuest*

The Game Module (GM)

Serving as the front-end program, the GM has three main submodules that perform the following functionalities: managing game logic, handling user interaction, and rendering game graphics and the GUI. The GM controls the core of the gameplay, such as defining the rules, tracking player progress, and determining the next moves/actions based on user decisions. It also processes and responds to user text inputs, while handling user errors (such as mis-typed instructions) or incorrect navigation instructions. Additionally, the GM is responsible for generating and updating the graphical user interface that visually represents the game environment and player status.

A critical aspect of the GM is its seamless integration with the Translation Module (TM). Whenever the game requires translation, the GM communicated directly with the TM to retrieve the appropriate translations. This dynamic exchange allows the game to adapt the content to the learner's proficiency level.

1. Game Overview

The in-game user experiences centers around a suspenseful storyline designed to captivate the young learners while providing a platform for language practice. Players assume the role of Fred, an adventurer distinguished by his iconic red bandana. One day, after falling asleep, Fred awakens to find himself trapped inside a mysterious mansion with rooms to navigate, locked doors and many objects to look at and/or grab on the go. Figure 2 displays some of the mansion rooms' adventures.

Fred's goal is to escape the mansion, but achieving freedom is not a simple task. To unlock the way out, players must explore the mansion and collect certain items scattered throughout the rooms. However, the path to freedom is fraught with danger, a mysterious entity lurks around every corner, waiting to catch the players. This suspenseful setup challenges the players to strategically navigate the mansion and capture the right items to unlock their freedom. The setup encourages players to stay engaged while interacting with the in-game text and language translations challenges.

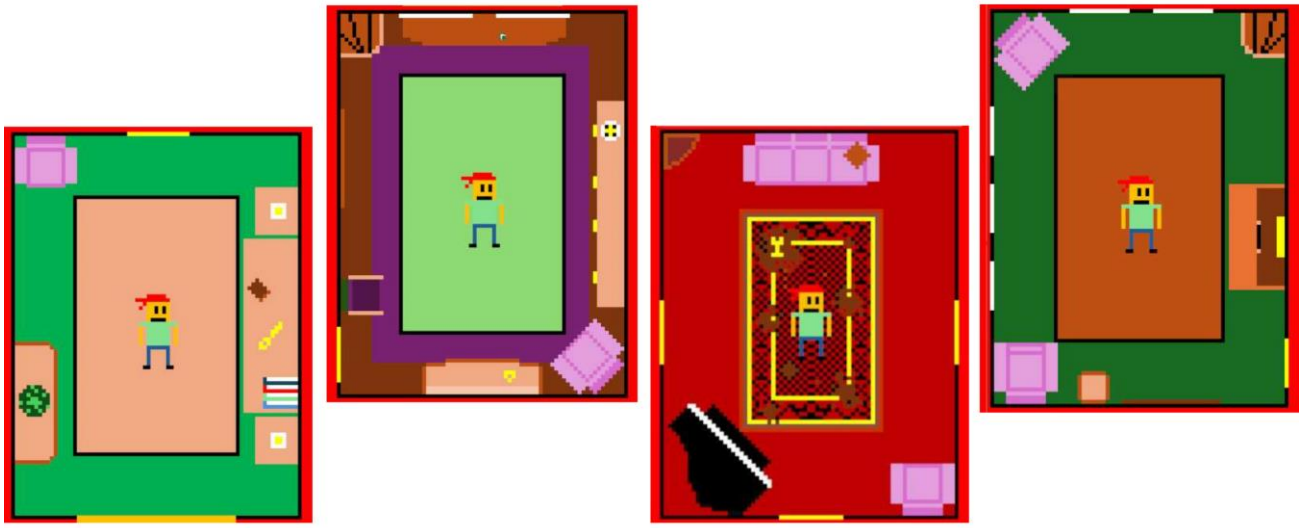

Figure 2. Example game locations

2. Game Details

LangQuest's logic revolves around the rooms that the player explores throughout the mansion. The entire game graphics are original drawings designed from scratch using Pixel Art. Doors and exits are highlighted by yellow markers on the edge of each room. The user interacts with the game by typing the commands in an entry field. For example, Figure 3 shows the starting location of the player upon entering the game. To progress to another room, the player must look around with the look command and grab certain items with the take command. In Figure 3b, the player decides to look at the doors to see if there are any exits from this room. The game tells the player in Figure 3c that there is an exit to the north and south of the current room. It is here that the TM begins to translate words based on the difficulty level of the game. The difficulty, which will be explored in detail later, is at a high level indicated by the entire sentence being translated. The player then types 'go north' based on the indicated exits. This action transforms the player to the ballroom, Figure 3d. This is a simple preview of how the game is played. Next we will explore the other elements of the game: inigames and the mysterious entity.

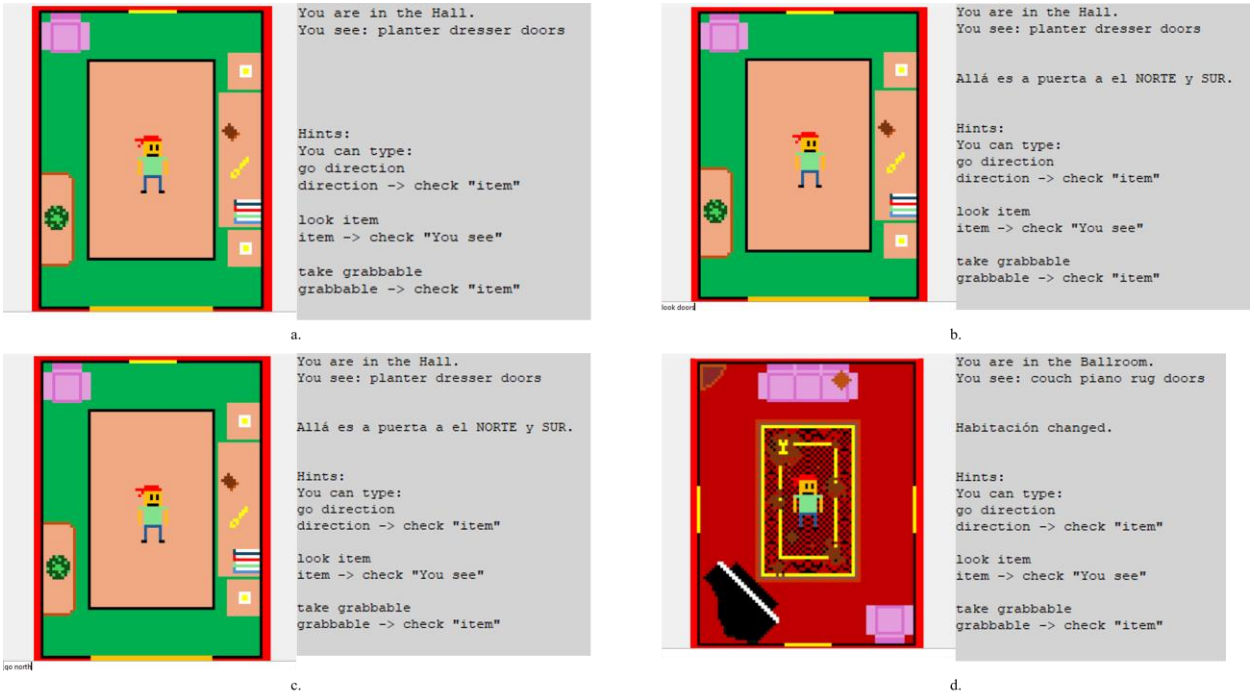

Figure 3. Early game progression

By inspecting the objects listed in each room, players can identify which items are grabbable and which are not. Successfully grabbing an item transitions the game into a **minigame state**, as shown in Figure 4, where the player is prompted to answer a basic question in the target learning language. Only by answering correctly can the item be collected, allowing the player to progress towards escaping the mansion. The collected items are essential for unlocking the final exit, forming the core progression of the game.

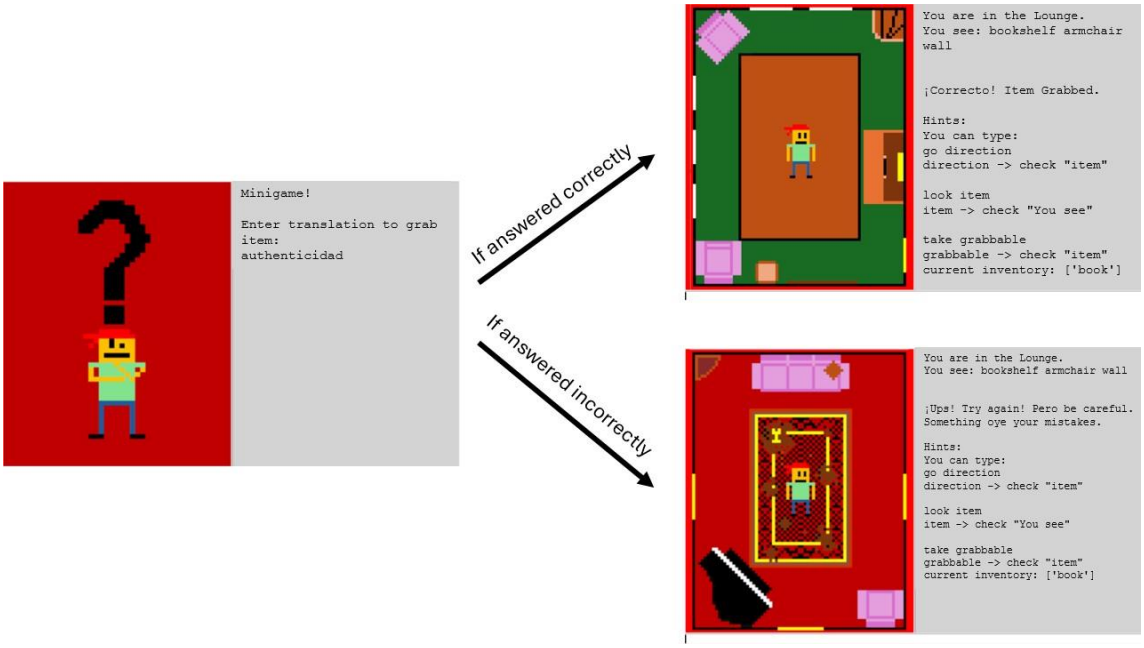

Figure 4. Minigame

An additional layer of complexity is introduced through the mysterious entity that lurks in the mansion. This entity serves as the "game over" screen and triggers one of the possible endings to the game. The entity can be encountered in three distinct ways, each designed to challenge the player's decision-making and language-learning engagement:

i. Returning to the Previous Room: The entity's basic logic places it in the player's previous location. If the player re-enters this room, the entity triggers immediately. This encourages the players to plan their movements carefully and avoid backtracking unnecessarily.

ii. Accumulating Incorrect Answers: each incorrect answer to a minigame question adds a point to a counter. Once this counter reaches a certain threshold. This reinforces the importance of accuracy and language comprehension during gameplay.

iii. Generating Too Much Noise: a sperate counter tracks player actions, such as interacting with objects or moving within a room. Each action adds a point to the counter and when the threshold is reached, the entity appears. This means that the player must be quite stealthy and strategic with how they interact with objects and move to avoid making excessive noise.

By combining the mansion exploration, the careful decision-making, and solving the language challenges, the players are engaged in the game while reinforcing their understanding of the target language. The following section expands on the technical details of the TM.

The Translation Module (TM)

One of the main focuses of the overall design was to make the code as modular and reusable as possible. Hence, the TM was decoupled from the specific details of the game, allowing for this specific module to be reused in other applications beyond the current gamified language learning application.

To achieve this modularity, the TM was designed with simple and clearly defined inputs and outputs which will be elaborated on in the following subsections.

1. Libraries

The TM utilizes a python library called `deep_translator`[1]. The library allows text translations between different languages. It interfaces with a variety of translator APIs such as Google Translate, Microsoft Translator, and could integrate with AI such as ChatGPT to translate text. This specific application relies on Google Translate [3]. The TM module issues the API calls to translate the game text based upon certain game configurations as we will discuss in later subsections.

2. Translation of Words and Sentences

The TM is designed to handle both individual word translations and complete sentences. For simplicity and efficiency, the module includes two distinct functions: one for translating words and another for translating sentences. Both functions follow a consistent pattern:

1. The desired text - whether single word or a block of sentences - is taken as input and passed to Google Translate[1] for translation.
2. The returned translated text is then reconstructed to match the original structures, which is then displayed to the player within the game interface.

```
1 def translate_sentences(text, frequency):
2     sentence_pattern = re.compile(r'(?<=[.!?])\s+')
3     sentences = sentence_pattern.split(text)
4     translated_sentences = []
5     for i in range(0, len(sentences), frequency):
6         sentence = sentences[i]
7         try:
8             translated_sentence = GoogleTranslator(source='auto',
               target='es').translate(sentence)
9             translated_sentences.append(translated_sentence)
10        except Exception as e:
11            print(f"Error translating sentence '{sentence}': {e}")
12
13    return '. '.join(translated_sentences)
```

Figure 5. Translating sentences

Figure 5 illustrates the definition of the `translate_sentences(text, frequency)` function, which is responsible for selectively translating sentences within a given block of text. The `frequency` parameter is used to control how many sentences in the input text should be translated. Within the function, regular expressions are used to detect punctuation and accurately split the input text into individual sentences. The splitting also allows the translation of selective sentences based on the

frequency. After successful translation, the text is reconstructed one sentence at a time ensuring that the original sentence order is preserved.

```
def translate_words(sentence, frequency):
    words = sentence.split()
    translated_words = []
    for i, word in enumerate(words):
        if i % frequency == 0:
            try:
                translated_word = GoogleTranslator(source='auto',
                target='es').translate(word)
                translated_words.append(translated_word)
            except Exception as e:
                print(f"Error translating word '{word}': {e}")
        else:
            translated_words.append(word)

    return ' '.join(translated_words)
```

Figure 6. Translating words

When translating individual words, both standalone and within sentences, the same general approach was used as with translating sentences. Figure 6 illustrates the `translate_words(sentence, frequency)` function which oversees splitting the words within the sentence and translating each word independently. The frequency parameter also controls the selection of the words to be translated. Note that a frequency of 1, would result in the full sentences being translated instead of specific words within the sentences. The translated words are then reconstructed to regenerate sentences.

3. Reconstruction Function

During the process of outputting translated text from `deep_translate`, an issue arose: duplicated punctations. This occurred because the text was sent to the API with punctuation still attached, and the API often returned the translated sentences with punctuation. Consequently, when reconstructing the sentences, both the original and API-return punctuation were preserved, leading to duplication. To address this issue, a dedicated reconstruction function `adjust_punctuation`, was implemented, as shown in Figure 7. This function ensures the translated text aligns with the original text's punctuation format.

```
def adjust_punctuation(original_text, translated_text):
    # Remove duplicated punctuation marks in translated_text
    translated_text = re.sub(r'([.!?])\1+', r'\1', translated_text)

    # Replace any '.' with '.' in original_text
    adjusted_translated_text = translated_text.replace('.', '.')
    # Replace any '?' with '?' in original_text
    adjusted_translated_text =
adjusted_translated_text.replace('?',
                                 '?')
    # Replace any '!' with '!' in original_text
```

```
10    adjusted_translated_text = adjusted_translated_text.replace('!',
                                 '!')
11    return adjusted_translated_text
```

Figure 7. Reconstruction of the punctuation for translated text

Despite that the punctuation adjusting function is only required when translating whole sentences as sentences were the ones with punctuation, it was decided to run all text through this function to ensure that all edge cases were covered where the word selected for translation happened to have punctuation included with it.

4. Game Configurations

As mentioned earlier, the entire program was created with modularity in mind. Thus, the inputs and outputs had to be both simple, clearly defined, tracible, and modifiable to a degree by the player.

The TM incorporates customizable parameters that adapt to the learner's preferences. The parameters include:

- Language Selection: Allowing the player to choose their target language.
- Difficulty Adjustment: There are 10 different difficulty levels in the game. The required level by the player is specified at the start of the game and accordingly the percentages and frequencies of words to be translated will be adjusted.

```
1 def translate_text(text, frequency, d):
2     if d == 0:
3         return translate_sentences(text, frequency)
4     else:
5         return translate_words(text,frequency)
```

Figure 8. The game setup function to toggle translation of words or sentences

As shown in Figure 8, the `translate_text` function argument `d` is used to decide whether to translate sentences or words within sentences. This adjustable behavior of the TM allows the translation module to integrate seamlessly with a variety of games or applications.

```
1 def translate(long_text, d, f):
2     frequency = random.randint(1, f)  # Randomly select the frequency of
translation
3
4     new_text = translate_text(long_text, frequency, d)
5     adjusted_text = adjust_punctuation(long_text, new_text)
6     return adjusted_text
```

Figure 9. Main driver function for the TM

Figure 9 shows the main driver function of the TM which handles all of the major function calls and allows interfacing with the GM and should be used by any other application or game that requires interfacing with the TM. Following the same design principles of modularity and code reusability, the function `translate(long_text, d, f)` allows programmers to easily adjust the parameters and control the inputs and outputs of the TM.

Implementation Challenges

1. Grammatical Structures

While the program has no issues with translating select words or full sentences, challenges arise with languages that have different grammatical structure, which makes it difficult to translate small segments accurately. Currently, the program is only capable of either blindly replacing single words or accurately translating full sentences. While this is great for the two extremes, it means that in between these stages, the system is rough at best.

2. API Limitations

While the translation library that is used allows for a wide selection of translation APIs and even AI-powered translators, there is no inherent way to avoid the timing issue. Since we were utilizing the free edition of the library, we had limitations on both the amount of text we can submit to the API for translation and limitations on the response time which can range from a few seconds to tens of seconds. The larger the requests sent the more delayed responses we get. This results in an obvious delay between when the game gets an input that needs to be translated and when the game finally outputs the correct translation. One way of solving or at least mitigating this time issue is by having the translation done inside of the program itself rather than making external API calls. This would also remove the need for the game to operate with an Internet connection. However, this also introduces more complexities as now the translation program would need a library of all of the translations in all of the languages as well as an account for grammar and sentence structures. This would not only massively complicate the coding of the program but also increase the size of the program significantly and may introduce its own bottlenecks. With all of that in mind, going the route of using APIs was the best choice not only in terms of complexity and ease of use, but also in terms of size and compatibility.

Future Work

1. Game Improvements

LangQuest, as a proof-of-concept application, demonstrates the integration of TM into a language-learning game. While the current version showcases its potential, there is significant room for improvements of precision, performance, and graphical design. To address the precision and accuracy of translated text, the use of a verified AI grammar checker to reorganize and refine translated sentences would ensure grammatical accuracy and natural flow in the target language. API limitations could also be solved by relying on subscription-based translation APIs which have a service agreement of being very responsive and have no limitations on the amount or frequency of translated text. Perhaps a more cost-effective solution would be a hybrid approach where the most used words of a language being translated natively, and the more obscure ones being handled by an API. Finally, graphic enhancements using a game development framework, such as `Pygame`, can create a more engaging and visually appealing experience for the young learners.

2. Game Testing

Following the implementation of the aforementioned game improvements, LangQuest can be deployed on a small scale in classrooms to assess effectiveness. The primary focus will be on higher level elementary to junior high students who possess at least a basic knowledge of the target language. It is in these scenarios that LangQuest is designed to work most effectively and will have the highest potential of making an impact. We will incorporate the learners feedback in our following iterations of the game.

Conclusion

In summary, this paper illustrates a simple yet innovative solution to language education in the form of a gamified language-learning program. This language learning program is meant to maximize engagement and entertainment at its core. Built upon modular coding building blocks that can be implemented into any other program, use cases are potentially endless. The Python-based codebase further supports ease of development and efficient deployment across various platforms including web-based applications. While the current demonstration program has its limitations, the concept holds significant promise. With further development, it could evolve into a comprehensive, gamified language-learning solution with the capability of changing the way languages are learned worldwide.

References

1. [1]“Google Translate - A Personal Interpreter on Your Phone or Computer,” translate.google.com. https://translate.google.com/about/?hl=en

2. [1]Lamija Huseinović, “The Effects of Gamification On Student Motivation And Achievement In Learning English As A Foreign Language In Higher Education,” MAP education and humanities, vol. 4, pp. 10–36, Jul. 2023, doi: https://doi.org/10.53880/2744-2373.2023.4.10.

3. D. López-Fernández, A. Gordillo, P. P. Alarcón and E. Tovar, "Comparing Traditional Teaching and Game Based Learning Using Teacher-Authored Games on Computer Science Education," in IEEE Transactions on Education, vol. 64, no. 4, pp. 367-373, Nov. 2021, from https://doi.org/10.1109/TE.2021.3057849

4. [1]“Welcome to deep_translator’s documentation! — deep_translator documentation,” deep-translator.readthedocs.io. https://deep-translator.readthedocs.io/en/latest/

COLBY E. KURTZ

Colby Kurtz is a sophomore student pursuing a BSc in Cyber Engineering at Houston Christian University interested in Cybersecurity, Mathematics, and Electrical Engineering.

MATTHEW Z. BLANCHARD

Matthew Blanchard is sophomore student pursuing a BSc in Cyber Engineering at Houston Christian University interested in Cybersecurity, Computer Science, and Computer Hardware Engineering