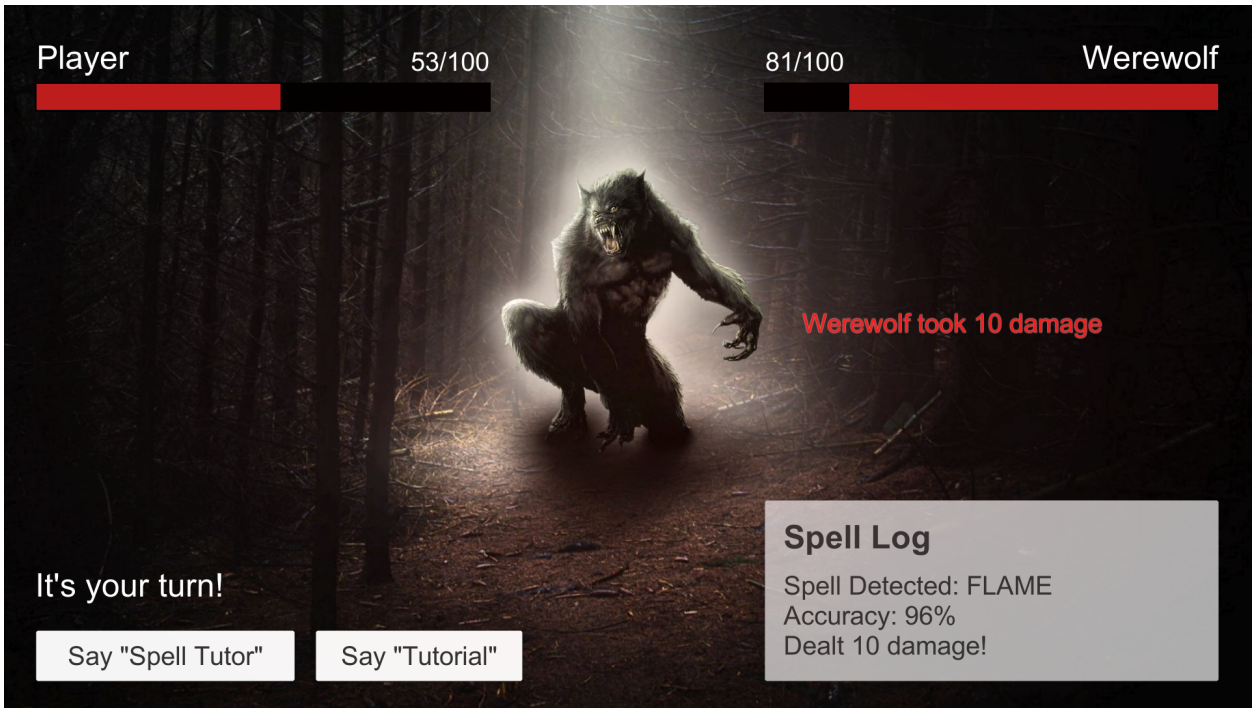

Kalpana: A Multimodal Adventure Spellcasting Game

Sohini Kar, Lilian Luong

We built a D&D-inspired multimodal spellcasting game focused on teaching users how to perform various spells using both speech and gesture recognition, and in turn, use these spells to attack and defeat a variety of enemies. Through a game UI full of features, infinite enemies, and exciting spells, Kalpana creates an immersive experience that encourages users to perfect their spellcasting. Our software includes Python 3.8, the Leap Motion Tracking Software, LeapSDK, SpeechRecognition, Numpy, and Flask. For the game UI, we used Unity. Our hardware included the Leap and a microphone.

Kalpana: A Multimodal Spellcasting Adventure Game

Sohini Kar (ID: 919186318), Lilian Luong (ID: 910850097)

1. Introduction

1.1. Motivation

The real world can sometimes feel boring; people often look to combine imagination and excitement through role-playing games filled with foreign lands, grand adventures, and wondrous magic. Tabletop roleplaying games were developed with the intention of supporting these desires, with Dungeons Dragons (DD) standing as the most well-known of these games. The game involves acting as a specific character defined by a character sheet and performing various spells, whose accuracy and power are randomized based on dice rolls.

However, while immersive materials and excellent acting can help players imagine themselves as part of a story, the game always inevitably returns to its dice and pen-and-paper mechanics. When attacking a horde of goblins merely requires marking off a spell slot on a character sheet, and persuading a nobleman to offer his home as sanctuary against vampires takes only a lucky dice roll, the fantasy fades. During our own DD experiences, we have noticed how fellow players have lost the motivation to act like their characters within the story, instead just weaponizing the numbers on their character sheets to "win the game". It has become clear that the tabletop medium, using only numbers and lucky dice rolls, is not enough to fully support the fantasy world, adventure, and - most of all - actions that DD aims to create. Our goal in this project is to use the power of multimodal interfaces to craft magical experiences in a more immersive way.

2. Related Work

Our project focuses on generic gesture and speed recognition, sentiment analysis on speech, and creating a multimodal flow integrating gesture and speech commands together. The functionality for gesture and speech recognition is key in developing the ability to recognize and execute spells, and sentiment recognition can bring in a deeper experience through encouraging true roleplay.

Gesture recognition has been explored and developed to create a rapidly processing stream of information. Lovell [5] describes a design for an inertial sensor system that may be used for gesture recognition. These data streams can be quickly processed to detect specific gestures. Another study by Yin [6] uses a hierarchical Hidden Markov model to create a real-time continuous gesture recognition application with the capability to add new gestures through only 3-6 repetitions and a small training time. This is highly applicable to our project, as we are interested in developing a gesture recognition system that may run quickly and accurately. Additionally, the Gesture Recognition Toolkit [4] was developed to make gesture recognition using machine learning more accessible. This can be adapted for our project to learn about potential functions and methods for signal processing and classification algorithms.

A key study by Chang [3] focuses on implementing a Hidden Markov model for the purpose of gesture recognition, with the goal of using fluid and natural movements to control a Powerpoint presentation. This model also uses speech commands to give the user an additional level of precision and control over the system, similar to our system.

Badjatiya et al. [1] worked on understanding hate speech and running sentiment analysis for the purpose of detecting hate speech in Twitter tweets. Natural language processing is a difficult task, and one of our key goals is parsing sentiment from speech to aid in calculating the power or type of spells casted within the game, especially since the role-play aspect is key for our project. Cachola et al. [2] also performed a study on detecting vulgarity from speech, and they explain how vulgar words are commonly part of popular sentiment and emotion analysis lexicons.

Finally, there has been work developing realistic spell-casting technology in relation to Harry Potter, which uses wands to perform spells. An example of one such technology is displayed in Figure 1. However, we are focused on a D&D-related use case, with a separate focus on roleplay and lacking wands.

This project aims to combine gesture recognition, speech parsing, and sentiment analysis to effectively integrate different modes of communication into one game focused on encouraging roleplay and accurate spellcasting. While there have not been efforts prior combining all three of these, there are numerous studies individually analyzing all of them as well as combining gestures and speech.

Figure 1. Diagram of wand motions and incantations associated with spells in the Harry Potter series

2.1. Collaboration Statement

For the preliminary version, Lilian implemented the gesture recognition and Sohini implemented the speech recognition, and both worked on combining the systems to run asynchronously. Lilian then set up Flask and connected to Unity, while Sohini implemented the game mechanics and logic as well as a command-line version of the game. To create the main game, Lilian worked on translating all of the code for the game logic to C# for use in Unity as well as designing and building the different game states and interface components, and Sohini worked on the art and assets for the game. Finally, both were responsible for testing and polishing the game.

2.2. Code

The code for our project is available at: https://github.com/lilianluong/multimodal-fantasy-game

3. System Overview

A diagram of system components, inputs, and outputs can be seen in Figure 2.

3.1. Functionality

We created a digital game which preserves the core principles and mechanics of traditional fantasy role-playing games: encounters and battles ranging from social to combative, and a large variety of (often magical) abilities that can be used to solve them. To take any action, however, players must perform it themselves using speech and gestures instead of simply rolling dice, where the quality of their spellcasting directly impacts their in-game performance. They thus insert themselves into a more immersive adventure and experience the hero's growth and improvement - formerly achieved through generic number-crunching - as their own journey.

The system combines gesture recognition - using input from a Leap Motion Controller - with speech recognition to offer the player a multimodal interface for casting spells. After learning the incantation and gesture needed for a spell, the player can use it in battles against monsters or to overcome obstacles in their path. The cast's timing and precision affects how powerful the spell becomes, encouraging further immersion.

3.2. Example Input and Response

The player has just entered the game and decides to test the Spell Tutor, a gloomy and medieval tune looping gently in the background. A dialogue on the screen explains that to cast the spell, the player must say the spell's name, "Flame", while using their hand to perform the gesture shown in the animation in the center of the screen. The player tries casting the spell: they move their hand over the sensor on the table face down, saying "Flame" at the same time. A red glow appears on the screen, indicating that the spellcast succeeded.

Having learned the "Flame" spell, the player proceeds to an Adventure, a fight against a snarling werewolf. A health bar displayed over the werewolf's head is full; a similar bar shows the adventurer's health at the bottom of the screen. The player immediately performs their newly learned spell, repeating the gesture and shouting "Flame"; a red glow appears and strikes the werewolf. Its health bar reduces by a mere quarter, and a small visual indicator shows the damage dealt. In the lower right box appears the spell detected and the accuracy of the spellcasting. The werewolf proceeds to take its turn in combat, clawing at the adventurer for half their health. The player considers attacking again, but decides that they're in danger of losing all their health and instead casts a different spell: while waving one hand in a circle, they call out the spell "Cure": the player's health bar is restored to its full capacity. After withstanding another attack by the werewolf, the player attempts "Flame" again, this time making sure to emphasize the word. It's more successful: the new attack causes significantly more damage than the first and destroys the remainder of the werewolf's health, earning the player a victory.

4. Implementation

Figure 2 shows the layers and components of our system. It can be separated into a backend developed using Python 3.8, which is responsible for processing input and classifying spells, and a game frontend implemented in Unity with C#. The Unity game uses HTTP requests (using the UnityWebRequest library) to communicate with a Flask API in order to prompt and access the results of the spell recognition system.

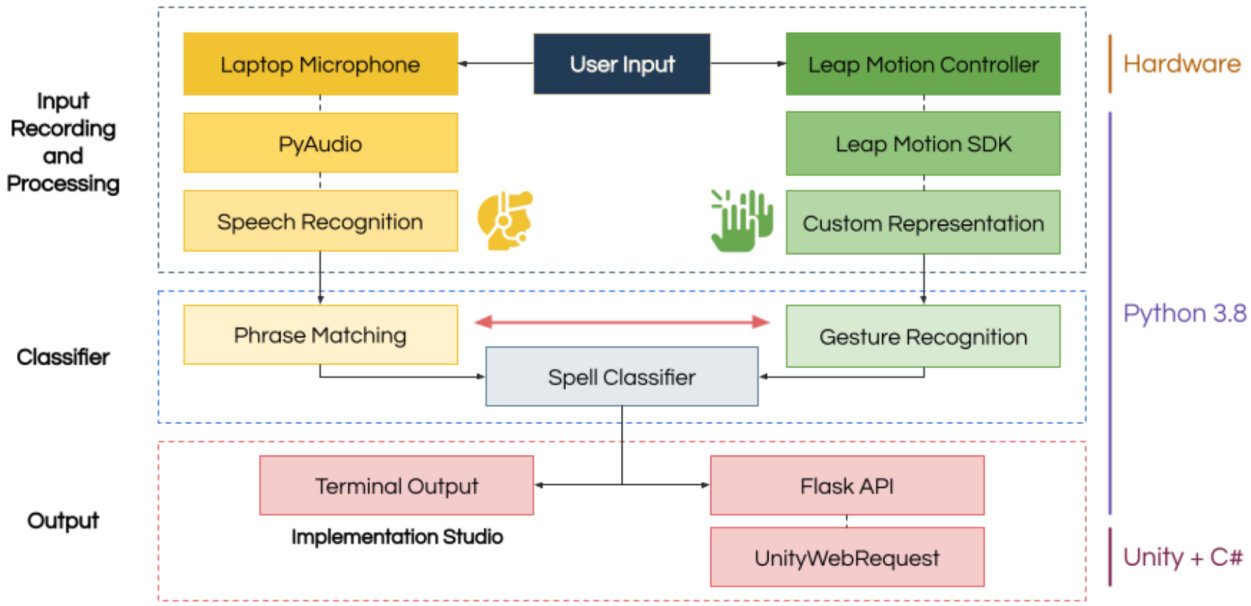

Figure 2. System diagram showing components and inputs of game system

4.1. Spell Recognition Backend

4.1.1 Structure

The Python backend runs as a Flask app in the localhost server, which the Unity frontend interacts with using HTTP requests. The app itself runs three parallel subprocesses: the Flask app itself, a main classification loop, and a speech recording loop.

The classification loop, when triggered, saves hand position frames from the Leap Motion Controller while waiting for the speech recording loop to return a result. It then separately classifies both forms of input, before combining the results of the classifiers to determine the spell that should be returned, if any such match exists. The process shares its output with the Flask app, so that a GET request is responded to with the results of the classifier, or a timer state if the classifier is running. POST requests are used to trigger the classifier loop.

4.1.2 Classifiers

The spell recognition system combines the results of independent speech and gesture recognizers to identify the casted spell. The speech recognition component selects a set of candidate spells, and the candidate that is scored best by the gesture classifier is returned.

Audio input from the laptop microphone is converted into text using Python's SpeechRecognition library to access the Google Web Speech API. Each spell in the system's database is marked as a valid candidate if its incantation is a substring of the returned transcript.

The gesture classifier uses image-based template matching similar to the method used in Mini Project 1 (early sketch understanding). For each gesture template in the database, a window of the same time length is slid along a sequence of recorded hand positions. The 2D points in each window are compared against the point set of the template using the Modified Hausdorff distance (MHD) metric:

$$h_{mod}(A, B) = \frac{1}{N_a} \sum_{a \in A} \min_{b \in B} ||a - b||$$

Both point sets are normalized by horizontal and vertical scaling. The similarity score of each spell is its minimum MHD obtained across all windows.

4.2. Game Interface

The Unity game implements the general gameplay logic as well as the user-facing interface. It is separated into four distinct scenes: the title screen, an instructions screen, a Spell Tutor, and the Adventure. The title is accessed upon starting the game and transitions automatically after a fixed number of seconds. Meanwhile, the instructions screen provides a written tutorial explaining how the game should be navigated and played. Either clickable buttons or speech commands can be used to switch between the tutorial, the Spell Tutor, and the Adventure screens.

The game operates on a continuous update loop. Based on an internal finite state machine, it communicates with the Python spell recognition backend by repeatedly polling the API for any detected speech commands or casted spells. In

the Spell Tutor, depending on which spell name is clicked or spoken, an animation demonstrates the required gesture while the user can make spellcast attempts of their own. In the Adventure, the player can apply these spells in turn-based combat against enemy monsters.

4.3. Turn-based Gameplay

Combat encounters use a turn-based spell system, where the player and enemy alternately take actions. Enemies are randomly generated and possess statistics such as attack damage and maximum health. Based on these statistics as well as those of the player, actions can either deal damage (decreasing opponent health), heal (increasing own health), or a combination of both. When the player casts a spell, as returned by the Python spell recognizer, the spell that was cast as well as the similarity score obtained by the gesture classifier determines its effects.

5. User Studies

5.1. Implementation Feedback

Our first main stage of user testing was in the Implementation Studio, where we had a player try a command-line version of the game. Here, most of the gameplay mechanics were already implemented but the graphics and instructions were all text-based in the terminal itself. Our user gave some key feedback, first criticizing how we were giving instructions. Several important points were missing, such as the user's hand having to be open and face down while doing the gesture, the gesture and speech should be simultaneous, and the game mechanics themselves were not very well balanced. He specifically said to add more variety to the moves and to make the spell names more different, since heal and shield were confused by the system. We prioritized this feedback entering our final version.

5.2. Project Feedback

For our final user tests, we had users play through various adventures and test the extra features, such as the Spell Tutor. Although improving the instructions for the users was a key task we focused on for this version, the players specified that feedback from the system was still lacking. Specifically, they wanted more feedback on why spells would sometimes not be recognized, and would have preferred to see the speech detected. Additionally, we added more spells with different effects, but did not teach the users these effects. In turn, the players mentioned understanding what different spells did was a point of confusion. The impact of spellcasting accuracy was not obvious as well, and users did not realize that it impacted the amount of damage done. Finally, players found it difficult to build a strategy within the game. One phenomena that stood out to us was that different players would be able to perform one spell consistently and struggle with another during the Spell Tutor, and then during an adventure, they would exclusively use the one they had become comfortable with instead of combining them and developing a strategy.

6. Discussion

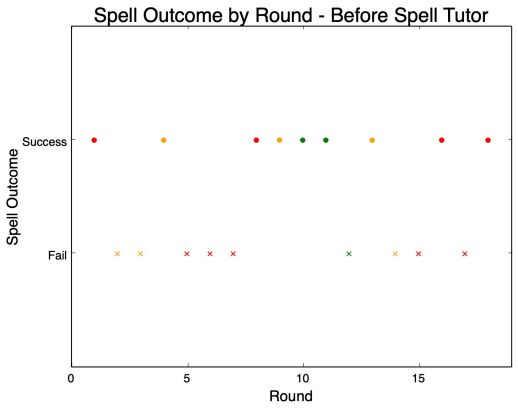

Figure 3. Diagram of spell results by round before practicing using the spell tutor

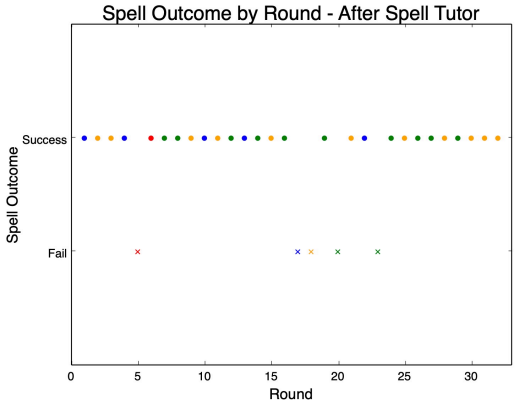

Figure 4. Diagram of spell results by round after practicing using the spell tutor

6.1. Performance

Our system generally worked well at recognizing the various spells, consistently outputting the correct results and allowing for seamless gameplay. Spells that sounded similar or had similar gestures were sometimes confused by the system, due to the nature of the recognition, so we focused on creating spells that were as different as possible in these respects. With the final system, we ran analytics on our user study to identify how well the game detected the correct spells, both before and after the player practiced spells in the Spell Tutor.

Before the user practiced spells in the spell tutor, in one adventure the player successfully cast a spell 9 out of 18 rounds, yielding a rate of 50%. A diagram of these results by round is shown in Figure 3, where the point colors correspond to the attempted spell. Specifically, "cure" is displayed as a green point, "flame" is red, "lightning" is orange, and "leech" is blue.

After the Spell Tutor, our user successfully cast a spell 27 out of 32 rounds, yielding a rate of 84.38%. This provided us with a baseline accuracy of the system and demonstrated that a large limiting factor was the performance and understanding of the player themselves on how to perform the spells. Another diagram of these results by the round is shown in Figure 4, with the same colors corresponding to spells.

6.2. Successes

Building the speech recognition system and gesture recognition system themselves was successful, especially since we focused on modularity while developing the game and we were able to use much of what we had learned in the class to build up both recognition systems. Additionally, creating the game mechanics and rules was interesting and exciting. We knew we wanted to have a Spell Tutor to teach players the spells ahead of time, as well as create different enemies with their own advantages to keep the game exciting and interesting. By adapting a simple turn-based combat system, we were able to focus less on teaching players about how to battle and more on how to improve their spellcasting. Users reported being thoroughly immersed, and would go through several rounds of the game without us prompting them to continue. Choosing to have new enemies spawn immediately after the player beat one ensured players were caught up in the fast-paced nature of the game and stayed engrossed in mastering spells and defeating the opponents.

6.3. Challenges and Modifications

Our first major roadblock was in combining the speech and gesture recognition systems. Because our proposed method of spellcasting required players to both speak the incantation and perform the gesture at the same time, we looked into methods on how to run the functions asynchronously and combine their outputs to detect one spell and its accuracy. Originally, we intended to provide some feedback to the user on how to improve their spellcasting but this proved to be difficult on the gesture side, since it felt out of the scope of the project to map the player's movements to the intended gesture, understand which part of the gesture this was, and translate to the user how to fix the spell. However, it was an easier task to provide the transcript of the detected speech to the user to allow them to see any mistakes. In the end, we chose to modify the project and remove gesture feedback. Gesture feedback was an interesting next step our system turned out not equipped to handle. However, it would be exciting to implement and provide very concrete feedback to the player.

Another challenge was simultaneously starting our game system and running the Flask server. The game system was necessary to allow the game mechanics to run smoothly, and the Flask server was necessary to communicate with the Unity game UI. Unfortunately, we realized we could not run both asynchronously on the same processor, as both were blocking. We turned to multiprocessing to allow them to run together, which ended up working well and did not result in many modifications.

A separate modification moving from the implementation studio to the project studio was to remove several of the character attributes, such as intelligence, strength, and charisma. We were using these attributes to impact the effects of different spells, but after receiving feedback from the command-line implementation that we needed to focus more on balancing the game, we decided to simplify the spells slightly to focus more on the spellcasting itself and the game's simplicity. This is another feature we would like to bring back, as it creates a layer of personalization and individuality to each character and encourages immersion.

6.4. Interesting Failure

One failure we faced was not on the technical side, but on the gameplay itself. Using the quality of spellcasting in the gameplay itself to motivate players to truly perfect and focus on their spells was a key draw that we emphasized in our system design. We wanted to provide the player with feedback and incentives to immerse themselves in the game, so using the recognition accuracy as a metric for spellcasting quality, which would in turn either strengthen or weaken the spell's effects accordingly, was very unique to our system and hard to replicate outside of this type of multimodal gameplay.

However, after we completed our user studies, we learned that our players did not realize that accuracy was playing a role in how strong the spells they were casting, and thus did not create the immersive effect we anticipated. This was an interesting failure as it demonstrated how we, as the developers, assumed users would understand and interpret the displayed metric as intended, but the users were so engrossed in the fast-paced gameplay that they did not dwell on the accuracy metric. We learned not to assume user prior knowledge and to create a clearer and more informative UI, and we did add a warning about the effect of spellcasting accuracy to the preliminary instructions.

7. Future Work

Looking into the future, there are a number of changes and additional features we would like to work on imple-

menting. They are either features that we could not achieve within the original scope of the project due to time or resource limitations, or changes brainstormed based on feedback from user studies.

7.1. Further Spell Development and Balancing

One of the main pieces of feedback we received from user testing focused on balancing the game mechanics to make it more fun and challenging from a strategic standpoint. With all spell effects centered around either damaging or healing, our players typically determined that one or two spells were superior choices and used them repeatedly without ever finding a need to try the other spells. This made the game repetitive and tiring to play after a few levels.

In addition to tweaking the numbers and effects of our existing spells so that none of them are obviously "better" than others, we also have some ideas for additional types of spells that can introduce more variety to Kalpana's gameplay. For example, spells might temporarily weaken the opponent, have a chance of stunning them, or create a shield to reduce the effect of the next attack.

A distant stretch goal of our initial project proposal that we did not end up achieving would also help to make the game more interesting and complex. This was to add spells where the speech recognition aspect is more freeform compared to our current spells, where the only task is to speak the incantation. Instead, spells might rely on the user coming up with their own phrase tailored to the situation, with variable effects based on word choice, content-based sentiment analysis, or emotion detection. This feature would rely on more complex natural language processing, but due to the modularity of the system could be simply built on top of the speech recognition component of the Python backend.

7.2. Better Spellcasting Feedback

As discussed in Sections 5.2 and 6.4, players were often confused when testing our game because it didn't make it clear that the accuracy of their spellcasts had an impact on the strength of their effects. One future goal for developing the game is to make this more clear, not only in the instructions but also through UI changes. For example, we can add sound effects and scale the lighting flare effect when a spell is cast so that more powerful spells are louder and brighter. To make it obvious what each spell does, we can add a written description for each one in the Spell Tutor, that can also describe how the effect is influenced by spellcasting accuracy.

Another aspect of this idea is improving the feedback as to why a player's spell scored low, so the player can understand how to improve their spellcasting. Spellcasting accuracy currently depends on three main factors: spoken incantation, gesture shape, and gesture timing. For the first, we can display detected speech on-screen after each turn. For the second, we want to have an animation similar to the Spell Tutor, which traces the gesture done by the player so it can be compared to the template. This would also help to provide feedback on timing, but we can also modify the gesture classification algorithm. Instead of computing MHD for a fixed-size sliding window, we can instead try different sizes of windows to determine if the gesture matches better when shortened or lengthened, and suggest accordingly for the player to slow down or speed up their hand gesture.

7.3. Game Progression and Difficulty

In terms of overall game design and gameplay, we currently provide an infinite series of randomly generated enemies for the player to fight against. However, the combat encounters are ultimately the same difficulty throughout, and beyond beating an enemy, there is no way to "beat" the game. To make Kalpana into a more challenging and engaging game, we want the level of difficulty to increase as the player progresses through the game and improves their spellcasting skill.

Scaling difficulty can be implemented by changing the enemy statistics so that they have more health, damage, and other attributes as more enemies are defeated. However, the battle still needs to be fair. To make the game simulate an actual adventure, we would like a larger catalogue of spells where individual spells are unlocked throughout the game. This would incentivize the player to learn new spells instead of relying on just a couple of them, and provide room for more powerful and complex spells to be added to the game without upsetting its balance.

8. Tool Descriptions

The table of tools used in this system is attached at the end of this paper as an appendix.

9. Conclusion

We were able to build a full, robust, and immersive spellcasting adventure game that used multimodal input to craft an engrossing and exciting experience. By combining gesture and speech recognition, we successfully taught our players how to use different spells to take down their enemies in the game.

Through various rounds of user testing and iterating on the game design, Kalpana has many different features packaged into one immersive game. By incorporating feedback from playtesters of the game and our peers in the classroom watching our presentations, we are excited to present this game and continue developing it.

References

[1] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. *CoRR*, abs/1706.00188, 2017. 1

[2] Isabel Cachola, Eric Holgate, Daniel Preoţiuc-Pietro, and Junyi Jessy Li. Expressively vulgar: The socio-dynamics of vulgarity and its effects on sentiment analysis in social media. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2927–2938, Santa Fe, New Mexico, USA, Aug. 2018. Association for Computational Linguistics. 1

[3] Stephen M. Chang. *Using gesture recognition to control PowerPoint using the Microsoft Kinect*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2013. 1

[4] Nicholas Gillian and Joseph A. Paradiso. The gesture recognition toolkit. *J. Mach. Learn. Res.*, 15(1):3483–3487, jan 2014. 1

[5] Steven Daniel Lovell. *A system for real-time gesture recognition and classification of Coordinated Motion*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2005. 1

[6] Ying Yin. *Real-time continuous gesture recognition for natural multimodal interaction*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2014. 1

Package/Tool/Library	Where is it available? (eg url)	What does it do?	How well did it work?	Did it work out of the box? If not, what did you have to do to use it?
Python 3.8 x86-64	https://www.python.org/downloads/release/python-380/	Python programming language	As expected	Yes
Unity	https://store.unity.com/#plans-individual	2D and 3D game engine	As expected	Yes
Leap Motion Tracking Software v2.3.1	https://developer.leapmotion.com/releases	Runs the Leap Motion Controller on computer	Well, although the Leap Motion Controller tracked some hand poses better than others.	Mostly. Needed to check some settings (e.g. “Allow Background Apps”) in the Leap Motion Control Panel.
Leap Motion SDK	https://developer-archive.leapmotion.com/get-started (SDK) https://support.leapmotion.com/hc/en-us/articles/360004362237 (instructions for generating a wrapper) https://github.com/Cipulot/Leap-Motion-Python-3 (third-party generated wrappers for Python 3.7+, untested)	Allows us to access input data from the Leap Motion Controller using Python.	It worked well once it was set up, but setup took a lot of additional, unexpected effort.	No, since the Leap Motion SDK is only available for Python 2.7. Since we wanted to use Python 3.8, we followed instructions to generate a custom wrapper for our Python version, which was tricky to make work. We would recommend using a pre-made wrapper as linked, instead of doing as we did.

SpeechRecognition library	https://pypi.org/project/SpeechRecognition/	Python library for accessing various speech recognition engines/ APIs	There was more latency than expected which we had to design our system to work around. The accuracy of speech-to-text was good in a quiet environment.	Yes
JSON .NET for Unity	https://assetstore.unity.com/packages/tools/input-management/json-net-for-unity-11347	Unity package for processing JSON data.	Worked well, though there is little documentation.	The documentation isn't clear you need to add "using Newtonsoft.Json" to the code, and the package isn't maintained for current versions. There were a few issues getting the package to work.
Flask	https://flask.palletsprojects.com/en/2.1.x/	Microframework for building web apps (in our case, a web API) in Python.	Worked well.	Yes
NumPy	https://numpy.org/install/	Python library for efficient array manipulation	As expected	Yes

Your project name: Kalpana: A Multimodal Spellcasting Adventure Game

Project author(s): Sohini Kar, Lilian Luong