# CS423 Natural User Interaction

Course Project Final Paper

***Group Members:***

Nicholas Rin-Laures Kent Lizardo

Daniel Rodriguez Victor Colome

***Section:*** Undergraduate/Graduate

# Bard

|  |
| --- |

# 

# Introduction*.*

The objective of this project was to recreate a classic game that utilizes speech as a form of natural user interaction. Our team chose to develop a project known as Bard, which is a reinterpretation of the classic dungeon-crawling computer game, Rogue.

In Rogue, players navigate procedurally generated dungeons in search of a special amulet while facing various enemies and traps. While the original game relies on keyboard controls, we believed that incorporating speech input would significantly enhance accessibility and gameplay flexibility. This addition aimed to allow users to respond more swiftly to challenging scenarios, eliminating the need to memorize complex controls and facilitating a seamless connection between the player's intentions and the game itself.

One relevant piece of literature we discovered is a research paper titled “Voice Typing: A New Speech Interaction Model for Dictation on Touchscreen Devices,” authored by Anuj Kumar, Tim Paek, and Bongshin Lee. The study discusses improvements in speech-to-text models to make them faster and more user-friendly by recognizing spoken words in real time. This advancement aims to speed up transcription and simplify error correction, aligning well with our project’s goal of enhancing alignment in human-computer interactions.

# Iterative Design and Development Process

## Design Process

Our project consisted of two primary components: the game and the interpreter. The design process for the game was largely informed by our team’s previous game development experience, which allowed us to deconstruct the game into its component parts that could be developed either sequentially or concurrently.

The interpreter's design was finalized following a formative user study in which we conducted eight informal interviews. The purpose of these interviews was twofold: to identify the potential speech commands users might employ and to understand the features they would expect from a voice interface. After analyzing the results, we created personas based on the identified clusters and developed scenarios for each persona to explore how they might interact with the game.

Interviewees were asked about their experiences with both voice recognition and gaming. They were then presented with before-and-after screenshots for various tasks and were asked to describe the commands they would use to transition from the first screenshot to the second. While it was expected that participants would utilize different commands, some also attempted sequences of commands. Variations in familiarity with both voice recognition technology and gaming contributed to the diversity in the commands used as well.

Generally, the interpreter’s design aimed to avoid prescribing specific commands, instead responding only to simple directions. However, when this approach proved impractical or when implementing sequences of commands was necessary, we incorporated as many commands for those actions as were documented or could reasonably be conceived.

Upon reviewing the interviews, we discovered that our responses fell broadly into main categories: individuals who are familiar with voice recognition technology but not our specific game style, and those who are familiar with our retro game but not so much with voice recognition. For each group, we included a scenario to illustrate how they might interact with the game.

* Example: Steve, a student who enjoys gaming but isn't old enough to be familiar with Rogue. He is adept at using voice controls on his phone, but he has never played a NUI game before. What Steve seeks in the NUI game is an experience he can easily pick up and play using voice commands, without having to grapple with outdated controls.
* Steve’s Scenario: While at home, Steve attempts to play the game on his computer. Lacking knowledge of the original game and being accustomed to using voice assistants, he issues more complex speech commands that involve detailed sequences of movement and phrases these commands in terms of objectives rather than typical game behaviors.

In light of each member’s experience in various toolsets, we opted to use Python, as it was a language we were familiar with and had shared intermediate proficiency in. It also allowed us to iterate through feature development and testing at an efficient rate. Having some knowledge of procedural generation, Kent chose to utilize binary space partitioning for the dungeon's level generation, which satisfied our requirements for creating varied rooms with maze-like traversal. For implementation details, please refer to the second entry of the bibliography.

We also employed several object-oriented design patterns, including the command pattern for entity actions and the adapter pattern to integrate various speech recognizers to work with our program. All members of the team have informal gaming experience, which influenced many of our feature development decisions.

During the in-class user feedback session on 10/10/2024, our team received key insights regarding the lack of clarity of visual components, particularly concerning error messages from the speech recognizer. Another significant concern was the improper functionality of the speech recognizer in environments with moderate to heavy background noise. Additionally, testers noted the awkwardness of the controls, specifically the need to press the spacebar to activate the speech recognizer's listening phase and the lack of clear indications that both keyboard and microphone prompts were available to users.

Many of our testers faced considerable difficulty in understanding how to engage the program, which was exacerbated by the speech recognizer's inefficiency in the noisy classroom setting. To rectify these issues, we modified the error messages displayed in the game for improved readability and expanded the list of supported commands to enhance user engagement. Furthermore, we eliminated the requirement to press the spacebar, allowing the recognizer to continuously listen as the program runs, enabling users to interact with the game more seamlessly.

Our team explored alternative speech recognition models like PocketSphinx and DeepSpeech; however, we determined that our current model at the time, Google Cloud Speech, remained the most effective option. Adjustments were made to the recognizer to accommodate ambient noise, though we still encountered challenges with stalls in noisier environments.

The notes from our in-class user feedback session can be found here:

* <https://docs.google.com/document/d/1FKMu9Ol5Jcl22nfrTlQ23evHGdOKgzf-HMW1IqGqNm8/edit?tab=t.0>

Our participant sign-up sheet can be found here:

* <https://docs.google.com/spreadsheets/d/1UgID0aypR2rYlblS3VRtopkiQQO_yZPXNdbzYTT1P2s/edit?resourcekey=&gid=902464557#gid=902464557>

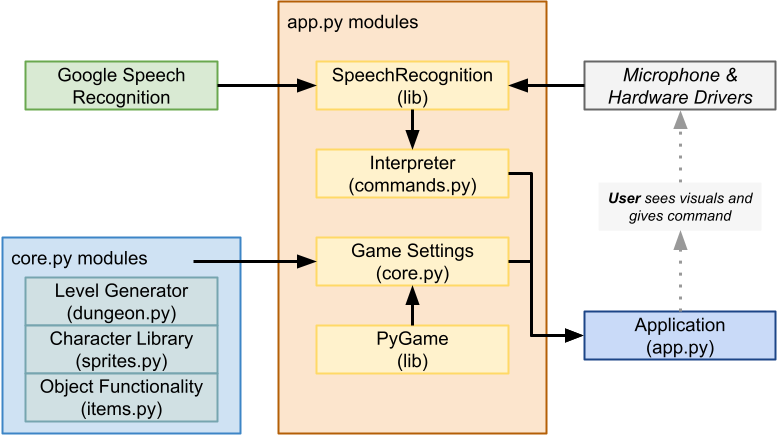
The key NUI tasks that we planned to support in our program were:

* Command 1: Move the player character.
  + **What Action**: Say a direction (Up, Down, Left, Right); Say a direction with a number (Up three); Say a sequence of directions (Up then left).
  + **Rationale**: Users familiar with voice controls expect the efficacy of commanding a sequence of simple movements.
  + **Possible Ways**: Any appropriate verb is recognized: “Move,” “Go,” “Walk,” etc.
  + **Feedback**: The player's sprite will move to the directed destination.
* Command 2: Pick up a game item.
  + **What Action**: Say “Pick up (item)” or a similar command.
  + **Rationale**: Needed to make progress in the game.
  + **Possible Ways**: Commands include “Pick up,” “Take,” “Target,” etc.
  + **Feedback**: The object is removed from the level and added to an inventory GUI element.
* Command 3: Attack an enemy.
  + **What Action**: Say “Attack” or a similar command.
  + **Rationale**: Users familiar with games will expect to be able to protect themselves from enemies.
  + **Possible Ways**: Commands include “Attack,” “Target,” “Kill,” “Fight,” etc.
  + **Feedback**: The enemy’s sprite is removed from the level.
* Command 4: Open inventory and use items.
  + **What Action**: Say “Open inventory” and “Use potion” or a similar command.
  + **Rationale**: Users familiar with games will expect to be able to use items picked up during the game.
  + **Possible Ways**: Commands include “Open inventory,” “Inventory,” “Use potion,” “drink potion,” etc.
  + **Feedback**: The item is removed from inventory. If a potion is used, the player’s health increases.

## Developing the Prototype Application

To summarize our development process, our group selected a NUI to integrate into our project and drafted a proposal outlining our planned development timeline, the APIs we intended to use, and the specific tasks we aimed to support through the NUI. Following the project’s approval, we conducted interviews to create user personas that would help us address additional user needs, and to identify which commands the game should support. As development progressed, we collaborated with three individuals for user testing, through which we identified issues that needed to be addressed in future iterations.

Our team utilized GitHub to host our repository and adopted a Kanban approach for lean development. We created a Kanban board within Github and assigned tasks to each team member on a weekly basis. Throughout the development process, our repository used the ‘main’ branch for major milestones and other feature branches such as ‘parallel’, or ‘whisper’ for implementing significant changes. These were later merged into the main branch once extensive testing was completed and once it was confirmed that they would not introduce any issues with the existing code.



*\*A diagram depicting the different frameworks and parts of our application as well as the processing links between them.*

The contributions from each member of our team was divided as follows:

* **Nick**
  + Conducted voice command research and design
  + Contributed to project report documentation.
  + Crafted the majority of informal interview forms.
  + Researched target demographic and consulted on the design of NUI features using class and instructor materials.
  + Created the presentations for the mid-term and final prototype demos.
* **Victor:** 
  + Established a repository and Kanban board for the development process.
  + Monitored deliverables and contributed to report documentation & presentations.
  + Researched and evaluated various speech recognition models to determine the most accurate option.
  + Conducted user studies as a note-taker and facilitator during sessions.
  + Enhanced the readability of text feedback.
* **Kent:** 
  + Built and designed the rogue game engine and basic app structure in Pygame.
  + Made a basic entity system and command framework for others to extend.
  + Added hooks for the recognizer and user to act on the game.
  + Constructed the recognizer to be callback-based to run passively alongside the game window.
  + Implemented dungeon generation.
  + Created interaction hint system and damage animation.
* **Daniel:**
  + Designed the program’s recognizer and interpreter.
  + Extended the program’s structure with the implementations of commands and command parsing.
  + Added items and entities such as enemies and behaviors for each.
  + Developed the inventory system.
  + Added coordinate movement using chess notation.
  + Implemented A\* path-finding algorithm for coordinate movement.

## System Architecture

Hardware sensors and driver software were managed externally by the system's default microphone. When voice commands were detected, the SpeechRecognition module converted these commands into text format within the application. The interpreter then processed this text and interacted with the game to execute the user's intended actions. Consequently, the game would update the screen, allowing the user to see the results.

## Code Modules

* **app.py:** Handles the main execution and usage of all interconnected parts of the program.
  + **Classes**
    - **SrModel:** Enum class used to select which Speech Recognition model to use**.**
  + **Methods**
    - **player\_decide:** Selects the playable player entity and executes an action command, then advances a step in the game.
    - **parse\_and\_execute\_command:** Splits an input into parsable sentences and executes commands contained within.
    - **split\_command:** Splits a command into “sentences” based on a list of connectors.
    - **get\_neighbors:** Gets all neighboring walkable tiles for A\*.
    - **manhattan\_distance:** Calculates manhattan distance between two grid positions.
    - **calculate\_path:** Generates a list of tiles to “walk” to a specific tile.
    - **grid\_to\_draw**: Converts grid position to a draw position.
    - **view\_grid\_to\_draw:** Converts a draw position to a grid position.
    - **render\_chess\_coordinates:** Render pipeline function that displays chess coordinates on all walkable tiles.
    - **get\_chess\_notation:** Converts grid position to chess notation string.
    - **parse\_chess\_notation:** Attempts to parse an input string to chess notation.
    - **render\_game:** Main render pipeline function blitting contents of Game to game\_screen surface.
    - **render\_health\_bar:** Render pipeline function to display the player's health bar.
    - **render\_inventory:** Render pipeline function to display detailed information about the player inventory.
    - **render\_text:** Changes the main text display surface to show a string.
    - **splash\_text**: Renders to the whole screen a black screen with text. Used outside of normal rendering loop like on startup.
    - **recognize\_data:** Converts PyAudio speech data to string using the configured recognizer.
    - **on\_listener\_heard:** A callback function used for passive phrase listening.
    - **move\_until\_obstacle\_step:** Moves player until it hits an object.
    - **move\_player\_step:** Moves the player by a single step.
    - **process\_player\_action:** Used to queue actions with a delay. (Deprecated).
    - **main\_loop:** The main loop of the program. Calls event handling and rendering with a fixed timestep.
* **commands.py:** A list of constant definitions linking terms to their Action counterparts,
* **core.py:** Library containing interface functions and classes to run a roguelike engine.
  + **Classes**
    - **Game:** Instance of an entire Rogue game.
    - **Entity:** Representation of any game object meant to be interacted with.
    - **Character:** Subclass of Entity representing a living creature.
    - **Door**: Subclass of Entity, openable wooden door.
    - **Stairs:** Subclass of Entity that represents the stair to complete the level.
    - **LockedDoor:** Subclass of Entity similar to a door but requires a key to open.
    - **ItemEntity:** Subclass of Entity that represents an Item on the ground. Connected to an Item instance.
    - **Player:** Subclass of Character that represents the player character. Not inherently controlled, but contains the sprite for the player.
    - **Slime:** Subclass of Character that represents an enemy slime.
    - **Skeleton:** Subclass of Character that represents an enemy skeleton.
    - **EntityAction:** Action bound to an entity.
    - **WaitAction**: EntityAction that does nothing.
    - **TeleportAction:** EntityAction that moves the entity to a specific position.
    - **MoveAction:** EntityAction that moves the entity via a change in position.
    - **MoveUntilObstacleAction:** EntityAction that moves an entity until it collides with another wall.
    - **AttackAction:** EntityAction that makes an entity attack an entity target.
    - **OpenInventoryAction:** EntityAction that opens the inventory.
    - **CloseInventoryAction:** EntityAction that closes the inventory.
    - **UseItemAction:** EntityAction that consumes any usable item.
    - **PickUpAction:** EntityAction that adds an item on the ground to the inventory.
    - **InteractAction:** EntityAction that interacts with a specific entity. (Deprecated since targeted commands were not implemented).
    - **InteractEverything:** EntityAction that interacts with all entities surrounding the controller.
* **dungeon.py**
  + **Classes**
    - **Rect:** Class representing a rectangle composed of grid coordinates.
    - **BspNode:** Class used to represent a partition in the Binary Space Partitioning algorithm.
  + **split\_factor:** Generates a split factor for a partition to be split across an axis constrained in a specific range.
  + **split\_rect**: Randomly splits a rectangle across a split axis.
  + **valid\_position:** Queries a game map if an entity can be placed at a certain location.
  + **paint\_node:** Recursively paints a BspNode tree at all of its leaves, placing walls and tiles.
  + **paint\_corridors:** Paints walkable corridors between all leaf BspNodes
  + **place\_npcs:** Places NPCs throughout the map.
  + **neighbor\_idx\_dist:** Returns smallest wraparound index difference. Used for door placement algorithm.
  + **add\_doors:** Bespoke algorithm thatplaces doors in a variety of special cases throughout the map.
  + **print\_rec:** Used to debug all of the rooms.
  + **pick\_key\_room:** Picks a key room that is accessible to the player by traversing the BspNode tree.
  + **\_pick\_key\_room\_rec:** Helper function that populates possible key rooms.
  + **generate:** Main function to generate and populate a map level.
* **items.py**
  + **Classes**
    - **Item:** Representation of an item that can be interacted with using the inventory systems.
    - **HealingPotion(Item)**: Item archetype representing a potion that heals the player.
    - **Key(Item):** Used to open the locked door.
* **sprites.py**
  + **Classes**
    - **Spritesheet**
      * **load\_sprites:** Initializes the sprite sheet and loads the masks and sprites.
      * **get\_sprite:** Accesses the sprite at a certain index.
      * **get\_mask:** Accesses the mask at a certain index.

## Third-Party Tools

The following APIs, toolkits, and frameworks were used in the development of our prototype:

* Python 3.12
* PyGame-CE - Python-friendly framework that handles windows, rendering, and input events.
  + <https://pyga.me/>
  + Pygame was used as the framework and rendering engine for Python. Minimal difficulties were faced, save for implementing mask-based damage highlighting. A fading red square was used instead for placeholder purposes.
* Google Voice Recognition - Third-party API that takes speech input and returns text
  + <https://cloud.google.com/speech-to-text>
* SpeechRecognition - implements Google’s Voice Recognition API into Python
  + <https://pypi.org/project/SpeechRecognition/>
  + SpeechRecognition is a very simple but limited library. It was easy to implement, but virtually impossible to extend or customize. Its opinionated design choices prevented our team from adding more manual and specialized control involving the listeners when implementing the passive listening feature.
* OpenAI Whisper
  + <https://github.com/openai/whisper>
  + Whisper was the second speech recognition model we used. Given our experience working with the previous model, we faced no issues in adjusting the program’s code to work with Whisper.
* PyAudio
  + <https://pypi.org/project/PyAudio/>
  + PyAudio was used to support speech input via a microphone. Some issues were encountered with specific versions of Python, such as the latest release, 3.13, which removed some modules that PyAudio depended on. After troubleshooting, we determined that the ideal version of Python to use for this project was 3.12
* m3x6 Font
  + <https://managore.itch.io/m3x6>
  + The font used for the game’s UI. No issues were encountered when implementing this font.

# User Study Overview

Our testers primarily consisted of college-age individuals from diverse backgrounds. Many of them already had experience playing games. We conducted ten individual user studies at different times, ensuring that each participant approached our prototype with a fresh perspective.

Each study followed a standardized format comprised of the following components:

* **Explanation for the study:** We introduced ourselves and the project, outlined the study's objectives, informed participants about the expected duration, and reassured them that the focus was on the prototype rather than their individual performance.
* **Observation of task completion:** The facilitator assigned a series of tasks to evaluate the prototype's functionalities. The facilitator monitored critical incidents, while a note-taker recorded the number of successful and unsuccessful voice commands and the total task completion time. Both the facilitator and note-taker watched for unrecognized user commands. The tasks included exploring movement commands, taking an item for interaction commands, opening a locked door, and locating stairs to complete the level.
* **Post-study interview:** Following the study, a brief interview was conducted which focused on three areas: user experience to gather thoughts and feelings, pain points to identify common issues, and open feedback for additional insights beyond predefined categories.

Studies were conducted from 11/4/2024 to 11/14/2024, frequently during class periods, as well as at times when participants were available in private settings. Each study lasted approximately 20 to 30 minutes, allowing sufficient time for an introduction, task execution, and subsequent feedback questions and discussions.

Half of the studies were carried out in the CS 423 classroom, where team members facilitated the sessions. The remaining studies were conducted over streaming, with the tester’s audio being fed into the prototype as microphone input and the game’s visuals being streamed back to the tester.

In each study, the roles of note taker and facilitator have rotated to afford each member the experience in conducting the tests, and the overall collection of data.

There was at least one facilitator and one note-taker present during the studies. The facilitator was responsible for monitoring the time allocated for each task assigned to the user while also providing assistance as needed. Conversely, the note taker meticulously recorded the number of commands issued and the errors encountered throughout the process.

# User Study Findings

The user study identified four critical incidents or CI. CI#1 described inadequate response recognition, with 43.4% of the 325 commands issued by 10 participants going unrecognized, and 17.% being misinterpreted, often due to unexpected syntax or vocabulary limitations. This resulted in user frustration, leaving them uncertain about how to phrase commands to receive appropriate responses. Noise pollution, identified as CI#2 further compounded the issue, although it was not the only contributing factor. Instances of multi-step command misfires were observed as CI#3, in which the interpreter struggled to process sequential commands or variations in syntax, as experienced by participants 10 and 7. Background noise and unintended speech frequently caused delays in command recognition, leading to errors and repeated inputs. Visual feedback also emerged as a weak point, described as our CI#4, with participants finding the interaction mechanics confusing due to inadequate feedback following their actions, such as attacking enemies. These challenges underscored the need for improvements, including an expanded vocabulary and syntax recognition, enhanced noise filtering, better interface feedback, and more visual indicators to clarify system responses.

# Lessons Learned

Following our user studies, we concluded that our speech recognition model at the time would not adequately serve the project objectives. Each participant encountered challenges with the interpreter and background noise, even in more controlled environments, as mentioned above in CI#1 and CI#2. Consequently, we researched more alternative models and opted to utilize the Whisper Speech-to-Text API, which proved to be more robust than our previous model, offering enhanced accuracy and superior noise reduction capabilities. We adjusted the calibration phase of the speech recognizer by extending the background noise capture duration from one second to five seconds. This change allows for sufficient audio data to be collected to accurately set the recognition threshold.

We also expanded the interpreter's vocabulary to accommodate the variations in commands identified during our user studies, per CI#3. Additionally, we enhanced the interpreter's command parsing capabilities. This improved parsing method accomplishes two primary functions: it accounts for variations in sentence structures and enables the interpreter to decompose multi-step commands into discrete actions. Visual indicators were implemented to clarify which items or enemies are within the interaction range of users. These indicators also register user actions, such as attacking an enemy or sustaining damage from an enemy. Finally, we optimized player movement to ensure that the player advances one square at a time and receives error notifications when attempting to navigate through impassable doors. This change prevents players from unintentionally bypassing barriers by issuing commands to move into areas beyond those boundaries.

# Prototype Changes

The following changes were made to the prototype after conducting our user studies:

### Change 1: Expanded interpreter vocabulary & command parsing functionality

* **Reason:** This change was implemented to address Critical Incidents #1 and #3, specifically regarding the difficulties users faced in having their commands understood by the interpreter, often requiring multiple attempts. These improvements were expected to improve the interpreter's understanding of a wider range of words and commands, thereby reducing the frequency of misunderstandings.
* **Image:**

| **Before Change** | **After Change** |
| --- | --- |
| 1. User is about to enter a huge room. | 1. User is in the middle of a hallway. with a space between them and the door. |
| 1. User has entered the room with “Move right, then move down” displayed at the top left. | 2. User has moved next to the now opened door with the text “Move right, then open door” |

### Change 2: Addition of range-based visual indicators

* **Reason:** In direct response to CI#4, this modification enhances the alignment of human-AI interactions within the project. By implementing selection indicators for items and hit indicators for enemies, users will gain a more comprehensive understanding of the game's responsiveness to their actions.
* **Image**:

| **Before** | **After** |
| --- | --- |
| 1. User is next to a skeleton sprite and potion on the ground | 2. User is next to a slime with a label overhead saying “slime”, and next to a potion saying “Health Potion” above it. In the bottom of the screen it lists the actions, “attack (slime)” and “pick up (Health Potion)” |

### Change 3: Single-step movement & improved error reporting for doors

* **Reason:** This adjustment addresses CI#3 and CI#4 to enhance the user’s comprehension of how the project interprets their commands. By providing continuity between the user’s inputs and notifying them of any potential issues, users can more effectively adjust their input, thereby improving the project’s understanding of their intentions.
* **Image**:

| **Before** | **After** |
| --- | --- |
| \* The command “Up three” allowed the user to bypass the locked door | \* Player is in front of a locked door with error text saying “Action is not valid.” |

# Appendix: Task Sheets, Surveys, and Interview Questions

* Sign-up sheet
  + <https://docs.google.com/document/d/1MrEHasnlb-r4BNTC083z6QINVvMofWqW/edit>
* Task Instructions
  + <https://docs.google.com/document/d/1Dvx_MdgW3QcjJGr7cNX5kzTzBVxXkBjg/edit>
* Survey
  + <https://docs.google.com/forms/d/1NnPlXhC954zm2CB4p7FxOKMTeViiRc1V_XCUqjHm8Ms/edit>

# Future Work

If we were to extend the project beyond this semester, our initial step would be to conduct additional user testing with our enhanced prototype, as we did not have this opportunity before. One improvement to the user tests would involve standardizing the game environment for each participant, rather than having them navigate through a procedurally generated level. The hope with this adjustment is to gather more accurate data on task completion times, as some users finished tasks more quickly simply due to fortunate alignments of the level key and door being in close proximity to each other. The second step, in tandem with the first, would be to collect data from a broader pool of participants, particularly those who would benefit more from a speech-based approach to gaming rather than those using it solely for leisure or educational purposes. This approach will help us identify any potential blind spots in our program concerning accessibility as well the effectiveness of the program’s feedback. We anticipate that producing a viable prototype for our target audience will require substantial effort, with a goal of achieving this within a six-month development period.

# Source Code

* Source code available on Blackboard
* Link to GitHub repository
  + <https://github.com/vcolome42/CS-423-BARD>
* README file included in source code

# 10. Video

<https://drive.google.com/file/d/1dzgqt2ym76hdsaiRmSjcbDThWz812fJw/view?usp=sharing>

**AI Usage Statement**

No AI was used in the development of this project nor in the process of writing this report.

However, AI models were used in the application such as Open AI Whisper, Vosk, and more. More explicit usage in project repo.

**Bibliography**

1. Zhang, Anthony. *Speech Recognition* (Version 3.11). 2017. Software. GitHub,<https://github.com/Uberi/speech_recognition#readme>. Accessed 5 Dec. 2024.
2. "Basic BSP Dungeon Generation." *RogueBasin*, n.d.,<https://www.roguebasin.com/index.php/Basic_BSP_Dungeon_generation>. Accessed 5 Dec. 2024.