



Przedmiot: Interfejsy człowiek-komputer

## HCI2025\_1\_VOICE

Temat projektu: **VOICE - Moduł sterowania głosem dla gry.**

Spis treści:

1.	ABSTRAKT.....	2
2.	WSTĘP I BADANIA LITERATUROWE .....	3
3.	KONCEPCJA PROPONOWANEGO ROZWIĄZANIA.....	3
4.	REZULTATY I WYNIKI TESTÓW I/LUB REALIZACJA ZADAŃ PROJEKTOWYCH.....	4
5.	REALIZACJA PROPONOWANEGO ROZWIĄZANIA.....	5
6.	PODSUMOWANIE I WNIOSKI.....	6
7.	LITERATURA .....	6
8.	DODATEK A: OPIS OPRACOWANYCH NARZĘDZI I METODY POSTĘPOWANIA .....	7
9.	DODATEK C. OPIS INFORMATYCZNY PROCEDUR.....	8

### Wykonali:

*Paweł Lejczak*

*Kamil Maj*

*Remigiusz Mietła*

Rok I AiR II st.

**Konsultant:** *Jaromir Przybyło*

Wersja 1.0

Kraków, maj 2025.

## 1. Abstrakt

Celem projektu było opracowanie modułu gry komputerowej umożliwiającego sterowanie postacią i zdarzeniami w grze za pomocą poleceń głosowych. Założeniem było stworzenie intuicyjnego i responsywnego interfejsu głosowego, który zwiększy immersję oraz komfort użytkownika podczas rozgrywki. Prace obejmowały integrację lokalnego systemu rozpoznawania mowy z mechaniką gry w czasie rzeczywistym, co wymagało opracowania efektywnych algorytmów przetwarzania sygnału i optymalizacji opóźnień.

Do implementacji wykorzystano bibliotekę Vosk, pozwalającą na offline'owe rozpoznawanie mowy bez potrzeby stałego dostępu do internetu. W ramach komunikacji między modułami zastosowano protokół UDP, zapewniający szybkie i niezawodne przesyłanie komend głosowych. System został skonfigurowany do wykrywania i interpretowania podstawowych poleceń takich jak „enter”, „cancel”, „fight”, „block”, „heal” oraz „aim”.

Algorytm został zaprojektowany z myślą o pracy w zróżnicowanych warunkach akustycznych – moduł rozpoznawania głosu automatycznie dostosowuje się do zmiany źródła dźwięku oraz umożliwia łatwe dodawanie kolejnych komend za pomocą przejrzystego pliku konfiguracyjnego JSON. Algorytm skoncentrowano na wykrywaniu pojedynczych słów w języku angielskim, wykorzystując lekki model „small” w celu zwiększenia szybkości i płynności działania w czasie rzeczywistym. Przeprowadzono testy na różnych komputerach o zróżnicowanej specyfikacji, z użyciem wielorakich źródeł dźwięku oraz przez grupę testerów o odmiennych głosach i akcentach. Ostateczna weryfikacja funkcjonalności odbyła się podczas spotkania integrującego wszystkie moduły gry, gdzie potwierdzono stabilność i ergonomię interakcji głosowej w pełnym środowisku rozgrywki.

**Słowa kluczowe:** rozpoznawanie mowy, sterowanie głosem, gry komputerowe, interfejs głosowy, Vosk, UDP, immersja.

## **2. Wstęp i badania literaturowe**

Niniejszy raport opisuje projekt modułu sterowania głosowego dla gry komputerowej, mający na celu umożliwienie interakcji z wirtualnym światem za pomocą predefiniowanych komend głosowych. Głównym problemem projektowym było stworzenie systemu zdolnego do rozpoznawania mowy użytkownika w czasie rzeczywistym i przekazywania zidentyfikowanych komend do aplikacji gry w sposób efektywny i z niskim opóźnieniem.

Integracja automatycznego rozpoznawania mowy (ASR) z aplikacjami interaktywnymi, w tym grami, jest obszarem aktywnie rozwijanym, oferującym potencjał zwiększenia immersji i dostępności [1]. Badania w tej dziedzinie często porównują rozwiązania chmurowe (online) z lokalnymi (offline). Rozwiązania chmurowe, choć zazwyczaj oferują wyższą dokładność ogólnego rozpoznawania, wprowadzają opóźnienia zależne od połączenia sieciowego i wymagają stałego dostępu do Internetu, co stanowi wyzwanie w dynamicznych środowiskach [2]. Lokalne silniki ASR, działające bezpośrednio na urządzeniu użytkownika, takie jak biblioteki oparte na Kaldi (np. Vosk) czy CMU Sphinx, są preferowane w aplikacjach czasu rzeczywistego ze względu na niższe opóźnienia i możliwość pracy offline [2]. Wybór biblioteki Vosk dla tego projektu podyktowany był jej możliwością działania bez dostępu do Internetu, relatywnie niskimi wymaganiami sprzętowymi oraz łatwością integracji z językiem Python, co czyni ją odpowiednią dla zadań typu command & control.

## **3. Koncepcja proponowanego rozwiązania**

Koncepcja systemu sterowania głosowego opiera się na architekturze modułowej, w której główny moduł rozpoznawania mowy działa niezależnie od aplikacji gry i komunikuje się z nią za pomocą standardowego protokołu sieciowego. Przyjęto model klient-serwer, gdzie moduł głosowy pełni rolę klienta wysyłającego komendy, a aplikacja gry rolę serwera nasłuchującego na te komendy.

Główne komponenty koncepcji to:

1. Moduł Przechwytywania Audio: Odpowiedzialny za pobieranie strumienia danych dźwiękowych z mikrofonu systemowego użytkownika.
2. Moduł Rozpoznawania Mowy (ASR): Przetwarza strumień audio, konwertując mowę na tekst. Działa w trybie offline, wykorzystując lokalnie zainstalowany model językowy.
3. Moduł Identyfikacji Komend: Analizuje tekst zwrócony przez moduł ASR i dopasowuje go do predefiniowanego zestawu komend gry. Definicje komend, w tym alternatywne frazy, są

wczytywane z zewnętrznego pliku konfiguracyjnego (JSON). Moduł ten odpowiada za mapowanie rozpoznanej frazy na unikalny identyfikator komendy zrozumiałej dla gry.

4. Moduł Komunikacji Sieciowej: Odpowiedzialny za wysyłanie zidentyfikowanych komend do aplikacji gry. Wykorzystuje protokół UDP (User Datagram Protocol) ze względu na jego charakterystykę niskiego opóźnienia, co jest kluczowe dla interakcji w czasie rzeczywistym w grach.

#### 4. Rezultaty i wyniki testów i/lub realizacja zadań projektowych

Przeprowadzone testy potwierdziły skuteczność oraz stabilność działania zaprojektowanego systemu rozpoznawania mowy w kontekście sterowania grą komputerową. Zastosowany model rozpoznawania mowy typu „small” potrzebował średnio około 350 milisekund na wykonanie jednej predykcji. Pomimo tego, dzięki zastosowaniu mechanizmu wykrywania wielu słów jednocześnie oraz odpowiedniemu buforowaniu danych wejściowych, udało się uzyskać płynność działania pozwalającą na interakcję w czasie rzeczywistym. Model bardzo dobrze radził sobie z językiem angielskim, co było zgodne z jego przeznaczeniem. Gdy użytkownicy wypowiadali komendy wyłącznie po angielsku, dokładność rozpoznawania została oszacowana na poziomie około 95%. W sytuacjach, gdy mowa była prowadzona w innym języku (np. polskim), algorytm próbował błędnie dopasowywać usłyszane dźwięki do znanych mu angielskich komend, co skutkowało spadkiem trafności i licznymi pomyłkami. Funkcja obsługi wyjątków została zaimplementowana skutecznie – system potrafił wykrywać i reagować na problemy z połączeniem sieciowym oraz dynamiczne zmiany źródła dźwięku. W takich przypadkach uruchamiano odpowiednie procedury diagnostyczne. Takie podejście zapewniło stabilność i niezawodność aplikacji w różnych warunkach testowych.

```
Using audio input device: (index 1)
2025-05-08 21:08:36,684 - INFO - Application was setuped
2025-05-08 21:08:41,572 - INFO - Message was send - MSG: 20 -> Grupa Voice: enter
2025-05-08 21:08:43,551 - INFO - Message was send - MSG: 21 -> Grupa Voice: cancel
2025-05-08 21:08:45,548 - INFO - Message was send - MSG: 22 -> Grupa Voice: fight
2025-05-08 21:08:47,284 - INFO - Message was send - MSG: 23 -> Grupa Voice: block
2025-05-08 21:08:50,048 - INFO - Message was send - MSG: 24 -> Grupa Voice: heal
2025-05-08 21:08:52,045 - INFO - Message was send - MSG: 25 -> Grupa Voice: aim
2025-05-08 21:08:55,794 - INFO - Message was send - MSG: 2112 -> Grupa Voice: cancel, cancel, fight
2025-05-08 21:09:00,040 - INFO - Message was send - MSG: 2341 -> Grupa Voice: block, heal, cancel
2025-05-08 21:09:04,831 - INFO - Message was send - MSG: 22 -> Grupa Voice: fight
2025-05-08 21:09:08,796 - INFO - Message was send - MSG: 252 -> Grupa Voice: aim, fight
```

Rysunek 1. Rezultaty działania systemu

```
2025-05-08 21:09:46,554 - INFO - Message was send - MSG: 240 -> Grupa Voice: heal, enter
Audio read error: [Errno -9999] Unanticipated host error
Attempting to recover audio stream...
Error while closing audio stream: Stream not open
Using audio input device: (index 0)
Audio stream successfully restored.
2025-05-08 21:10:06,912 - INFO - Message was send - MSG: 25 -> Grupa Voice: aim
2025-05-08 21:10:10,389 - INFO - Message was send - MSG: 234 -> Grupa Voice: block, heal
2025-05-08 21:10:26,633 - INFO - Message was send - MSG: 25 -> Grupa Voice: aim
```

Rysunek 2. Obsługa zmiany wejścia audio

## 5. Realizacja proponowanego rozwiązania

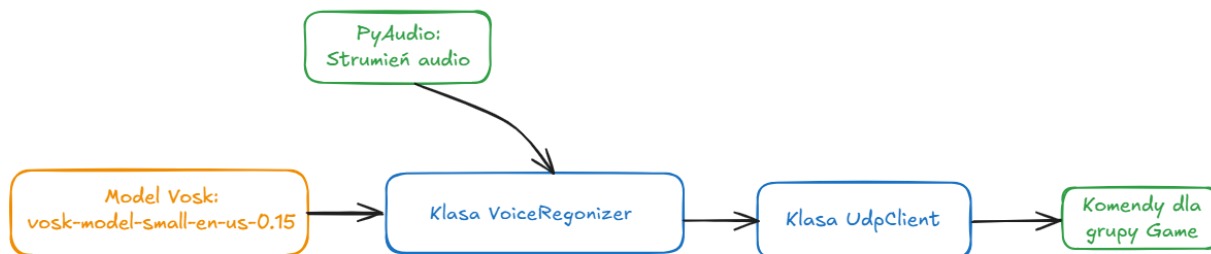
Realizacja proponowanej koncepcji została przeprowadzona w języku Python, wykorzystując dostępne biblioteki do poszczególnych zadań. Kod źródłowy został podzielony na moduły odpowiadające za rozpoznawanie mowy i komunikację sieciową, zgodnie z zaprojektowaną architekturą.

Moduł rozpoznawania mowy został zaimplementowany w klasie VoiceCommandRecognizer. Jego głównym zadaniem jest przetwarzanie sygnału audio z mikrofonu i identyfikacja komend głosowych. Wykorzystuje on bibliotekę Vosk jako silnik rozpoznawania mowy, który działa w trybie offline i konwertuje strumień dźwięku na tekst. Przechwytywanie danych audio z mikrofonu systemowego realizowane jest za pomocą biblioteki PyAudio. Moduł ten zawiera również logikę zarządzania strumieniem audio, w tym próby automatycznego odzyskiwania połączenia z urządzeniem w przypadku błędów.

Definicje komend, które system ma rozpoznawać, są wczytywane z zewnętrznego pliku konfiguracyjnego w formacie JSON. Plik ten zawiera mapowanie pomiędzy wewnętrznymi identyfikatorami komend (używanymi przez grę) a listą alternatywnych fraz, które użytkownik może wypowiedzieć. Moduł VoiceCommandRecognizer przetwarza ten plik, tworząc strukturę danych ułatwiającą szybkie wyszukiwanie.

Główny proces rozpoznawania polega na ciągłym odczytywaniu niewielkich fragmentów danych audio i przekazywaniu ich do silnika Vosk. Gdy Vosk zwróci wynik rozpoznawania (nawet częściowy), moduł analizuje uzyskany tekst. Następnie, korzystając z biblioteki difflib i wczytanych definicji z pliku JSON, próbuje dopasować rozpoznane słowa do zdefiniowanych alternatywnych fraz komend. Zastosowanie dopasowania rozmytego pozwala na pewną tolerancję na niedoskonałości wymowy. Jeśli dopasowanie zostanie znalezione, moduł generuje ustandaryzowany ciąg znaków reprezentujący rozpoznaną komendę (np. zawierający prefiks i identyfikator komendy z JSON-a). Po pomyślnym rozpoznaniu komendy, silnik Vosk jest resetowany, aby przygotować się na kolejną frazę.

Komunikacja sieciowa z aplikacją gry została zaimplementowana w klasie UdpClient. Klasa ta tworzy i zarządza socketem UDP. Jej główną funkcjonalnością jest metoda umożliwiająca wysłanie ciągu znaków (reprezentującego rozpoznaną komendę) jako datagramu UDP do predefiniowanego adresu IP i portu, na którym nasłuchuje serwer gry. Wykorzystanie protokołu UDP zapewnia niskie opóźnienia, co jest kluczowe dla responsywności sterowania w grze.



Rysunek 3. Schemat systemu

## 6. Podsumowanie i wnioski

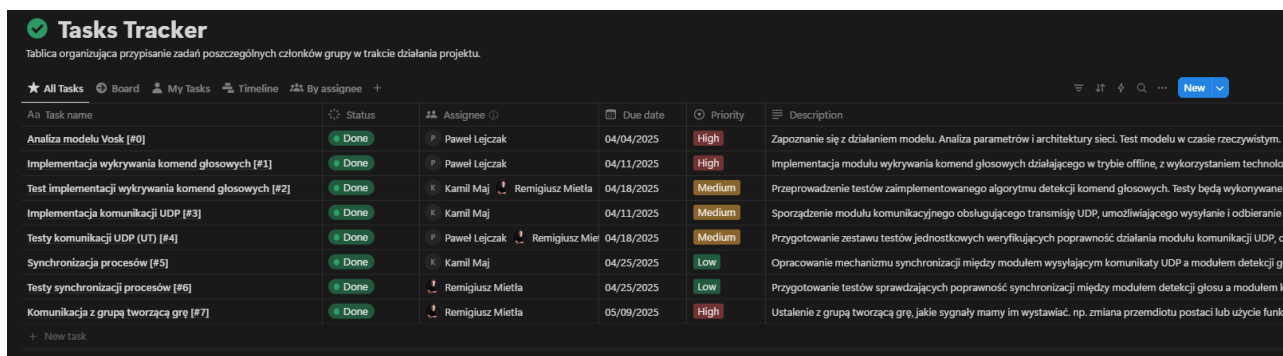
W ramach projektu zrealizowano funkcjonalny moduł głosowego sterowania grą, który działa stabilnie i skutecznie rozpoznaje komendy w języku angielskim. System spełnia założenia dotyczące pracy w czasie rzeczywistym oraz odporności na problemy sieciowe i zmiany sprzętowe. W przyszłości rozwój projektu może objąć rozszerzenie obsługi o inne języki, co zwiększy dostępność aplikacji dla szerszego grona użytkowników. Kolejnym kierunkiem może być implementacja rozpoznawania bardziej złożonych wypowiedzi, np. pełnych zdań w formie poleceń lub zaklęć, co pozwoliłoby na bardziej naturalną interakcję głosową. Można również rozważyć zastosowanie zaawansowanych modeli językowych opartych na sieciach neuronowych, które umożliwią lepsze rozumienie kontekstu i intencji użytkownika. Ważnym aspektem będzie też optymalizacja działania systemu pod kątem wydajności i możliwości uruchamiania na słabszym sprzęcie.

## 7. Literatura

- [1] J. Smith, „Enhancing Player Experience: The Role of Voice Control in Modern Video Game,” *Journal of Interactive Systems*, pp. 112-125, 2021.
- [2] L. Chen, „Performance Evaluation of Online vs. Offline Speech Recognition for Real-Time Interactive Applications,” *Proceedings of the International Conference on Human-Computer Interaction*, pp. 45-52.

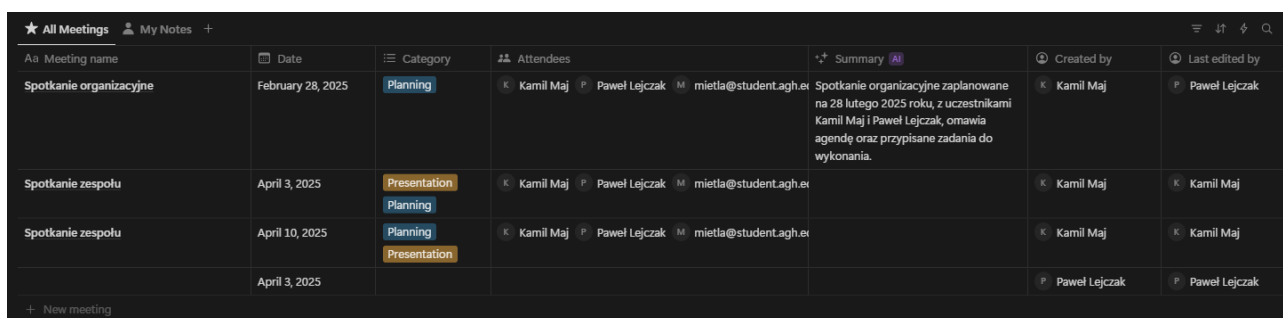
## 8. DODATEK A: Opis opracowanych narzędzi i metody postępowania

W projekcie wykorzystano język programowania Python, który dzięki swojej czytelności i bogatemu ekosystemowi bibliotek idealnie nadaje się do szybkiego prototypowania oraz tworzenia aplikacji współpracujących z systemami rozpoznawania mowy i komunikacją sieciową. Do realizacji funkcjonalności rozpoznawania mowy użyto biblioteki *Vosk*, która umożliwia lokalne (offline) rozpoznawanie mowy. Biblioteka ta wspiera wiele języków, jednak w projekcie skupiono się na modelu angielskim, optymalizowanym do pracy w czasie rzeczywistym. W celu umożliwienia komunikacji pomiędzy poszczególnymi komponentami gry, zastosowano bibliotekę *socket*, bazującą na protokole UDP. Rozwiązanie to pozwala na szybkie przesyłanie danych między procesami uruchomionymi lokalnie lub w sieci. Dla zapewnienia równoległego działania różnych fragmentów kodu (np. nasłuchu komend głosowych oraz przesyłania danych) wykorzystano bibliotekę *multiprocessing*, umożliwiającą uruchamianie wielu procesów w ramach jednej aplikacji i lepsze wykorzystanie zasobów systemowych. Do współpracy zespołowej, synchronizacji prac oraz kontroli wersji kodu używano platformy GitHub – repozytorium dostępne jest pod adresem: [VoiceGitHub](https://github.com/VoiceGitHub). Do organizacji zadań projektowych, przypisywania obowiązków oraz planowania etapów prac wykorzystano narzędzie Notion, które umożliwiło przejrzyste zarządzanie projektem w czasie jego trwania.



Task name	Status	Assignee	Due date	Priority	Description
Analiza modelu Vosk [#0]	Done	Paweł Lejczak	04/04/2025	High	Zapoznanie się z działaniem modelu. Analiza parametrów i architektury sieci. Test modelu w czasie rzeczywistym.
Implementacja wykrywania komend głosowych [#1]	Done	Paweł Lejczak	04/11/2025	High	Implementacja modułu wykrywania komend głosowych działającego w trybie offline, z wykorzystaniem technologii Vosk.
Test implementacji wykrywania komend głosowych [#2]	Done	Kamil Maj, Remigiusz Mietla	04/18/2025	Medium	Przeprowadzenie testów zaimplementowanego algorytmu detekcji komend głosowych. Testy będą wykonywane w trybie offline.
Implementacja komunikacji UDP [#3]	Done	Kamil Maj	04/11/2025	Medium	Sporządzenie modułu komunikacyjnego obsługującego transmisję UDP, umożliwiającego wysyłanie i odbieranie danych.
Testy komunikacji UDP (UT) [#4]	Done	Paweł Lejczak, Remigiusz Mietla	04/18/2025	Medium	Przygotowanie zestawu testów jednostkowych weryfikujących poprawność działania modułu komunikacji UDP, oraz testów integracyjnych.
Synchronizacja procesów [#5]	Done	Kamil Maj	04/25/2025	Low	Opracowanie mechanizmu synchronizacji między modułem wysyłającym komunikaty UDP a modułem detekcji głosu.
Testy synchronizacji procesów [#6]	Done	Remigiusz Mietla	04/25/2025	Low	Przygotowanie testów sprawdzających poprawność synchronizacji między modułem detekcji głosu a modułem komunikacji.
Komunikacja z grupą tworzącą grę [#7]	Done	Remigiusz Mietla	05/09/2025	High	Ustalenie z grupą tworzącą grę, jakie sygnały mamy im wystawić. np. zmiana przemiotu postaci lub użycie funkcji.

Rysunek 4. Rozpisane zadania w programie Notion



Meeting name	Date	Category	Attendees	Summary	Created by	Last edited by
Spotkanie organizacyjne	February 28, 2025	Planning	Kamil Maj, Paweł Lejczak, mietla@student.agh.edu.pl	Spotkanie organizacyjne zaplanowane na 28 lutego 2025 roku, z uczestnikami Kamil Maj i Paweł Lejczak, omawia agendę oraz przypisane zadania do wykonania.	Kamil Maj	Paweł Lejczak
Spotkanie zespołu	April 3, 2025	Presentation, Planning	Kamil Maj, Paweł Lejczak, mietla@student.agh.edu.pl		Kamil Maj	Kamil Maj
Spotkanie zespołu	April 10, 2025	Planning, Presentation	Kamil Maj, Paweł Lejczak, mietla@student.agh.edu.pl		Kamil Maj	Kamil Maj
	April 3, 2025				Paweł Lejczak	Paweł Lejczak

Rysunek 5. Odbyte spotkania projektowe zespołu w programie Notion

## **9. DODATEK C. Opis informatyczny procedur**

Pełna implementacja algorytmu wraz ze szczegółowym opisem funkcji w języku Python znajduje się w repozytorium projektu na platformie GitHub: [VoiceGitHub](#).