**Uniwersytet im. Adama Mickiewicza w Poznaniu**

**Wydział Matematyki i Informatyki**



**KCK SAPER**

Mikołaj Balcerek

Piotr Budkowski

Grzegorz Boiński

Arkadiusz Adam Powęska

Paweł Karczewski

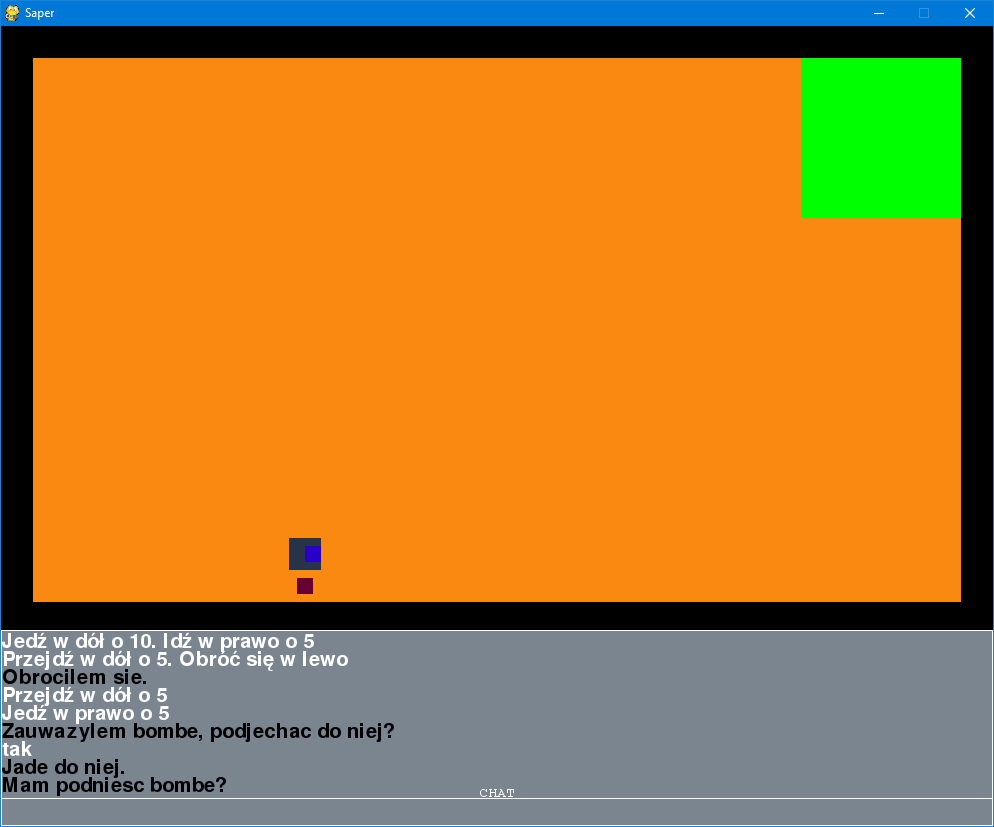
Sporządzający dokumentację: Mikołaj Balcerek

Prowadzący zajęcia:

prof. zw. dr. hab. Zygmunt Vetulani

1. **Model użytkownika**

Użytkownikiem programu ma być osoba szkoląca się na operatora robota-sapera lub tymczasowo odgrywająca taką rolę. Operator wymaga od aplikacji symbolicznego przedstawienia otoczenia sapera, interfejsu do sterowania robotem (za pomocą komunikatów w języku polskim) oraz efektywnego systemu podpowiedzi/ograniczeń dotyczących prawidłowej obsługi bomby.

1. **Opis zadania**

Treścią zadania było stworzenie systemu obsługi sapera za pomocą języka naturalnego (polskiego). Stworzona symulacja miała zawierać wizualizację problemu. Saper odpowiada na polecenia typu „Idź, Obróć się”, jest w stanie podjąć interakcję z bombą („Podnieś bombę”, „Zdetonuj”..) oraz dopytać się operatora o szczegóły polecenia lub zasugerować mu działania. Program jest również w pewnym stopniu zautomatyzowany. Potrafi wykryć bombę i jej typ z daleka, rozpoznaje możliwe działania i akcje związane z nią (np. nie każda bomba może zostać zdetonowana), nie pozwoli na podjęcie skrajnie niebezpiecznej akcji (detonacja bomby w strefie niebezpiecznej) i sam dojedzie do ładunku znajdującego się w małej odległości. Interakcja z systemem odbywa się przez zaawansowaną komunikację tekstową (pełne zdania, krótkie polecenia, odpowiedzi tak/nie i kombinacje powyższych) na ekranie. Aplikacja pamięta też ostatni kontekst rozmowy i na jego podstawie przetwarza niekompletne zapytania. Pewna część projektu jest w stanie przyjmować polecenia głosowe za pomocą aplikacji mobilnej (zadanie przyrostowe 2).

Przechwyt ekranu z aplikacji. Saper ma bazę synonimów, rozumie zgrupowane polecenia i prowadzi rozmowę z operatorem.

1. **Opis środowiska**

Do stworzenia aplikacji wykorzystaliśmy język **Python**.   
Wspiera on wiele paradygmatów programowania, a ze względu na wysoki poziom abstrakcji i jego dynamiczność, jest prosty w nauce i wykorzystaniu. Dzięki niejawnej deklaracji typów i ukrytym zarządzaniem pamięcią możliwy był szybki i eksperymentalny proces tworzenia programu.

Do stworzenia wizualizacji posłużyła biblioteka **pygame.**  
Moduł pygame jest najpopularniejszym tego typu rozszerzeniem do języka Python. Dzięki mnogości dostępnych poradników i jego prostocie wykorzystania był on oczywistym rozwiązaniem dla naszego projektu.  
  
Do zadania przyrostowego numer 2 (Komunikacja za pomocą aplikacji mobilnej) posłużył **HTML, Javascript, PhoneGap**.

Za pomocą HTML i Javascript powstała strona mobilna pozwalająca na wprowadzanie poleceń z dala od aplikacji. Obydwa te języki są standardami w Internecie. Platforma Adobe PhoneGap posłużyła za „zawinięcie” tej strony do aplikacji .apk obsługiwanej przez system operacyjny Android. Opcje ułatwienia dostępu systemu mobilnego pozwalają na przyjmowanie poleceń głosowych.

Uzupełnieniem zadania przyrostowego był **framework Spring** (Java) – serwer w

technologii REST.  
Framework Spring udostępnia proste API, pozwalające na odbiór poleceń zdalnych i uruchomienie systemu przetwarzania.

Na wczesnym etapie projektu wykorzystywaliśmy język programowania **Prolog.** Znany jest on ze swoich szerokich zastosowań w dziedzinie rozumienia języka naturalnego.

1. **Wymagania systemowe**

Do uruchomienia programu potrzebny jest interpreter języka Python w wersji 2.7. Dodatkowo wymagane jest także biblioteka pygame do Python 2.7.

Do uruchomienia zadań przyrostowych potrzebna jest aplikacja SWI-Prolog, przeglądarka internetowa/telefon komórkowy z systemem Android oraz JVM (Java Virtual Machine). Ważne, by serwer działał na tej samej sieci/tym samym komputerze co klient.