**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  
Arek Powęska

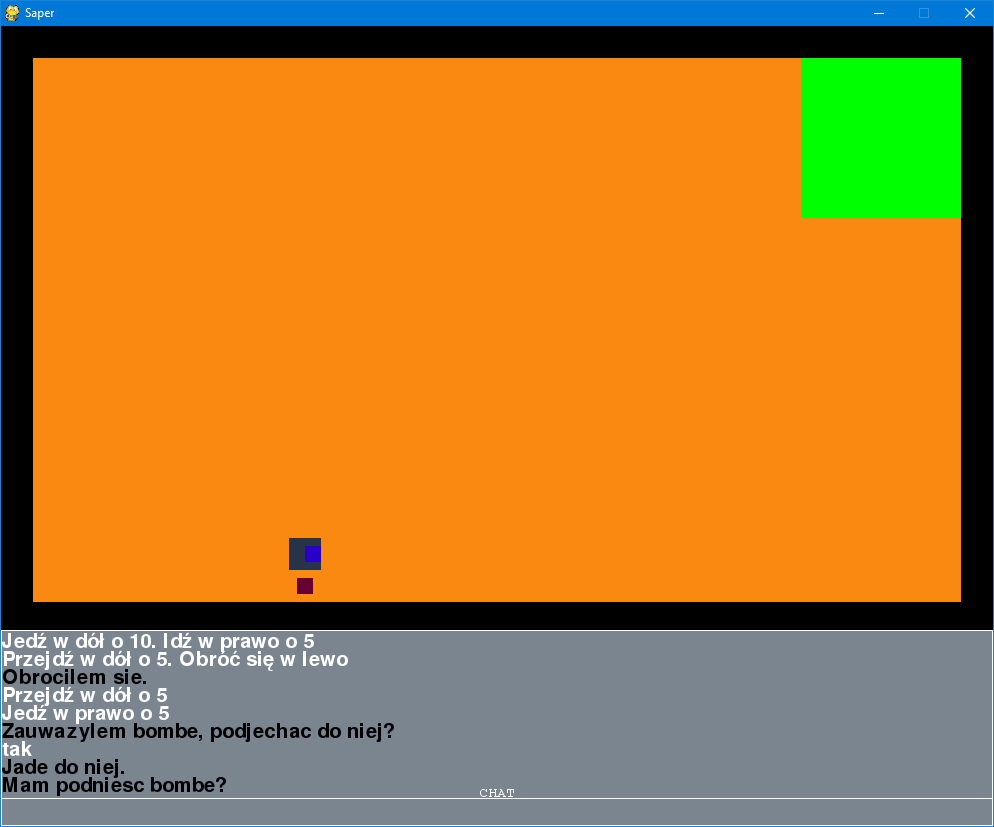
Prowadzący zajęcia:

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. W szczególności komunikacja w języku polskim ma się wyróżniać: akomodacją stylów pisania (polskie znaki lub ich brak), naturalnością rozmowy (pamiętanie kontekstu), precyzją (saper wykona tylko zlecone polecenia, w przypadku problemów zapyta się o kłopotliwe kwestie).

Operator posługuje się **językiem prostym, niespecjalistycznym.** Korzystać z wielu prostych określeń na podobne czynności (synonimów), zapisywać niektóre cyfry pisemnie oraz korzystać z określeń relatywnych do różnych punktów odniesienia (obróć się w tył, w przód (względem sapera), obróć się w górę (względem otoczenia)).

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. **Model fizyczny**

Oprogramowanie projektu składa się z programu SAPER w języku **Python** (wywołany przez uruchomienie KCK-Saper\Python-project\src\\_\_main\_\_.py).

Biblioteka **pygame** jest wykorzystana do wizualizacji (więcej w „Opis środowiska”).

Program **\_\_main\_\_.py** odpowiada za inicjalizację wszystkich obiektów i wstępną konfigurację. Nasłuchuje w pętli na znak końca wpisywania polecenia i sygnał zakończenia programu.

Obiekty podzielone są na klasy.

Klasa **Bomb (bomb.py)** odpowiada za obsługę bomby. Metoda \_\_init\_\_ wybiera właściwości bomby oraz znajduje jej miejsce na mapie. Metoda Update i Render odpowiadają za graficzną reprezentację bomby. Metoda Reset usuwa starą bombę i tworzy nową. Metoda Detonate obsługuje detonację bomby.

Funkcja save\_to\_file w pliku extract\_actions.py zapisuje logi rozmowy do pliku tekstowego.

Klasa **Map (map.py)** reprezentuję mapę. Metoda Load wczytuje mapę z pliku i tworzy teren o rozmiarze i typach zgodnych z zadanym plikiem. Metoda Render odpowiada za graficzną reprezentację Mapy.

Klasy **Wall i Strefa** są typami terenu na Mapie.

Klasa **Chat (recognize\_words.py)** obsługuje okno rozmowy z saperem. Metoda \_\_init\_\_ tworzy okno z domyślnymi ustawieniami. Metoda Update wyświetla animacje zgodnie z tempem odświeżania programu. Metody Render tworzy graficzną reprezentację czatu i pamięta historię rozmowy. Metoda get\_key pobiera wpisany klawisz. Metoda **ask** nasłuchuje specjalnych klawisze typu BACKSPACE, ENTER i je obsługuje.

Metoda **przetwarzanie\_jezyka** jest funkcją łączącą wszystkie podmetody i podfunkcje przetwarzania języka. Ustawia ona zmienne początkowe i wywołuje potrzebne podfunkcje. Metody get\_sentence\_from\_input\_to\_list i podziel\_po\_kropki pomagają dzielić rozkazy i pojedyncze słowa ze względu na kropki i spacje.   
Metoda **find\_word** szuka w Słowniku do jakiego obiektu należy podane słowo. Metoda **find\_all** za pomocą wielokrotnego wywoływania find\_word zamienia słowa na ich standardowe reprezentacje i klasyfikuje je według typów. Potrafi też obsłużyć liczby.  
Klasa **Saper (saper.py)** odpowiada za obsługę zachowań sapera. Metoda \_\_init\_\_ wybiera właściwości sapera oraz znajduje jego miejsce na mapie. Metoda Update sprawdza na bieżąco odległość do bomby, przenosi złapaną bombę oraz koryguje i pomaga w chodzeniu sapera. Metoda Render tworzy graficzną reprezentację. Metoda Polecenia obsługuje rozmowy z saperem, dopytuje się o brakujące dane oraz wywołuje podmetody znalezionych poleceń. Metody Detonate, Defuse, Drop, Pick\_up korespondują z akcjami związanymi z bombą. Metody Distance, Find\_Bomb, Move\_to\_bomb zajmują się automatyczną funkcją sapera podjeżdżania do bomby. Metody Walk, Rotate, Rotate\_dir, Move, Collision obsługują podstawowe funkcjonalności związane z obracaniem i chodzeniem.

Program uruchamiany jest na komputerze typu PC na systemie operacyjnym Windows/Linux z dostępnym interpretatorem języka Python i biblioteką Pygame.

Interpreter języka Python dynamicznie przypisuje programowi potrzebną pamięć.

1. **Model logiczny**

Zbiór wymagań stawianych przed systemem

-przemieszczanie sapera

-obracanie sapera

-możliwość samoczynnego podjazdu do bomby w przypadku gdy urządzenie ją „zauważy”

-rozpoznawanie typu bomby oraz podawanie zbiór akcji możliwych do podjęcia w danej sytuacji

-orientacja w terenie (poruszanie się tylko w wyznaczonym obszarze, świadomość występowania przeszkód, rozróżnienie strefy bezpiecznej)

-działania związane w obsługą bomby (podnoszenie, upuszczanie, rozbrajanie, detonowanie)

-nanoszenie na mapę nowej bomby, gdy obecna zostanie „obsłużona”

Zdefiniowanie funkcjonalnego modelu systemu opisującego

-interakcja systemu ze światem zewnętrznym odbywa się w oknie „CHAT” poprzez komendy tekstowe w języku prostym niespecjalistycznym

-funkcje, obiekty oraz parametry (dotyczące działań sapera) zawarte są w słowniku w języku XML umożliwiając łatwą edycję w razie potrzeby przez osobę obsługującą system

-program analizuje wpisany w okno CHAT-u tekst dopasowując każde słowo do tych znajdujących się w słowniku i w przypadku zgodności zwraca nazwę odpowiednio funkcji, obiektu bądź parametru. Na podstawie tego podejmowane są akcje sapera, jeżeli pozwala mu na to otoczenie (np. nie może pojechać dalej jeżeli napotkał ścianę; nie może zdetonować bomby w nieodpowiedniej strefie)

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**.

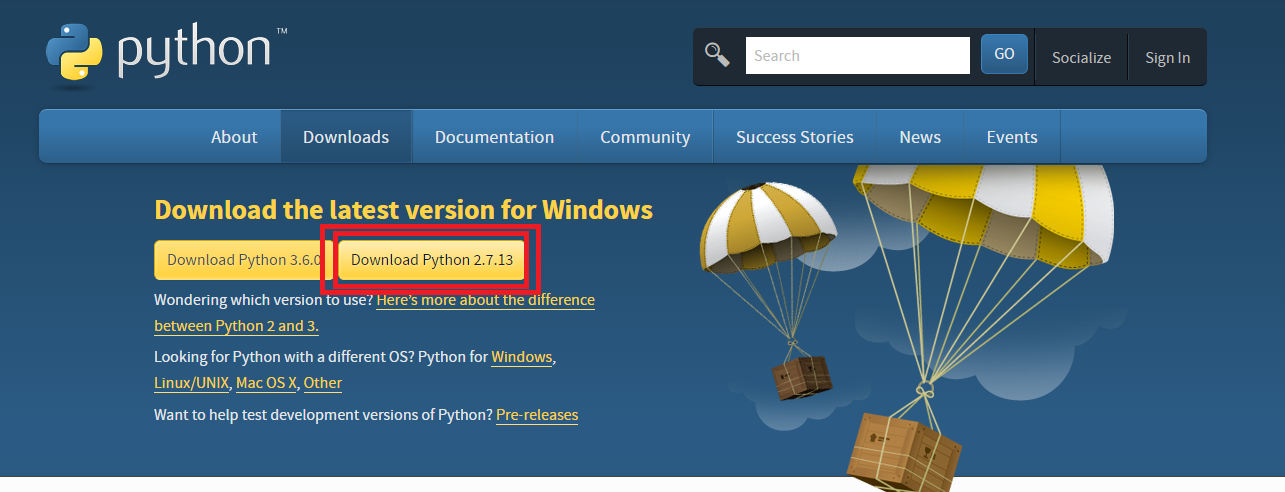
Za pomocą HTML i Javascript powstała strona mobilna pozwalająca na wprowadzanie poleceń z dala od aplikacji. Obydwa te języki są standardami w Internecie. Platforma Adobe PhoneGap posłużyła za „zawinięcie” tej strony do aplikacji .apk obsługiwanej przez system operacyjny Android. Opcje ułatwienia dostępu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

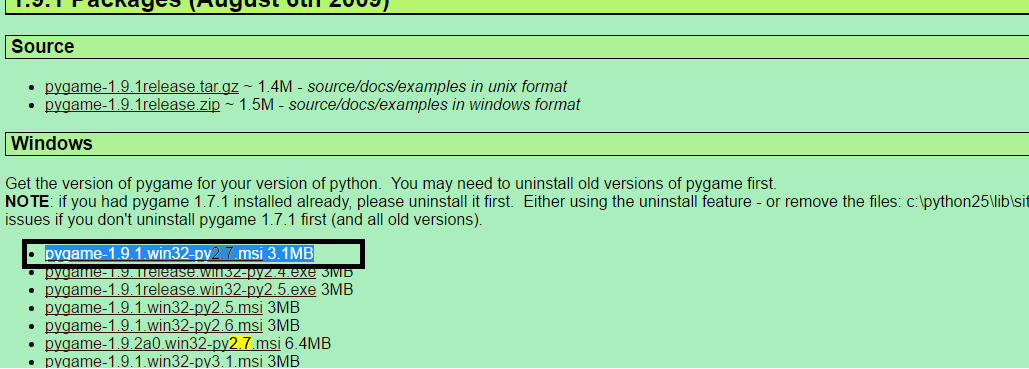
Na wczesnym etapie projektu wykorzystywaliśmy język programowania **Prolog.** Znany jest on ze swoich szerokich zastosowań w dziedzinie rozumienia języka naturalnego.

1. **Dokumentacja użytkownika**

Program wymaga komputera typu PC wraz z systemem operacyjnym **Windows/Linux.**

Strona internetowa z której należy pobrać interpreter Python. Wersja 2.7 zaznaczona w ramce

Do uruchomienia programu potrzebny jest interpreter języka Python w wersji 2.7. Interpreter języka Python można znaleźć na stronie internetowej: <https://www.python.org/downloads/>. Strona automatycznie dobierze wersję programu dla posiadanego systemu operacyjnego.  
 **Ważne, żeby wybrać wersję 2.7!!**

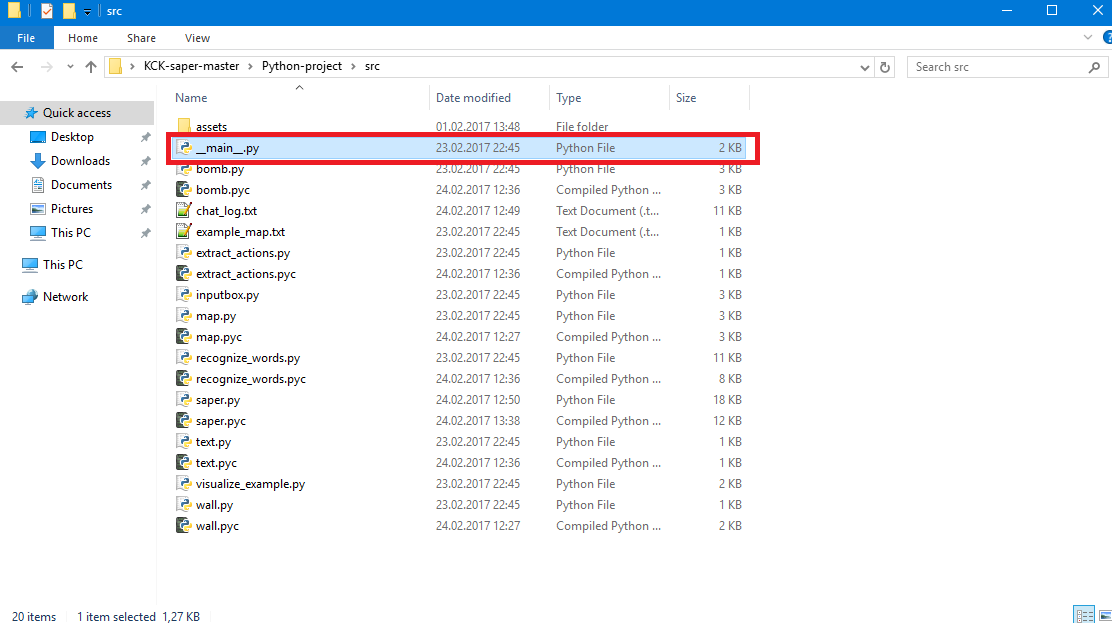


Dodatkowo wymagane jest także biblioteka **pygame** do Python 2.7.  
Znajdziemy ją na stronie <http://www.pygame.org/download.shtml>. Należy wybrać wersję dla Python 2.7 (na obrazku znajduje się zaznaczona poprawna opcja dla systemu Windows).

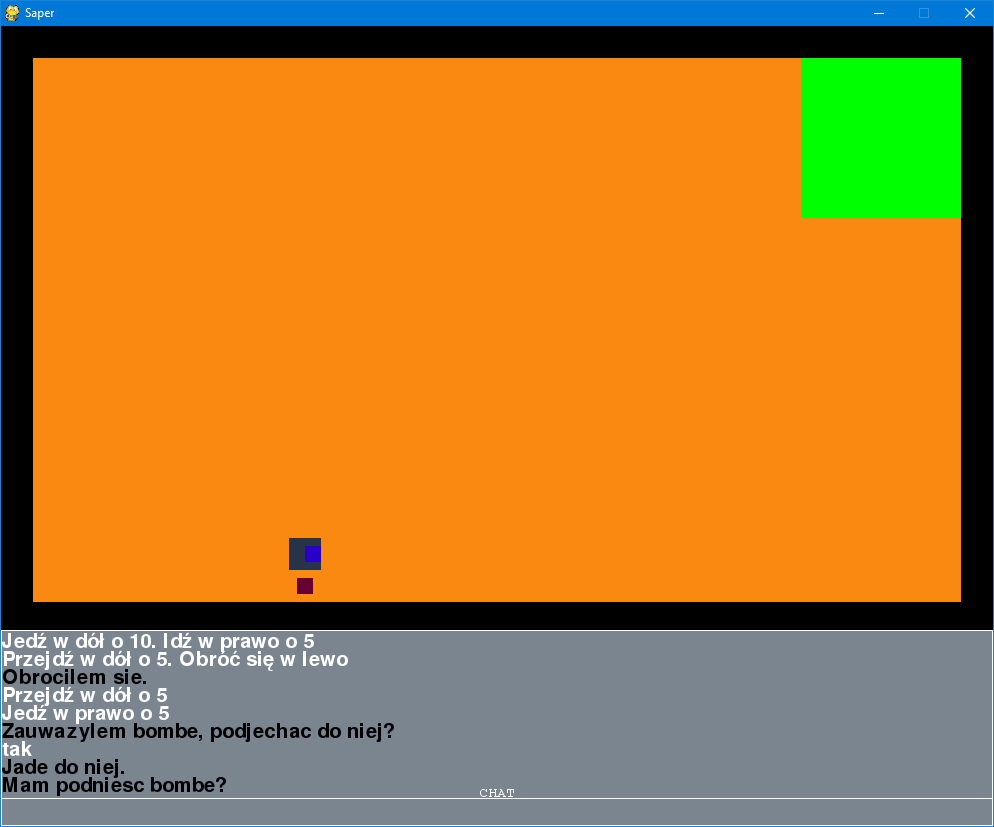
**Uruchomienie KCK-Saper**

Projekt uruchamia się poprzez uruchomienie odpowiedniego pliku. W systemie Windows po zainstalowaniu potrzebnego oprogramowania wystarczy nacisnąć podwójnie plik \\_\_main\_\_.py znajdujący się pod ścieżką KCK-Saper\Python-project\src\\_\_main\_\_.py.

Uruchomienie programu w systemie Windows 10



Korzystanie z **KCK Saper**

****

**Program reaguje na komendy:** Przejdź w prawo dwa, obróć się w tył, jedź w dół, podnieś bombę, rozbrój bombę, zdetonuj bombę, jedź o 6 metrów i podobne. Wiele rozkazów może być oddzielanych kropką. Liczby od 1-10 mogą być zapisywane słownie. Odpowiedzi na pytania sapera to np. tak, nie lub wybrana akcja z kilku zaprezentowanych.

**Zadania domowe**

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.