

Projekt z przedmiotu DKCK LI0 **Agent VLuke\_Jones** działający na aplikacji serwerowej „**Atlantyda**”

Skład grupy:  
-Magdalena Grygiel  
-Marcin Mateusz Hanc  
-Piotr Maciejewski  
-Miłosz Piotr Szczepaniak

Spis treści

[1. Wstęp 3](#_Toc315231369)

[2. Słownik pojęć 4](#_Toc315231370)

[3. Opis wykorzystanych technologii 5](#_Toc315231371)

[4. Podział programu na klasy, oraz ich opis 5](#_Toc315231372)

[1. Klasa „Program” 5](#_Toc315231373)

[1. Zmienne 5](#_Toc315231374)

[2. Metody 6](#_Toc315231375)

[2. Klasa „ZnanaMapa” 7](#_Toc315231376)

[1. Zmienne 7](#_Toc315231377)

[2. Metody 7](#_Toc315231378)

[3. Klasa „ZbadanePole” 8](#_Toc315231379)

[1. Zmienne 8](#_Toc315231380)

[2. Metody 8](#_Toc315231381)

[4. Klasa „Wspolrzedne” 8](#_Toc315231382)

[1. Zmienne 8](#_Toc315231383)

[5. Zawartość pliku AIML 9](#_Toc315231384)

[6. Opis algorytmu przeżycia 9](#_Toc315231385)

[7. Wyniki testów 10](#_Toc315231386)

[1. Samotny agent 10](#_Toc315231387)

[2. Kilkoro agentów 10](#_Toc315231388)

[3. Analiza wyników 10](#_Toc315231389)

[8. Dokumentacja użytkownika 10](#_Toc315231390)

[1. Instalacja 10](#_Toc315231391)

[1. Wersja normalna 10](#_Toc315231392)

[2. Wersja developerska 10](#_Toc315231393)

[2. Użytkowanie 10](#_Toc315231394)

[1. Uruchamianie agenta 10](#_Toc315231395)

[3. Praca nad projektem 11](#_Toc315231396)

[9. Najczęściej Zadawane Pytania (wraz z odpowiedziami) 11](#_Toc315231397)

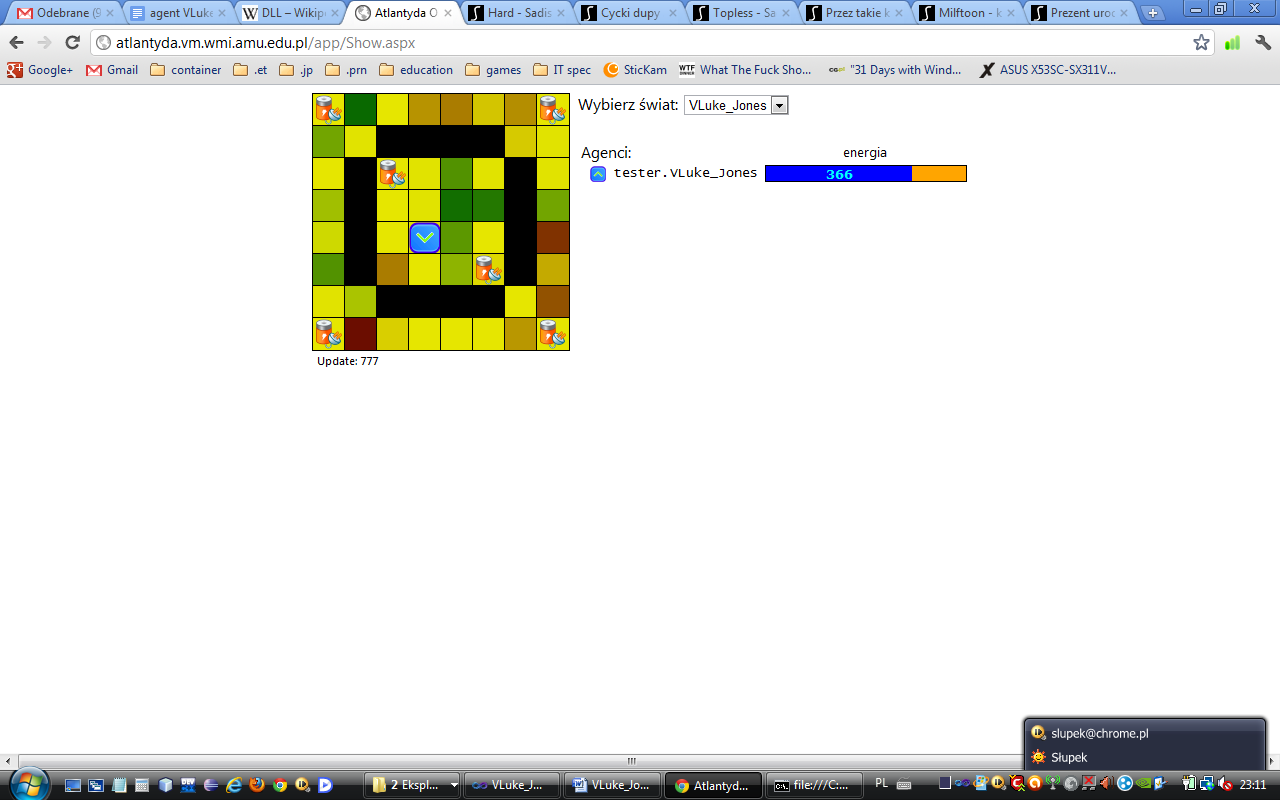
[Załącznik 1. Uproszczony diagram klas aplikacji. 12](#_Toc315231398)

[Załącznik 2. Uproszczony diagram stanów agenta. 13](#_Toc315231399)

# Wstęp

Praca ta została przygotowana przez naszą grupę jako projekt na przedmiot Komunikacja Człowiek-Komputer, który odbywał się w semestrze zimowym roku akademickiego 2011/2012. Założenia dotyczące projektu były mocno nieprecyzyjne. Najważniejszymi założeniami były (w kolejności priorytetu) :

1. Przeżyć
2. Eksplorować
3. Komunikować się(z pomocą AIML)
4. Współpracować ze sobą

Nasz agent skupia się na tych założeniach dokładnie w tej kolejności – bardzo martwi się o stan swojej energii. Lubi eksplorować, ale nawiąże też kontakt z podobnymi sobie. Nasz agent nie jest bardzo pomocny dla agentów pozostałych grup, gdyż komunikuje się TYLKO ze swoją grupą.

Ponieważ każda grupa miała samodzielnie decydować kiedy agent może stwierdzić, że wykonał zadanie, nasz agent kończy pracę w momencie gdy jest pewien, że mapa została maksymalnie odkryta. Maksymalnie, czyli poza obszarami położonymi za narożnikami bez przeszkód, za które to agent widzi, ale nie może tam dojść. Taki narożnik można zobaczyć na Rysunku 1.

Dokumentacja ta miała być też pełną dokumentacją techniczną dotyczącą tego projektu, tak więc taką dokumentację utworzyliśmy. Mamy nadzieję, że w razie gdyby ktoś chciał kontynuować nasze dzieło, nie będzie miał nam za złe lekkiego nieładu obecnego w kodzie.

Rysunek 1. Problematyczny narożnik.

Twórcy

# Słownik pojęć

Aby cała dokumentacja była w pełni zrozumiała, dodajemy słownik pojęć niekoniecznie dla każdego jasnych.

**agent** – Instancja programu klienckiego obsługiwanego przez „Atlantydę”. To właśnie ten program omawiamy.

**AIML** – (*Artificial Intelligence Markup Language)* Język znacznikowy służący do baz wiedzy chatterbotów. Jest bardzo łatwy do nauczenia.

**API** – (*Application Programming Interface*) Ściśle określony zestaw reguł i ich opisów, w jaki programy komunikują się między sobą.

**Atlantyda** – Aplikacja serwerowa obsługująca agentów różnych grup studentów. Obsługuje żądania agentów, pilnując ich położenia na mapie i energii. Przesyła też pomiędzy nimi komunikaty.

**biblioteka** – patrz **DLL**.

**DLL** – (*Dynamic-Link Library*) W środowisku Windows biblioteka współdzielona przechowująca implementacje różnych funkcji programu i/lub zasoby programu.

**seppuku** – Japoński rytuał samobójstwa poprzez rozpłatanie mieczem wnętrzności. Nazywany również harakiri.

# Opis wykorzystanych technologii

W naszym projekcie wykorzystaliśmy:

* **AIML** – Wymuszone zostało to założeniami projektu.
* **AIMLbot** – Najlepsze darmowa biblioteka dla języka C# służąca do obsługi chatterbota opartego na AIML.
* **Język C#** - Ponieważ jest to język najlepiej wspierany przez biblioteki „Atlantydy”.
* **Visual Studio 2010 - Ultimate** - Środowisko programistyczne najlepsze do programowania w języku C#.

# Podział programu na klasy, oraz ich opis

Nasz projekt znajduje się w jednej wielkiej klasie **Program**. Poniższy opis klas, ich zmiennych i metod opisuje nasz program. Uproszczony diagram klas naszej aplikacji znajduje się w **Załączniku 1**. Dodatkowo jako **Załącznik 2** zamieściliśmy diagram stanów przedstawiający w uproszczony sposób zachowanie się naszego systemu.

## Klasa „Program”

### Zmienne

* **AgentAPI** **agentVLuke\_Jones** – Zmienna służąca do komunikacji z serwerem Atlantydy. To przez nią wywołujemy polecenia dla agenta na serwerze.
* **int** **myEnergy** – Zmienna przechowująca stan energii naszego agenta.
* **int mojaPozycjaX, mojaPozycjaY** – Zmienne przechowujące położenie naszego agenta na zapamiętanej przez niego mapie.
* **bool gracz** – Zmienna logiczna ustawiana na `true`, jeżeli sterujemy agentem za pomocą klawiatury.
* **bool czyRozmawiam** – Zmienna logiczna ustawiana na `true` w trakcie rozmowy.
* **String rozmowca** – Zmienna przechowująca nazwę agenta z którym rozmawiamy.
* **String lastMessage** – Zmienna przechowująca ostatnio usłyszaną wiadomość.
* **String imie** – Zapamiętuje imię naszego agenta.
* **String groupname** – Zapamiętuje nazwę naszej grupy(„VLuke\_Jones”).
* **Wspolrzedne najblizszeZrodlo** – Zapamiętuje współrzędne najbliżej położonego źródła energii.
* **int kosztDoZrodla** – Zapamiętuje koszt najkrótszej drogi do źródła.
* **int wysokosc** – Zapamiętuje na jakiej wysokości znajduje się obecnie agent. Na polu startowym jest to 100.
* **int smallestCost** – Zapamiętuje najmniejszy koszt wykonania działania przez agenta na danej mapie. Służy określeniu kiedy agent ma popełnić seppuku.
* **int[2] kierunek** – Kierunek w którym patrzy nasz agent. Zapisany jako wektor.
* **int kierunekRozmowcy** – Zapamiętuje kierunek w którym zwrócony jest nasz rozmówca. Jest to enum o wartościach:
  + **0** – Patrzymy w tę samą stronę.
  + **1** – Widzimy lewy bok agenta.
  + **2** – Patrzymy na siebie.
  + **3** – Widzimy prawy bok agenta.
* **int[2] pozycjaRozmowcy** – Zapamiętuje współrzędne na których znajduje się nasz rozmówca. Współrzędne te dotyczą **naszej** zapamiętanej mapy.
* **WorldParameters cennikSwiata** – Zapamiętuje właściwości świata w którym wylądował nasz agent.
* **AIMLbot.Bot myBot** – Klasa pomocnicza do komunikacji za pomocą AIML.

### Metody

* **int CoTeraz()** – Klasa losująca zachowanie. Używana przez stan „Random”.
* **void Czekaj(int timeSec)** – Klasa wstrzymująca działanie głównego wątku naszego agenta na **timeSec** sekund.
* **void RotateLeft(), RotateRight(), StepForward()** – Podstawowe klasy służące do poruszania się. Służą kolejno do: obrotu w lewo, obrotu w prawo, pójścia o pole przed siebie.
* **bool isFrontStepable()** – Funkcja sprawdzająca, **tuż przed** wykonaniem kroku naprzód za pomocą metody **StepForward()**, czy pole tuż przed agentem nadal jest wolne. Zapobiega traceniu energii w skutek kary za próbę wejścia na przeszkodę.
* **void Look()** – Metoda dodająca i uaktualniająca pola znanej agentowi mapy. Sprawdza też, czy poznał on nowe najbliższe mu źródło energii, lub czy zauważył kogoś ze swojej drużyny i ma dość energii na konwersację. W razie posiadania odpowiedniej ilości energii – rozpoczyna konwersację.
* **void Listen(String krzyczacyAgent, String komunikat)** – Metoda wywoływana w przypadku usłyszenia przez agenta komunikatu. Jeśli sam jest jego nadawcą – ignoruje go. Jeśli nadał go ktoś spoza drużyny – też go ignoruje. W razie komunikatu nadanego przez kogoś innego niż aktualny rozmówca – ignoruje go. W razie komunikatu od rozmówcy – przesyła go do analizy.
* **void InterpretujWiadomosc(String komunikat)** – Metoda sprawdza czy komunikat dotyczy pozycji rozmówcy, czy danych mapy i przyswaja sobie odpowiednie dane, po czym odpowiada.
* **void Recharge()** – Metoda podstawowa służąca do automatycznego doładowania się energią gdy znajdziemy się na polu z nią, lub będąc na takim polu wykonamy czynność ją pobierającą.
* **void Speak(String wiadomosc)** – Metoda podstawowa służąca do wysyłania komunikatów. Niestety błąd Atlantydy uniemożliwia poprawne wyliczenie kosztów wysłania komunikatu. Obecnie zakładamy, że jest to koszt obrotu dla każdej wiadomości poza pierwszą po zmianie pola.
* **void KeyReader()** – Metoda pomocnicza służąca do debugowania agenta. Pozwala testerowi na „przejęcie sterów” agenta za pomocą klawiatury. Do sterowania agentem wykorzystywane są klawisze:
  + **↑** - Pójdź agentem o jedno pole do przodu.
  + **←** - Skręć agentem o 90° w lewo.
  + **→** - Skręć agentem o 90° w prawo.
  + **S** – wyświetl zbadaną mapę (patrz metoda **show()** w klasie **ZnanaMapa**).
  + **Enter** – Przechodzi w tryb wpisywania wiadomości. Po wpisaniu wiadomości ponowne wciśnięcie klawisza **Enter** wysyła ją.
  + **Q** – Wychodzi z programu.
* **void Talk(String toWhom, String komunikat)** – Metoda służąca do obsługi AIML. W oparciu o plik **bot.aiml** przygotowuje odpowiedzi na **komunikat**.
* **String hisCoordinates(int x, int y)** – Metoda zamieniająca na ciąg znaków gotów do wysłania w komunikacie współrzędne z zapamiętanej mapy naszego agenta przekształcone na relatywne współrzędne rozmówcy.
* **void Zachowanie(String myState)** – Metoda odpowiadająca za zachowanie się agenta w świecie. Jeżeli zmienna **myState** przyjmuje wartość „Random”, agent zachowuje się całkiem losowo (z pomocą metody **CoTeraz()**). W przeciwnym wypadku agent przechodzi w stan „SmartBot” w którym korzysta z całej dostępnej mu wiedzy i zaimplementowanych umiejętności.

## Klasa „ZnanaMapa”

### Zmienne

* **HashSet<ZbadanePole> znanaMapa** – Zbiór przechowujący wszystkie zapamiętane przez agenta pola.
* **int minX, maxX, minY, maxY** – Cztery zmienne przechowujące granice poza którymi agent nie wie aktualnie co się znajduje. Służą głównie do wyświetlania mapy agenta w czytelny dla człowieka sposób.

### Metody

* **void dodajPole(int x, int y, int jakaWysokosc, int ileEnergii, bool czyPrzeszkoda)** – Metoda dodająca na znaną mapę **ZbadanePole** o podanych właściwościach.
* **void uaktualnijPole(int x, int y, int ileOdjeto)** – Metoda uaktualniająca ilość energii na znanym wcześniej polu.
* **void uaktualnijPole(int x, int y, int jakaWysokosc, int ileEnergii, bool czyPrzeszkoda)** – Metoda dodająca na znaną mapę **ZbadanePole**, o ile pole o tych współrzędnych już nie istnieje. Wtedy uaktualnia je.
* **void show()** – Metoda służąca wyświetleniu w konsoli znanej agentowi mapy w czytelny dla człowieka sposób. Nie wyświetla wysokości oraz agentów (poza sobą). Występujące w tym wyświetleniu oznaczenia to:
  + **?** – Nieodkryte pole.
  + **@** - Pole tymczasowo uznane za nieosiągalne.
  + **X** – Przeszkoda.
  + **E** – Źródło energii.
  + **A < > V** – Cztery sposoby oznaczenia naszego agenta w zależności w którą stronę on obecnie jest skierowany.
* **bool test(int x, int y)** – Metoda sprawdzająca czy pole o podanych współrzędnych występuje już w naszej znanej mapie.
* **ZbadanePole znajdzPole(int x, int y)** – Metoda zwracająca pole o podanych współrzędnych. Wyszukuje te pole w znanej mapie. W razie nieistnienia takiego pola zwraca **null**.
* **Wspolrzedne findClosestEnergy()** – Metoda znajduje najbliższe agentowi (w znaczeniu najtaniej do niego się dostać) pole z energią i zwraca jego współrzędne.
* **void oneStepToThe(int x, int y)** – Metoda wykonująca pierwszy krok, jaki musi wykonać agent aby dostać się do pola o podanych współrzędnych.
* **int findShortestWay(int x, int y)** – Metoda zwracająca odległość najkrótszej ścieżki prowadzącej do pola o podanych współrzędnych.
* **int[2] findShortestWay(int x, int y, int FirstStep)** – Metoda zwracająca dwie liczby: odległość najkrótszej ścieżki prowadzącej do pola o podanych współrzędnych, oraz kod pierwszego kroku, jaki musi wykonać agent, aby dostać się do niego. Kody pierwszego kroku to:
  + **0** – Niemożliwość dostania się do tego pola
  + **1** – Wykonaj krok do przodu
  + **2** – Wykonaj krok w lewo (obrót i krok)
  + **3** – Wykonaj krok w prawo
  + **4** – Wykonaj krok do tyłu (najpierw podwójny obrót)
* **Wspolrzedne najblizszeUkryte()** – Metoda zwracająca współrzędne najbliżej położonego nieznanego nam pola do którego możemy się obecnie dostać.
* **int[2] znajdzDrogePowrotnaRek(int x, int y, int lastKier, List<Wspolrzedne> before, int upCost, int limit)** – Metoda rekurencyjna znajdująca najkrótszą drogę do pola obecnego pobytu agenta z pola o współrzędnych podanych. **lastKier** to kierunek z którego weszliśmy na obecnie rozpatrywane pole (**0** – północ, **1** – wschód, **2** – południe, **3** - zachód). **before** to odwiedzone już pola. **upCost** to koszt dojścia do obecnego momentu. **limit** to maksymalna odległość ścieżki wprowadzona z powodów kłopotów z efektywnością algorytmu przy większym znanym obszarze.
* **String toString()** – Metoda przekształcająca mapę do formy komunikacyjnej.
* **void dodajMapeZeStringa(String mapaString)** – Metoda przyswajająca agentowi wiedzę przekazaną przez rozmówcę. Dodaje do znanej mapy pola przekazane za pomocą **mapaString**.

## Klasa „ZbadanePole”

### Zmienne

* **int x, y** – Współrzędne pola.
* **int wysokosc** – Wysokość pola.
* **int energia** – Ilość energii w źródle energii na tym polu. Jeżeli jest to **-1**, oznacza to, że źródło to jest źródłem nieskończonym.
* **bool przeszkoda** – Wartość logiczna mówiąca o tym, czy pole te jest przeszkodą.
* **bool unrechable** – Wartość logiczna mówiąca o tym, czy pole jest uważane za nieosiągalne.

### Metody

* **void uaktualnijPole(int ileEnergii)** – Metoda uaktualniająca ilość energii obecną w źródle na danym polu.
* **void reached(int jakaWysokosc, int iloscEnergii, bool czyPrzeszkoda)** – Metoda uaktualniająca pole nieosiągalne. Zamienia pole nieosiągalne na pole poznane.
* **void pobierzEnergie(int ile)** – Metoda zmniejszająca ilość energii obecną na danym polu w naszej zapamiętanej mapie.

## Klasa „Wspolrzedne”

### Zmienne

* **int x, y** – Współrzędne w przestrzeni dwuwymiarowej.

# 5. Zawartość pliku AIML

<aiml>

<category>

<pattern>Stoj! \*</pattern>

<template>Moja mapa to:</template>

</category>

<category>

<pattern>Moja mapa to:\*</pattern>

<that>Stoj! \*</that>

<template>A moja mapa to:</template>

</category>

<category>

<pattern>\*</pattern>

<template>Nie rozumiem!</template>

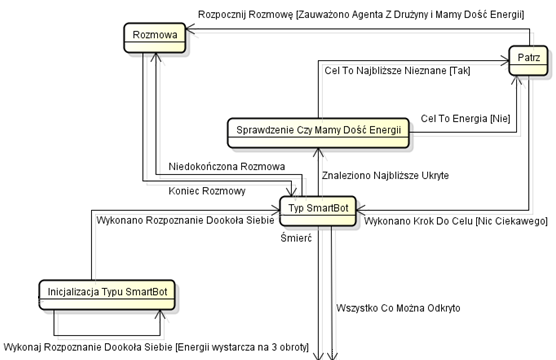
</category>

</aiml>

Ponieważ nasz agent może być zmuszony do naprawdę poważnego ograniczenia komunikacji z powodu na duże koszty ponoszone podczas wysyłania wiadomości, postawiliśmy na lakonizm w komunikacji. Nasz agent po zauważeniu kogoś ze swojej drużyny szybko wita się określając swoją pozycję aby rozmówca wiedział jak wysłać mu odpowiedź, po czym po odpowiedzi przesyła mu swoją mapę. Jeżeli w trakcie innych czynności agent zostanie powitany, przesyła rozmówcy swoją mapę i czeka na odpowiedź. Po wszystkim komunikacja się kończy. Ponieważ AIMLbot nie radzi sobie z obsługą plików aiml zawierających polskie znaki, nasz agent ich unika. Nasz plik **bot.aiml** można ujrzeć po prawej stronie tego tekstu.

# 6. Opis algorytmu przeżycia

W najprostszy sposób algorytm przeżycia naszego agenta można przedstawić za pomocą diagramu stanów:



Rysunek 2. Diagram Stanów przedstawiający algorytm przeżycia naszego agenta.

# 7. Wyniki testów

## Mapa typu „main”

## Mapa typu „mikro”

## Mapa typu „labirynt”

## Mapa typu „duża”

# 8. Dokumentacja użytkownika

## 1. Instalacja

### 1. Wersja normalna

Rozpakować do wybranego folderu archiwum „**VLuke\_JonesUserPack.rar**”.

### 2. Wersja developerska

Rozpakować do wybranego folderu archiwum „**VLuke\_JonesDevPack.rar**”.

## 2. Użytkowanie

### 1. Uruchamianie agenta

Uruchomić plik „**VLuke\_Jones.exe**”. Następnie należy wprowadzić adres serwera, nazwę grupy, hasło grupy, mapę docelową, oraz imię agenta. W razie popełnienia błędu należy powyższe dane wprowadzić jeszcze raz.

### 2. Obsługa agenta

Po pojawieniu się możliwości wyboru, należy wybrać tryb pracy agenta.

#### a)Klawiatura

Po wybraniu trybu ręcznego użytkownik steruje agentem. Agent sam zajmuje się obliczaniem energii, obserwowaniem pól oraz zapamiętywaniem mapy. Użytkownik może używać następujących klawiszy:

* **↑** - Pójdź agentem o jedno pole do przodu.
* **←** - Skręć agentem o 90° w lewo.
* **→** - Skręć agentem o 90° w prawo.
* **S** – wyświetl zbadaną mapę.
* **Enter** – Przechodzi w tryb wpisywania wiadomości. Po wpisaniu wiadomości ponowne wciśnięcie klawisza **Enter** wysyła ją.
* **Q** – Wychodzi z programu.

#### b)Agent automatyczny

Po wybraniu trybu „**Random**” lub „**SmartBot**” użytkownik może obserwować poczynania agenta na mapie udostępnianej przez **Atlantydę**. W momencie śmierci agenta lub zakończenia zadania, program wyświetli całą zapamiętaną przez agenta mapę i poprosi o wciśnięcie **Spacji**. Po wciśnięciu klawisza **Spacja** program kończy swe działanie.

### 3. Praca nad projektem

#### a) Ważne uwagi

Jeżeli masz zamiar zająć się rozwijaniem tego projektu, pamiętaj o skopiowaniu folderów „**aiml**” oraz „**config**” do każdego folderu ze skompilowaną wersją programu. Bez tego **AIMLbot** nie będzie poprawnie działać.

#### b) Kontakt do twórców

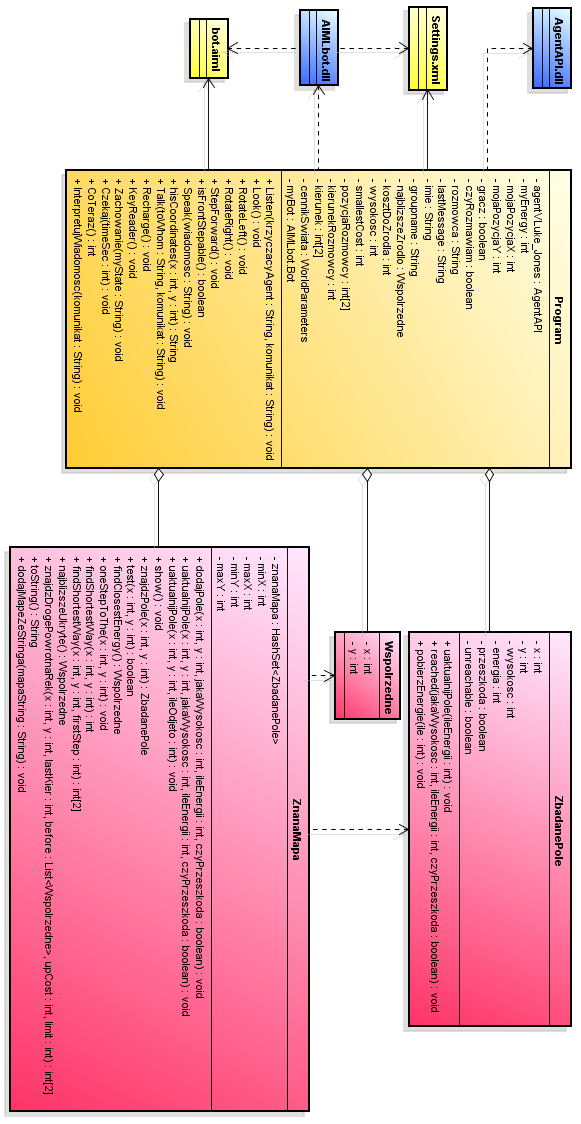
W razie problemów z obsługą lub rozwijaniem programu proszę pisać pod wybrany adres:

* dohacaha@poczta.fm
* g\_magdaa@op.pl
* jezus@g.pl
* ms91470@st.amu.edu.pl

# 9. Najczęściej Zadawane Pytania (wraz z odpowiedziami)

* Pytanie:
* Odpowiedź:
* Pytanie:
* Odpowiedź:

# Załącznik 1. Uproszczony diagram klas aplikacji.



# Załącznik 2. Uproszczony diagram stanów agenta.

