**Raport Końcowy**

**Inteligentne Techniki Obliczeniowe**

Bienias Krzysztof

Nr indeksu: 261235

Spis treści

**Treść projektu1**

**Wpisz tytuł rozdziału (poziom 1)4**

Wpisz tytuł rozdziału (poziom 2)5

Wpisz tytuł rozdziału (poziom 3)6

**Treść projektu1**

Wpisz tytuł rozdziału (poziom 2)2

Wpisz tytuł rozdziału (poziom 3)3

**Wpisz tytuł rozdziału (poziom 1)4**

Wpisz tytuł rozdziału (poziom 2)5

Wpisz tytuł rozdziału (poziom 3)6

**Treść projektu**

**Projekt L3 Rachunek sytuacyjny i wnioskowanie w przód.**

Wykonać program realizujący symulację środowiska i misję ratunkową autonomicznego

latającego drona (agenta) w nieznanym terenie, o cyfrowej postaci „N × N × H” „komórek”.

Celem działania agenta jest odnalezienie pilota i zrzucenie mu zaopatrzenia oraz powrót do

miejsca startu. „Zagrożenia” to radary i działa nieprzyjaciela – można je w uproszczeniu

modelować jako prostopadłościany o różnych rozmiarach i wysokości.

a) Wykonać symulator środowiska odpowiedzialny za dostarczanie obserwacji agentowi i za

śledzenie wykonywanych przez niego akcji. Agent „wyczuwa” radary i broń na kilka

kratek wprzód (im wyżej leci tym szybciej może lecieć, ale zużywa wtedy odpowiednio

więcej energii). Radary należy omijać w poziomie a wystrzały można ominąć w poziomie

albo wznieść się powyżej obserwowanego zasięgu działa. W celu zrzucenia zaopatrzenia

dron powinien obniżyć się do minimalnej wysokości Hmin = 0.1 H.

Ewentualny kontakt z radarem „unieruchamia” agenta na pewien czas, a trafienie

pociskiem zmniejsza jego maksymalną prędkość o połowę. Wynik polega na pomiarze

czasu od startu do powrotu przy uwzględnieniu faktu czy zadanie zostało wykonane czy

też nie.

b) Wykonać bazę wiedzy agenta. Wyrazić wiedzę agenta w rachunku sytuacyjnym języka

predykatów (można przyjąć własną implementację zapisu formuł). Zrealizować funkcje

TELL i ASK dla komunikacji funkcji głównej agenta z bazą wiedzy.

c) Zaimplementować ogólny (uniwersalny) algorytm wnioskowania w przód, w celu

wnioskowania o nowych faktach i wnioskowania o wyborze akcji.

d) Wykonać prosty interfejs graficzny. W osobnym okienku pokazywać wykonane

wnioskowania. Parametry N i H mają być zmienne w rozsądnych granicach.

e) Umożliwić wykonanie testowania działania programu dla różnych ustawień

początkowych świata. Wykonać testowanie i przeanalizować jego wyniki.

**Koncepcja projektu**

W realizacji projektu stworzony zostanie system ekspercki wykorzystujący bazę wiedzy

zbudowaną za pomocą klauzul Horna (posiadających jeden wniosek oraz mogących zawierać

wiele założeń). Klauzule te zostały wybrane ze względu na to posiadaną właściwość:

¬𝑝 ∨ ¬𝑞 ∨ . . .∨ ¬𝑡 ∨ 𝑢 ≡ (𝑝 ∧ 𝑞 ∧ . . . 𝑡) => 𝑢

Aby opisać poszczególne kroki posłużę się rysunkiem z wykładu:



Wykład ITO-mgr 1 strona 32

Dla właściwej oceny tego co się dzieje w środowisku agent posiada czujniki. Za ich pomocą możemy poczynić jakieś obserwacje, które z kolei są podstawą do tego, żeby aktualizować stan drona i wiadomości o tym środowisku. Na podstawie tego aktualnego stanu przy wykorzystaniu regułowej bazy wiedzy można wykonać odpowiednie wnioskowanie oraz wykonać jakieś akcje, które zależeć będą od aktualnego stanu środowiska.

Projekt ten rozpatruję w kategorii zmodyfikowanego 3-wymiarowego świata Wumpusa. Agent (dron) startować będzie z pozycji początkowej w terenie NxNxH komórek i będzie poruszał się w różnych kierunkach w celu odnalezienia pilota. Celem agenta będzie odnalezienie pilota i zrzucenie mu zaopatrzenia oraz powrót do miejsca startu. Agent za pomocą swoich czujników będzie wyczuwał zagrożenie (radary i broń) na kilka kratek w przód, aby móc je ominąć i dotrzeć do celu.

Obserwacje i akcje agenta będą następujące:

* Obserwacje: [ zagrożenie, unieruchomienie, zmniejszenie prędkości, dostosowanie wysokości]

- Agent wyczuwać będzie zagrożenie na kilka kratek w przód w każdym kierunku (będzie to uzależnione od podanego parametru)

- W przypadku gdy agentowi nie uda się ominąć radaru i jego tor ruchu będzie przechodzić przez kratkę z radarem będzie on unieruchomiony

- Gdy agent zostanie trafiony strzałą odczuwa zmniejszenie prędkości

- Gdy agent znajdzie się nad pilotem sprawdza swoją wysokość i jeśli nie jest na odpowiedniej wysokości to wykonuje korektę wysokości do poziomu 0.1H

* Akcje: { RuchWPrzód, RuchWTył RuchWLewo, RuchWPrawo, RuchWGóre, RuchWDół, Upuścić, Zginąć }

- ,,Upuścić” zaopatrzenie, jeśli dron znajdzie się nad pilotem

- ,,Zginąć” agent zginie, jeśli zostanie trafiony przez dwie strzały z rzędu

Dla pełnego wyjaśnienia schematu postępowania w przypadku algorytmu wnioskowania w przód wprowadzę następujące pojęcia:

**Baza wiedzy** składa się z reguł i faktów.

**Reguła** to wyrażenie postaci np. jeśli A to B ( A => B), gdzie A nazywamy **założeniem**, natomiast B(**wnioskiem**/konkluzją)

**Fakty** - są to wyrażenia logiczne, którym przypisano wartość logiczną.

Fakty mogą być **dopytywalne** bądź **niedopytywalne**. Fakty niedopytywalne to są takie, które nie występują jako wniosek.

**Predykat** wskazuje na konkretny obiekt( są to np. założenia, wnioski, fakty).

Dla przykładowej bazy wiedzy:

A => B

B => C

C jest faktem **niedopytywalnym** bo zawiera wniosek a z kolei B jest faktem **dopytywalnym** bo nie wynika z żadnej innej reguły.

Przy stworzeniu uniwersalnego algorytmu wnioskowania w przód będą wykorzystywane wprowadzone powyżej pojęcia:

* fakty **niedopytywalne**
* fakty **dopytywalne**

Dla przykładowej bazy wiedzy:

A => D

F, H => G

B => L

D, J => M

C, D => F

A, E => J

Musimy na początku ustalić, które fakty są dopytywalne, a które nie:

|  |  |
| --- | --- |
| **Fakty dopytywalne** | **Fakty niedopytywalne** |
| A | D |
| E | G |
| B | L |
| H | M |
| C | F |
|  | J |

Algorytm będzie po kolei przechodził przez każdą z reguł w bazie wiedzy i pobierał te predykaty, dla których znana jest wartość logiczna i mamy pewność że wartość ta się nigdy nie zmieni.

Warunki, jakie należy spełnić, aby móc zastosować dany algorytm:

- Trzeba znać wartości logiczne faktów dopytywalnych

- Będziemy ,,przeskakiwać” po kolejnych regułach (w przypadku braku pobrania kolejnego predykatu wartość pobrana z poprzedniej reguły przechodzi do kolejnej bez zmiany)

- Warunek stopu (pętla do while w której warunkiem jest to że nie dopiszemy żadnego predykatu po sprawdzeniu wszystkich reguł sprawdzając je kolejno w bazie wiedzy ).

Iteracja przejścia przez algorytm dla przykładowej bazy danych

|  |  |
| --- | --- |
| Krok | Wartość pobrana |
| A -> D | D |
| F, H -> G | (brak) a zatem zostaje D |
| B -> L | D L |
| D, J -> M | D L |
| C, D -> F | D L F |
| A, E -> J | D L F J |
| --------------------------------------------Kolejna iteracja----------------------------------------------- | |
| A -> D | D L F J |
| F, H -> G | D L F J G |
| B -> L | D L F J G |
| D, J -> M | D L F J G M |
| C, D -> F | D L F J G M |
| A, E -> J | D L F J G M |

Wnioskowanie wymaga 2 założeń

1. założenie zamkniętego świata (baza wiedzy, która zostanie stworzona nie będzie dynamiczna (nie będzie można dodać reguł podczas wnioskowania))

2. zasada poprawnego wnioskowania (reguła A =>B (jest spełniona dla par A,B))

|  |  |
| --- | --- |
| A | B |
| 1 | 1 |
| 0 | 1 |
| 0 | 0 |

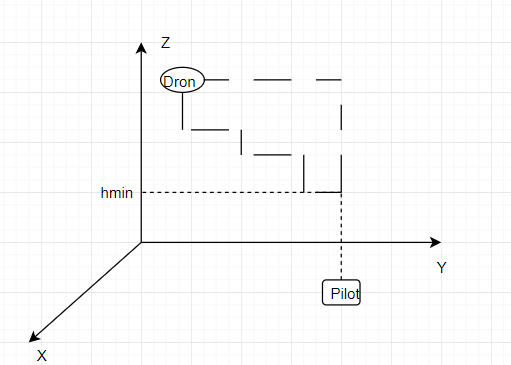
Przy założeniu dwóch wymagań jednocześnie (1 i 2) pozostaną nam pary **A B**

|  |  |
| --- | --- |
| A | B |
| 1 | 1 |
| 0 | 0 |

Wynika to z teorii wnioskowania. Na jedno zapytanie musimy otrzymać jedną odpowiedź.

Początkowo algorytm wnioskowania w przód będzie starał się znaleźć jak najkrótszą ścieżkę.

Poniżej przedstawiono przykładową sytuację:

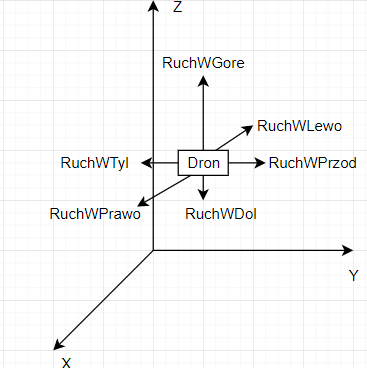


Liniami przerywanymi prowadzącymi od pilota został zaznaczony poziom hmin z którego możliwe jest zrzucenie zaopatrzenia przez agenta. Linie przerywane prowadzące od drona to przykładowe ruchy drona, które spowodują że znajdzie się on na określonym poziomie hmin. Istnieją co najmniej dwa sposoby podejścia do wyznaczenia najkrótszej ścieżki:

1. Agent(dron) dąży do tego aby znaleźć się nad dronem i wtedy zmniejsza swoją wysokość, aby zrzucić mu zaopatrzenie
2. Dron najpierw zniży się do poziomu hmin = 0.1H a następnie będzie próbował dostać się do pilota

Przez ścieżkę będę oznaczał sumaryczną liczbę ruchów jakie musi wykonać dron, aby przedostać się ze swojego położenia początkowego do pilota. W realizacji projektu wybrano 1 podejście. W przypadku gdy na najkrótszej ścieżce agent napotka jakieś przeszkody (radar, broń) ścieżka ta będzie modyfikowana z wykorzystaniem rachunku sytuacyjnego tak, aby uniknąć zagrożeń.

**Założenie początkowe**: dron będzie skierowany w kierunku dodatniej osi Y



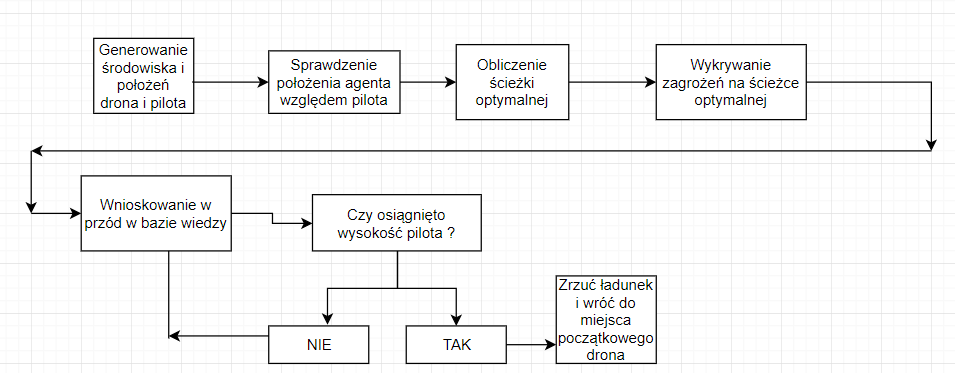
Ruch w kierunku osi X będzie to ruch w prawo lub w lewo

Ruch w kierunku osi Y będzie to ruch w przód lub w tył

Ruch w kierunki osi Z będzie to ruch w górę lub ruch w dół. Oś Z będzie wyznaczać wysokość.

**Ogólna struktura całego programu**

Ogólna struktura całego programu wygląda następująco:



Na początku działania programu generowane jest środowisko oraz położenia drona oraz pilota.

Odbywa się to za pomocą funkcji

**get\_simulation\_parameters**(drone, N\_max, N\_min, H\_max, H\_min, initial\_energy\_max, initial\_energy\_min, randomMode)

gdzie zmienne \*\_max oraz \*\_min są maksymalnymi i minimalnymi wartości N i H określające środowisko a także początkową energię drona.

RandomMode zostanie wyjaśnione na etapie GUI.

Zastosowano dwa mechanizmy generowania środowiska:

- (1) generowanie za pomocą funkcji losujących

- (2) generowanie poprzez wybranie ręcznie parametrów N i H. (wybrano granice **10-20** dla dobrego zobrazowania co się dzieje w środowisku)

1.Generowanie to odbywa się za pomocą funkcji losujących położenia początkowe drona oraz środowiska, pilota, rozmieszczenia gunów i radarów; realizowane poprzez wbudowaną funkcję Matlaba **round()**, która służy do zaokrąglania wartości. Realizowane jest to następująco:

%1. Parametryzacja parametrów H,N (wysokość, szerokość , głębokość)

H = round((H\_max - H\_min) \* rand(**1**)) + H\_min; %wysokość

N = round((N\_max - N\_min) \* rand(**1**)) + N\_min; %szerokość, głębokość

%Pozycja początkowa jako struktura (x,y,z)

initial\_position = struct;

initial\_position.x = round((N - **1**) \* rand(**1**)) + **1**; %9; 8; 10; 1;

initial\_position.y = round((N - **1**) \* rand(**1**)) + **1**; %9; 5; 4; 3;

initial\_position.z = round((H - **1**) \* rand(**1**)) + **1**; %4; 5; 6; 7;

drone.position = initial\_position; %inicjalizacja wartością początkową pozycji drona

drone.initial\_position = initial\_position;

initial\_speed = **1**; % inicjalizacja wartością początkową prędkości

initial\_energy = (initial\_energy\_max - initial\_energy\_min) \* rand(**1**) + initial\_energy\_min; % inicjalizacja wartością początkową energii

drone.speed = initial\_speed; %przypisanie do prędkości drona wartości zainicjalizowanej

drone.energy = initial\_energy; %przypisanie do energii drona wartości zainicjalizowanej

drone.initial\_energy = initial\_energy;

drone.sensors\_range = **1**; %zasięg czujnika (na ile komórek dron widzi w przód)

%pozycja pilota będąca strukturą (przechowującą współrzędne)

pilot\_position= struct;

pilot\_position.x = **3**;

pilot\_position.y = **5**;

pilot\_position.z = round(H/**10**); %obniżenie się do wysokości 0.1 H w celu zrzucenia zaopatrzenia

**if** (pilot\_position.z == **0**)

pilot\_position.z = **1**;

**end**

drone.pilot\_position = pilot\_position;

Aby zabezpieczyć się przed sytuacją, że wartości początkowego położenia dla jakiejkolwiek z osi wynosiłyby 0 dodano wartość 1 jako najmniejszą wartość jaką może mieć współrzędna.

Za pomocą funkcji **initialize\_guns\_and\_radars** możliwe jest dodawanie kolejnych przeszkód (radarów i gunów). Zostało to zrealizowane jako prostopadłościany o różnych długościach i wysokościach

Główna pętla programu opiera się o założenie, że energia drona jest > 0 oraz flaga sygnalizująca czy dron powrócił do miejsca początkowego jest nieustawiona (drone.if\_return\_to\_start=0)

Za pomocą analizy położenia drona i pilota wyliczana jest ścieżka optymalna (najkrótsza ścieżka łącząca drona i pilota). (odbywa się to w **calculate\_optimal\_path.m**)

Wyznaczanie ścieżki optymalnej odbywa się na zasadzie wprowadzenia priorytetów ruchu drona (agenta) w zależności od tego jaka jest różnica między położeniami drona i pilota na osiach X,Y,Z

y\_movement = pilot\_position.y - drone\_position.y; %ruch przod/tyl

x\_movement = pilot\_position.x - drone\_position.x;%ruch prawo/lewo

z\_movement = pilot\_position.z - drone\_position.z;%ruch gora/dol

Dla przypadku gdy wartość zmiennej y\_movement jest większa od zera mamy do czynienia z ustawieniem priorytetu ruchu w przód z ruchem w przód, w przeciwnym przypadku będzie to ustawienie priorytetu ruchu w tył.

Przykładowy zestaw ustawienia priorytetów:

**if** (y\_movement > **0**)

predicates.MF\_p.wartosc = true; %priorytet ruch w przod

**return**

**end**

**if** (y\_movement == **0** && x\_movement > **0**)

predicates.MR\_p.wartosc = true; %priorytet ruch w prawo

**return**;

**end**

**if** (y\_movement == **0** && x\_movement < **0**)

predicates.ML\_p.wartosc = true; %priorytet ruch w lewo

**return**;

**end**

**if** (y\_movement == **0** && x\_movement == **0** && z\_movement < **0**)

predicates.MD\_p.wartosc = true; %priorytet ruch w dol

**return**;

**end**

Priorytety ustawione są w następującej kolejności:

* Ruch na osi Y (przód, tył)
* Ruch na osi X (prawo, lewo)
* Ruch na osi Z (góra, dół)

Unikanie zagrożeń (skrypt **detect\_danger.m**) będzie odbywać się również na zasadzie wyznaczania priorytetów ruchu w zależności od tego w którym miejscu patrząc od strony agenta znajduje się zagrożenie.

Dla przykładu ruchu w przód realizowane jest to następująco:

RS = struct;

RS.nazwa = 'RS'; %'radar\_wykryty\_na\_wprost';

RS.wartosc = false;

**if** (optimal\_path.y > **0**) % optimal\_path.y to y\_movement

**for** i = **1**:drone.sensors\_range

**if** (drone.position.y + i <= N)

**if** (environment(drone.position.x , drone.position.y + i, drone.position.z) == 'r')

RS.wartosc = true;

**break**;

**end**

**end**

**end**

**end**

RS.jest\_ustawiony = true;

W przypadku gdy na optymalnej ścieżce agent wykryje radar (oznaczono jako ‘r’) ustawi wartość predykatu RS(Radar Straight) na wartość True

Do zapisu reguł w bazie wiedzy, które zostały rozróżnione na przesłanki i konkluzje użyto następujących skrótów:

**POP** – position over pilot (predykat oznaczający osiągnięcie wysokości pilota)

**DL** – drop load (predykat oznaczający zrzucenie ładunku)

**MF** – move forward (ruch w przod)

**RS** – radar straight (wykryto radar na wprost)

**MF\_p** – move forward priority (ustawiony jest priorytet do ruchu w przód)

**MB** – move backwards (ruch w tył)

**RR** – radar right (wykryto radar z prawej strony)

**RL** – radar left (wykryto radar z lewej strony)

**RU** – radar up (wykryto radar na górze)

**RD** – radar down (wykryto radar na dole)

**MR\_p** – move right priority (ustawiony jest priorytet do ruchu w prawo)

**MR** – move right (ruch w prawo)

**ML** – move left (ruch w lewo)

**MD** – move down (ruch w dol)

**MU** – move up (ruch w gore)

**ML\_p** – move left priority (ustawiony jest priorytet do ruchu w lewo)

Znajdują się także ich negacje poprzedzone przez **n**

Poniżej przedstawiono część reguł z bazy wiedzy:

baza\_wiedzy.reguly(**1**).przeslanki = [predicates.POP]; %wektor predykatow nF (not F)- zaprzeczenie

baza\_wiedzy.reguly(**1**).konkluzja = DL; % DropLoad - zrzuc ladunek

baza\_wiedzy.reguly(**2**).przeslanki = [predicates.nPOP]; %wektor predykatow nF (not F)- zaprzeczenie

baza\_wiedzy.reguly(**2**).konkluzja = nDL; % DropLoad - zrzuc ladunek

baza\_wiedzy.reguly(**3**).przeslanki = [predicates.nRS, predicates.MF\_p, nDL];

baza\_wiedzy.reguly(**3**).konkluzja = MF;

baza\_wiedzy.reguly(**4**).przeslanki = [predicates.nRS, predicates.MB\_p, DL];

baza\_wiedzy.reguly(**4**).konkluzja = MB;

baza\_wiedzy.reguly(**5**).przeslanki = [predicates.nRR, predicates.MR\_p];

baza\_wiedzy.reguly(**5**).konkluzja = MR;

baza\_wiedzy.reguly(**6**).przeslanki = [predicates.nRL, predicates.ML\_p];

baza\_wiedzy.reguly(**6**).konkluzja = ML;

baza\_wiedzy.reguly(**7**).przeslanki = [predicates.nRU, predicates.MU\_p];

baza\_wiedzy.reguly(**7**).konkluzja = MU;

baza\_wiedzy.reguly(**8**).przeslanki = [predicates.nRD, predicates.MD\_p];

baza\_wiedzy.reguly(**8**).konkluzja = MD;

Dla przykładowej reguły w bazie wiedzy:

baza\_wiedzy.reguly(**3**).przeslanki = [predicates.nRS, predicates.MF\_p, nDL];

baza\_wiedzy.reguly(**3**).konkluzja = MF;

Dla powyższego kodu należy to interpretować następująco:

Gdy nie wykryto radaru na wprost oraz predykat priorytetem jest ruch w przód jest ustawiony i nie zrzucono ładunku to wtedy idź w przód.

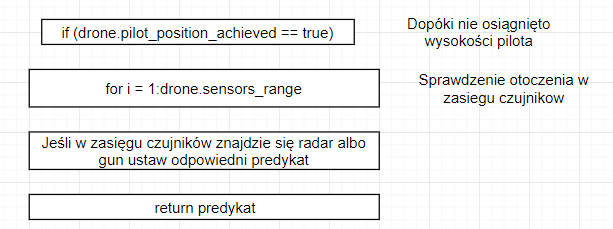
Aksjomatami w bazie wiedzy będzie zestaw następujących sytuacji:

- Położenie początkowe guna i radaru nie zmienia się w trakcie symulacji (po przejściu do kolejnego kroku są w tym samym miejscu)

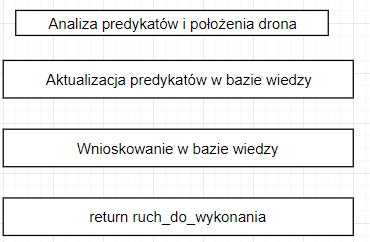
- Jeśli agent wejdzie w obszar oddziaływania radaru to zostanie on zamrożony na 3 kolejki

- Jeśli agent wejdzie w obszar oddziaływania guna to jego prędkość zostanie zmniejszona o połowę.

Funkcję **TELL** reprezentuje funkcja odpowiedzialna za pobranie aktualnych wartości predykatów oraz ich aktualizacje w podanej powyżej bazie wiedzy:



Funkcję **ASK** reprezentuje funkcja odpowiedzialna za pobranie aktualnych predykatów oraz położenia drona i poprzez wnioskowanie reguł w bazie wiedzy ustala kolejny ruch do wykonania przez agenta:



W momencie wyjścia z funkcji odpowiedzialnej za analizę bazy wiedzy sprawdzany jest jego otoczenie po wykonaniu jednego ruchu. W sytuacji gdy dron napotka po którejś ze stron (prawo, lewo, góra, dół, przód) jakąś przeszkodę to za pomocą reguł w bazie wiedzy próbuje ją ominąć. W przypadku gdy nie może tego zrobić kontakt z radarem powoduje zamrożenie jego ruchu na 3 kolejki.

**if** (freezeTime == **0**)

able\_to\_move = true;

freezeTime = **3**;

**else**

**if** (drone\_state.radar\_contact == true)

**if** (freezeTime == **0**) % to znaczy ze w poprzedniej iteracji sie ruszylismy

freezeTime = **3**;

**end**

freezeTime = freezeTime - **1**;

able\_to\_move = false;

**else**

freezeTime = **3**;

able\_to\_move = true;

**end**

**end**

natomiast kontakt z gunem powoduje zmniejszenie jego energii o połowę

**if** (drone\_state.gun\_contact == true)

drone.speed = round(drone.speed / **2**);

**if** drone.speed == **0**

drone.speed = **1**;

**end**

**end**

Użycie zapisu

**if** drone.speed == **0**

drone.speed = **1**;

**end**

Zabezpiecza, przed sytuacją kiedy prędkość drona byłaby 0( żeby agent nie stał w miejscu).

Dzięki zmiennej **able\_to\_move** możemy stwierdzić czy jest możliwe wykonanie ruchu analizując aktualne położenie drona.

W przypadku, gdy możliwe jest wykonanie ruchu wykonywana jest funkcja move\_drone:

**function** [drone] = **move\_drone**(drone, move\_to\_make)

drone.position.x = drone.position.x + move\_to\_make.x \* drone.speed;

drone.position.y = drone.position.y + move\_to\_make.y \* drone.speed;

drone.position.z = drone.position.z + move\_to\_make.z \* drone.speed;

**end**

Gdzie move\_to\_make jest wynikiem wnioskowania z bazy wiedzy

Po wykonaniu każdego ruchu wartość energii drona jest zmniejszana (take\_dron\_energy.m)

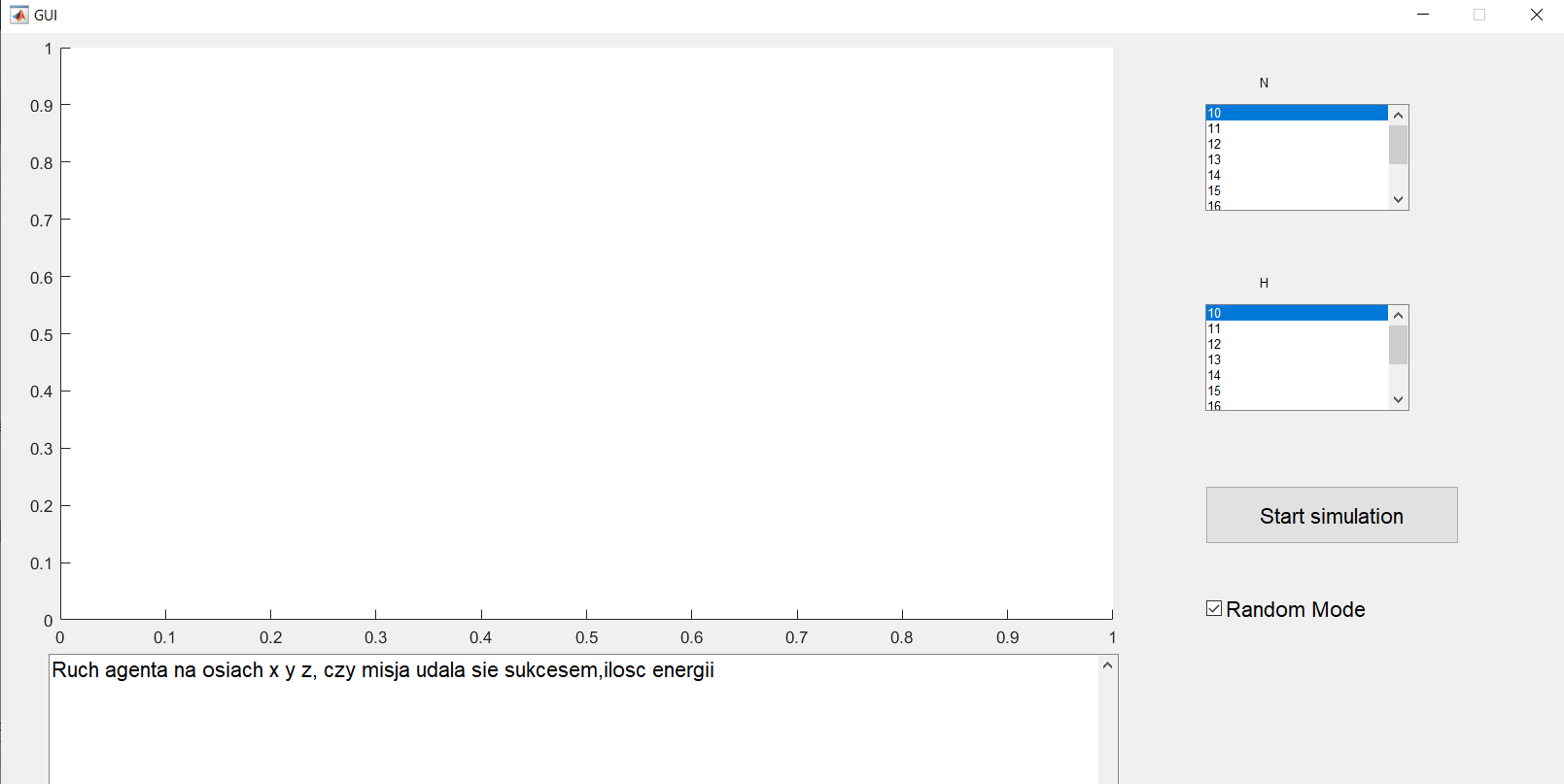
drone.energy = drone.energy - a \* drone.position.z;

gdzie a jest współczynnikiem. Przyjęto **a** = 0.1 aby energia drona zbyt szybko nie zmniejszała się.

W momencie gdy program wyjdzie z głównej pętli wyświetlane są wyniki za pomocą funkcji **plot\_simulation,** która to funkcja przy pomocy zbieranych wyników w każdej iteracji wyświetla aktualne środowisko, położenie drona oraz pilota, rozmieszczenie przeszkód a także drogę jaką pokonał dron od miejsca początkowego do pilota.

Zbieraniem wyników dotyczących analizy ruchu od drona do pilota oraz informacji o tym jakie są inicjalne wartości energii zajmuje się funkcja **analyze\_results**.

**Główne okno programu**:



Po prawej stronie znajdują się parametry **N** i **H** pozwalające na wybór wartości N i H w środowisku przed jego wygenerowaniem. Możliwy wybór to zakres 10-20 dla dobrego zobrazowania środowiska, drona, pilota oraz przeszkód w postaci radarów i gunów.

**Checkbox** Random Mode jest wstawiony dla zobrazowania dwóch możliwości testowania programu, za pomocą wbudowanych funkcji round() co jest możliwe w przypadku, gdy checkbox jest zaznaczony.

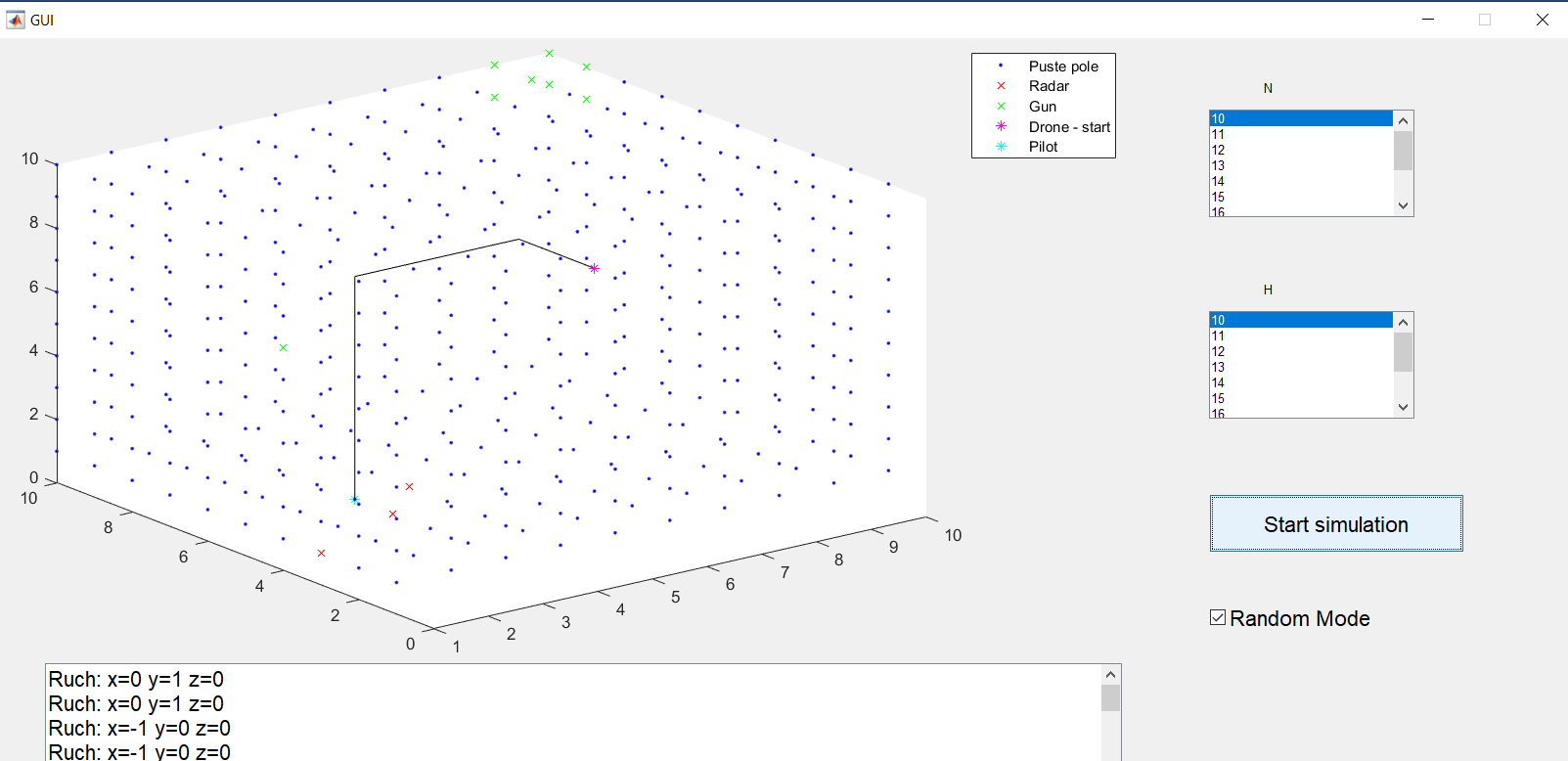
W przypadku braku zaznaczenia checkboxa Random Mode położenia początkowe będą mogły zostać manualnie ustawione przez użytkownika w skrypcie **get\_simulation\_parameters.m**

Przycisk Start simulation powoduje start symulacji z ustawionym środowiskiem.

Na wskazanym wyżej wykresie pojawiać się będzie środowisko, położenie drona, pilota, droga jaką przebył agent w celu dotarcia do pilota a także rozmieszczenie przeszkód.

Poniżej umieszczony będzie dokładny ruch agenta w kierunku osi X, Y, Z jakie od punktu startowego wykonał aby dojść do pilota oraz z powrotem do miejsca początkowego.

Dla przykładu:



Gdzie cała informacja poniżej zawiera poniższe informacje:

Ruch: x=0 y=1 z=0

Ruch: x=0 y=1 z=0

Ruch: x=-1 y=0 z=0

Ruch: x=-1 y=0 z=0

Ruch: x=-1 y=0 z=0

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=-1

Ruch: x=0 y=0 z=1

Ruch: x=0 y=0 z=1

Ruch: x=0 y=0 z=1

Ruch: x=0 y=0 z=1

Ruch: x=0 y=0 z=1

Ruch: x=0 y=0 z=1

Ruch: x=0 y=0 z=1

Ruch: x=1 y=0 z=0

Ruch: x=1 y=0 z=0

Ruch: x=1 y=0 z=0

Ruch: x=0 y=-1 z=0

Ruch: x=0 y=-1 z=0

Misja zakończona powodzeniem

Symulacja trwała: 25 iteracji

Średnia prędkość: 0.96

Energia początkowa: 27.6529 Pozostało energii: 13.3529

Testy: