VersaSim
Instrukcja użytkownika

Wprowadzenie - systemy wieloagentowe.

System wieloagentowe to zespół złożony z agentów, które się ze sobą komunikają i współpracują przy rozwiązywaniu zadanego problemu. Używa się ich na przykład do rozwiązywania problemów o naturze rozproszonej.

Pojedynczy agent, jest to jednostka zdolna do komunikacji z innymi agentami w swoim środowisku, zdolna do percepcji, a więc obserwowania swojego otoczenia i dokonywania decyzji, aby osiągnąć swoje cele, które zostały jej nadane.

Program VersaSim pozwala na tworzenie i przeprowadzanie symulacji wieloagentowych. Użytkownik może zdefiniować program wykonywany przez agentów za pomocą pseudo-kodu lub kodu w języku Java. Następnie ma możliwość budowy symulacji z agentów, obserwowania jej rozwoju i ingerencji w niego.

Cechy programu VersaSim.

- Program (cykl życia) agenta składa się z zachowań.
- Agent może mieć dowolną ilość zachowań.
- Symulacja dzieli się na cykle. Użytkownik ustala ile czasu na zegarze symulacji zajmuje jeden cykl.
- Podczas trwania jednego cyklu wykonywane są jednorazowo wszystkie aktywne zachowania wszystkich agentów. Następnie aktualizowany jest zegar symulacji o długość cyklu.
- Z punktu widzenia czasu symulacji wszystkie zachowania trwają tyle samo i wykonują się równocześnie.
- Ze względu na powyższe, szybkość mijania czasu symulacji uzależniona jest od obciążenia procesora, czyli m. in. od ilości działających agentów.
- Użytkownik ma możliwość ustalenia maksymalnej prędkości symulacji, która rozumiana jest jako stosunek mijającego czasu symulacji do użytego czasu procesora. Przykładowo, dla prędkości 3x, obliczone zostanie 150ms symulacji nie szybciej, niż w czasie rzeczywistym równym 50ms. Warto zaznaczyć, że zasoby obliczeniowe komputera będą ograniczać możliwość przyspieszenia.
- Zachowanie jest atomiczne, tj. jego wykonanie nigdy nie zostanie rozbite między cykle.
- Użytkownik wyznacza maksymalny czas procesora, jaki może być przeznaczony na obliczenie zachowań jednego agenta. Jeśli zostanie on przekroczony, agent zostaje unicestwiony.
- Symulacja przeprowadzana jest wielowątkowo. Użytkownik ma możliwość ustalenia liczby wykorzystywanych wątków.

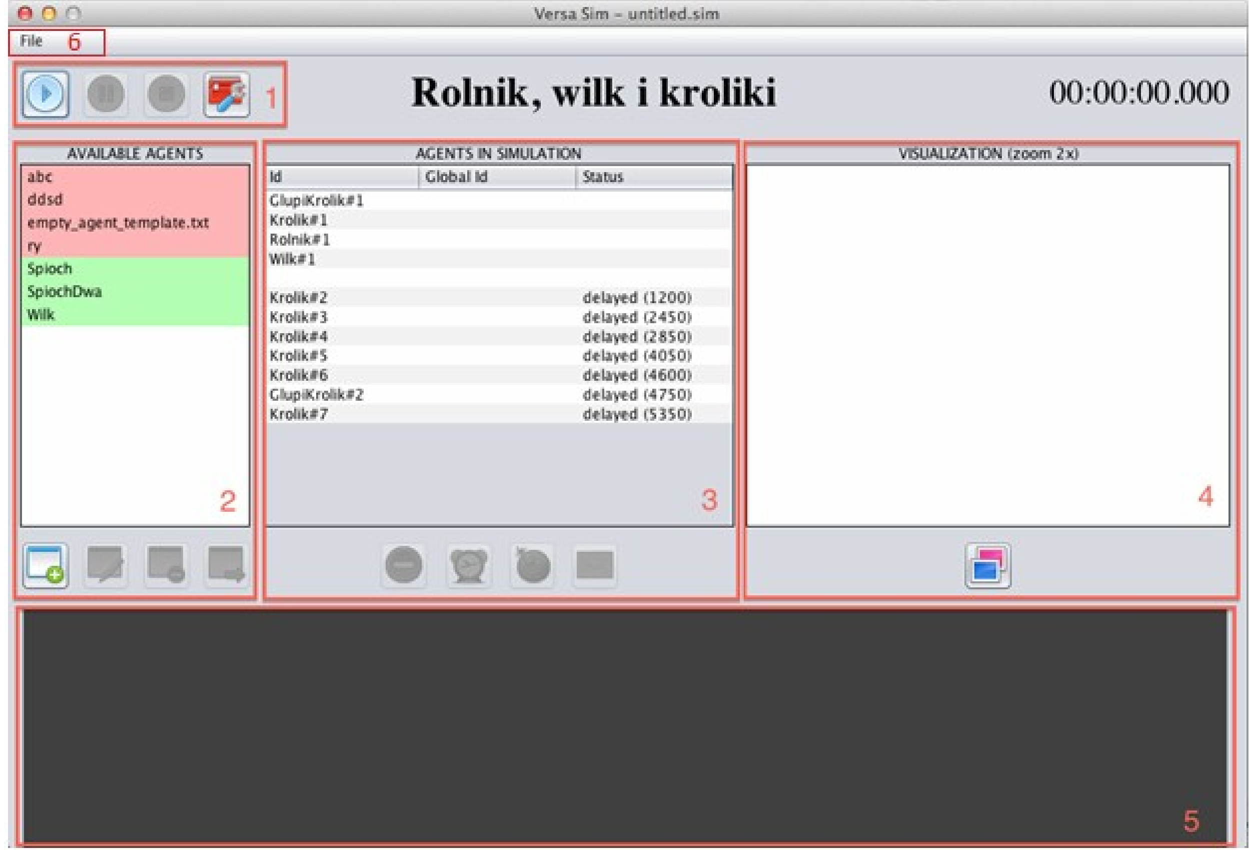
- Ilość wątków nie odpowiada ilości agentów.
- Zachowania jednego agenta w jednym cyklu wykonywane są zawsze przez jeden wątek.
- Ze względów bezpieczeństwa, w kodzie agentów zablokowane jest używanie funkcji operujących na zasobach systemu operacyjnego. Istnieje natomiast specjalny mechanizm, pozwalający na wczytywanie zawartości plików (np. obrazków) przed fazą wykonania zachowań.

Przykładowa symulacja

Aby ułatwić użytkownikowi rozpoczęcie pracy z programem, dodano opcję importowania przykładowej symulacji. Jest ona dostępna w menu Tutorial → Deploy example simulation. Wybranie tej opcji spowoduje skopiowanie plików agentów oraz wczytanie symulacji „Rolnik, wilk i króliki”. Z pewnością ułatwi ona obeznanie się z aplikacją i jest silnie polecana dla osób, które mają z nią pierwszy kontakt.

Instrukcja obsługi programu VersaSim

Po uruchomieniu aplikacji powinno wyświetlić się okno podobne do poniższego:



Rysunek : Główne okno programu

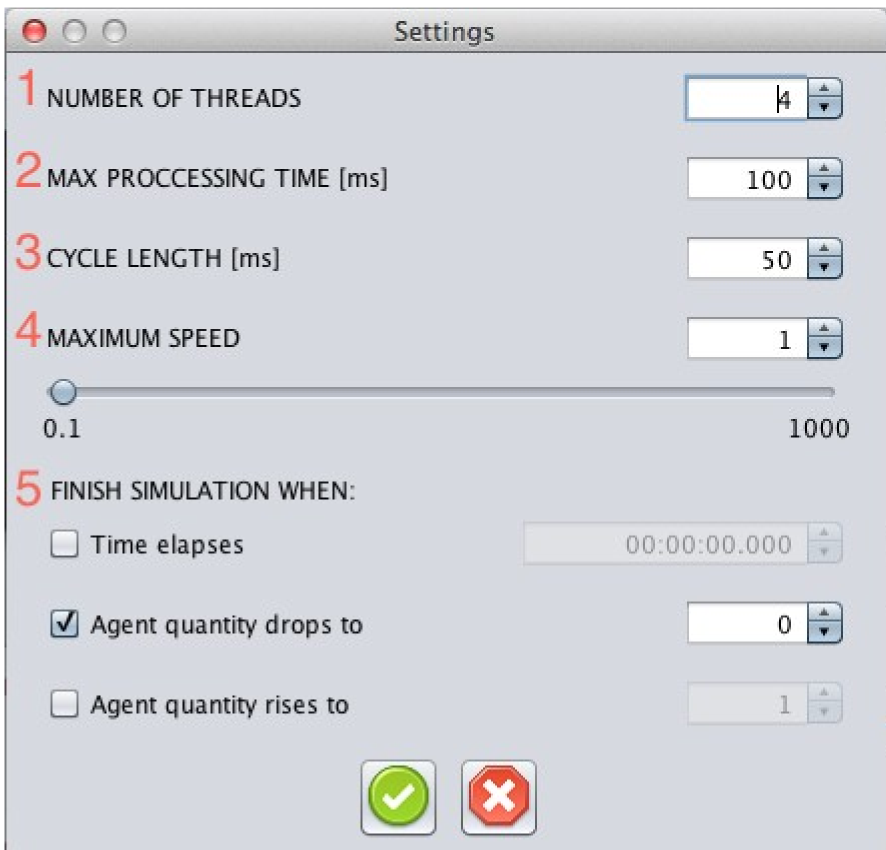
Główne okno aplikacji jest podzielone na 6 paneli:

- Główny panel kontroli symulacji
- Dostępne typy agentów, stworzone przez użytkownika
- Agenci biorący udział w symulacji
- Okno wizualizacji
- Konsola
- Menu

1. Główny panel kontroli symulacją.

Składa się on z 4 przycisków:

- Play - startuje (wznawia) symulację
- Pause - wstrzymuje symulację
- Stop (Reset) - kończy symulację
- Settings, po jego naciśnięciu wyświetla się nowe okno, z opcjami symulacji

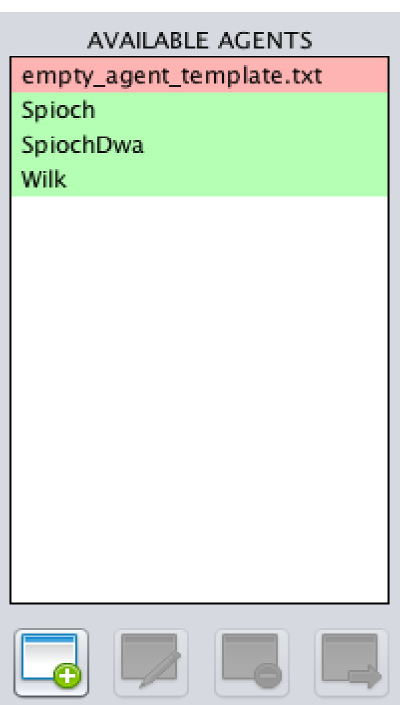


Rysunek : Ustawienia symulacji

Kolejne opcje oznaczają:

- Ilość wątków (Java Thread), które posłużą do przetwarzania symulacji
- Czas procesora, po którym agent zostanie unicestwiony jeśli jego zachowania dla jednego cyklu nie zostały policzone. Pozwala to na usunięcie m. in. "zapętlonych" agentów.
- Długość jednego cyklu - wyrażony w milisekundach czas cyklu symulacji. Gdy wszystkie aktywne zachowania każdego agenta zostaną wykonane (z punktu widzenia symulacji są wykonywane równolegle) - zegar zostanie zaktualizowany o tę wartość.
- Maksymalna prędkość symulacji - maksymalny stosunek mijającego czasu symulacji do czasu procesora.
- Warunek końca symulacji, do wyboru są 3 niewykluczające się możliwości:
    - gdy osiągnięty zostanie ustalony przez użytkownika czas symulacji
    - gdy ilość agentów spadnie do danej wartości
    - gdy wzrośnie do danej wartości

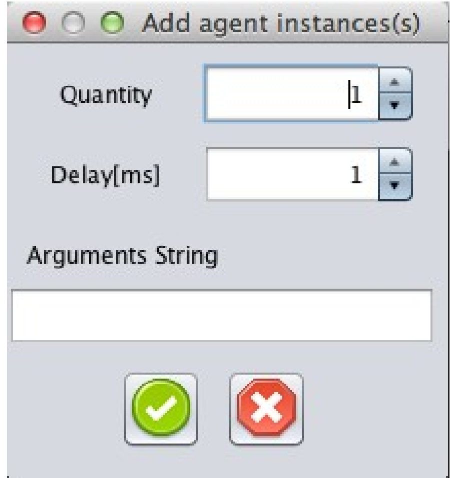
2. Available Agents.



Rysunek : Dostępni agenci

Drugi panel zawiera zdefiniowanych przez użytkownika agentów. Kolor zielony oznacza, że agent kompiluje się poprawnie i można dodać go do symulacji. Kolor czerwony informuje, że kod zawiera błędy.

Agentów można dodać do symulacji wybierając go z listy, a następnie klikając guzik . Wyświetli się dodatkowe okno, w którym można wybrać ilość dodawanych instancji agenta, opóźnienie w milisekundach po jakim zaczną funkcjonować oraz możliwość podania ciągu parametrów, z jakimi uruchomiony zostanie agent. Parametry (jeśli obecne) powinny być oddzielone spacjami i będą dostępne w kodzie agenta jako tablica Stringów.

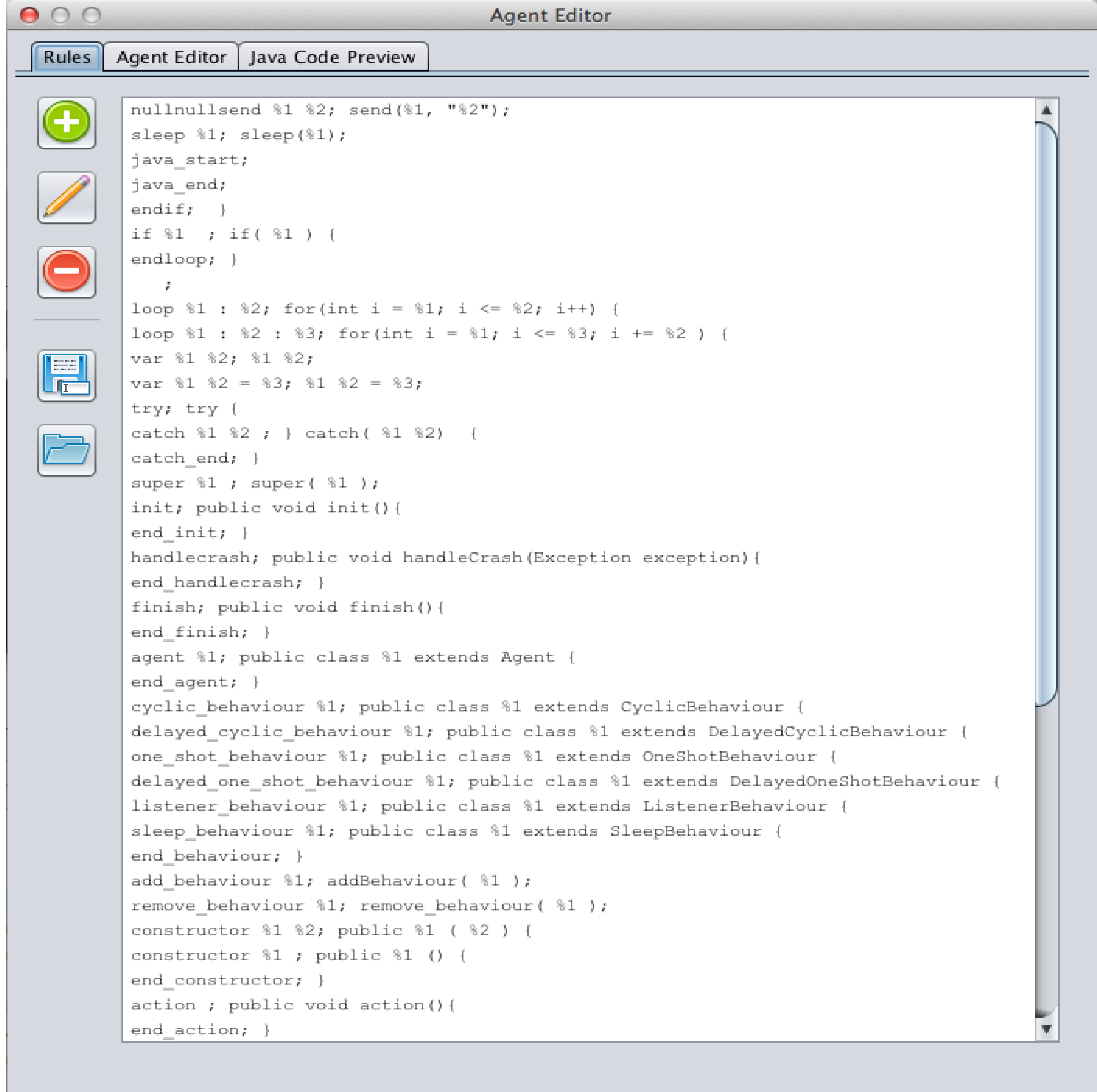


Rysunek : Opcje dodawania agentów do symulacji

W panelu drugim można również:

- Otworzyć edytor agentów z pustym agentem.
- Otworzyć edytor agentów w celu edycji wybranego z listy agenta.
- Usunąć wybranego agenta z dysku.

opcja pierwsza oraz druga spowoduje pojawienie się następującego okna:



Rysunek : Reguły parsera

Nowo wyświetlone okno, ma 3 zakładki:

- Rules
- Agent Editor
- Java Code Preview

W karcie **Rules** znajdują się reguły parsera, który służy do konwersji pseudokodu agenta do kodu w języku Java. Domyślnie zdefiniowane są reguły pozwalające na pełne wykorzystanie API agenta.

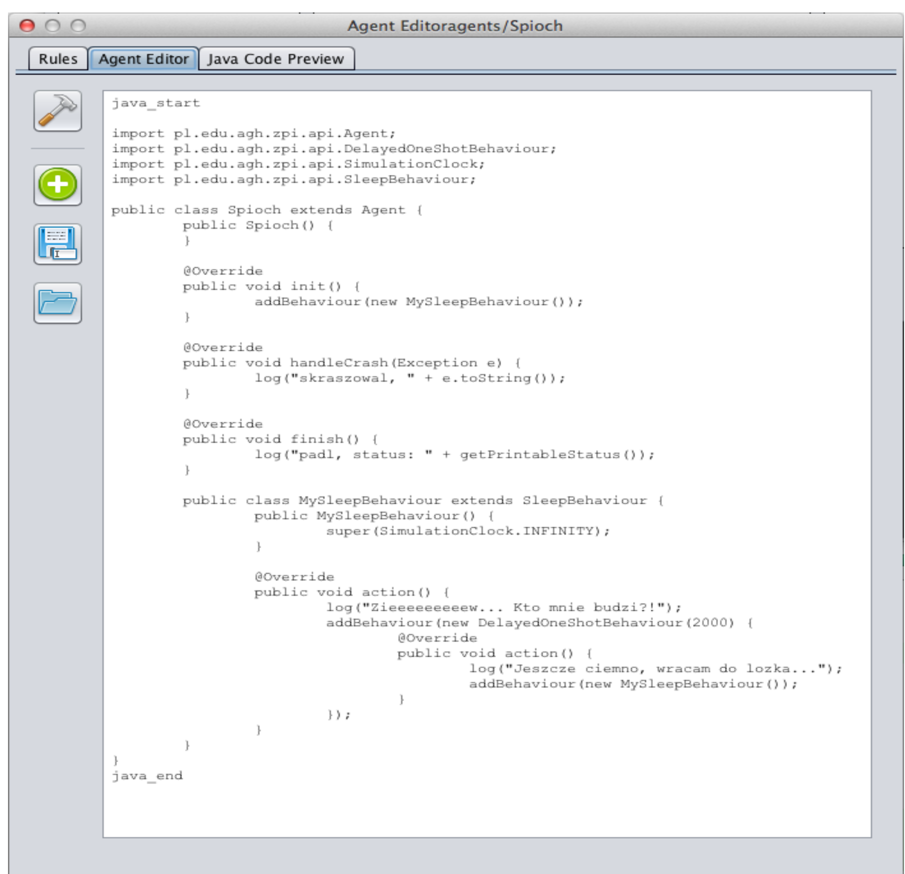
W miarę potrzeby, użytkownik może dodać , usunąć lub edytować reguły. Pojedyncza reguła składa się z 2 członów oddzielonych średnikiem “;”. Pierwszy człon to ciąg znaków w pseudokodzie. Drugi człon to odpowiednik w Javie. np.:

loop %1 : %2; for(int i=%1; i<=%2; i++)

oznacza to, że konstrukcja: loop 1 : 10

zostanie przekonwertowana do: for(int i=1; i<=10; i++)
znaczniki %1, %2, ... %n służą do podawania argumentów - podciąg znajdujący się w miejscu %1 w pseudokodzie będzie skopiowany w tej samej postaci do kodu wynikowego.
UWAGA: znaczniki typu %x zostaną dopasowane tylko do pojedynczych słów. Jeśli potrzebne jest wpisanie ciągu słów jako jednego argumentu, należy posłużyć się konstrukcją:{dlugi ciag znakow}.
Edytor reguł pozwala również na zapis oraz odczyt z pliku. Plik z domyślnymi regułami nazywa się **parser_rules.txt.**

Zakładka **Agent editor** służy do tworzenia kodu agenta z wykorzystaniem reguł zdefiniowanych w zakładce Rules. Użytkownik może posługiwać się kodem w czystym języku Java, jeśli zawrze go w znacznikach java_start oraz java_end.



Rysunek : Kod agenta napisany w javie

Dokumentacja API agentów w języku Java znajduje się w katalogu docs. Poniżej podsumowanie możliwości pseudojęzyka do tworzenia agentów, który korzysta z tego API.

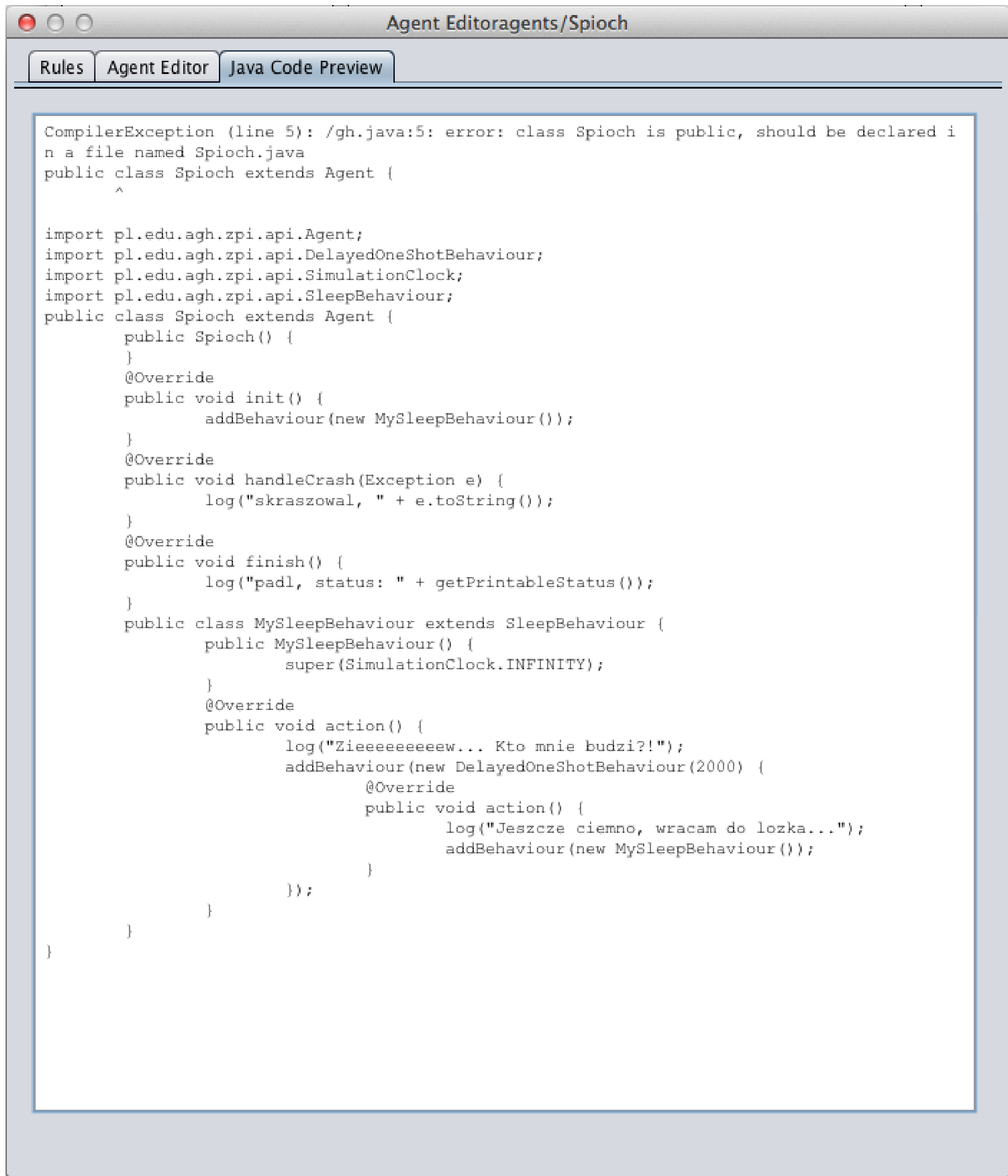
pseudokod	**kod w języku Java**	**opis**
java_start		początek bloku kodu w języku Java
java_end		koniec bloku kodu w języku Java
# %1		komentarz do kodu
if %1	if(%1) {	początek bloku instrukcji warunkowej %1 - warunek logiczny
else	} else {	instrukcja else
else if %1	} else if (%1) {	instrukcja else if %1 - warunek logiczny
end_if	}	koniec bloku instrukcji warunkowej
loop %1 : %2	for(int i = %1; i <= %2; i++) {	pętla z iteracją co 1 %1 - od wartość %2 - do wartości (włącznie)
loop %1 : %2 : %3	for(int i = %1; i <= %3; i += %2) {	pętla z iteracją o określoną wartość %1 - od wartość %2 - do wartości (włącznie) %3 - przeskok iteracji
foreach %1 %2 : %3	for (%1 %2 : %3) {	pętla typu foreach
endloop	}	zakończenie pętli
var %1 %2	%1 %2;	deklaracja zmiennej %1 - typ %2 - nazwa
var %1 %2 = %3	%1 %2 = %3;	inicjalizacja zmiennej %1 - typ %2 - nazwa %3 - wartość
set %1 %2	%1 = %2;	przypisanie wartości do zmiennej %1 - nazwa %2 - wartość
try	try {	początek bloku obsługi błędów
catch %1 %2	} catch(%1 %2) {	łapanie wyjątków %1 - typ %2 - nazwa zmiennej
end_catch	}	zakończenie przechwytywania wyjątków
init	public void init(){	deklaracja funkcji init() agenta
end_init	}	zakończnie deklaracji funkcji init()
handlecrash	public void handleCrash (Exception exception){	deklaracja funkcji handleCrash() agenta
end_handlecrash	}	zakończnie deklaracji funkcji handleCrash()
finish	public void finish(){	deklaracja funkcji finish() agenta
end_finish	}	zakończnie deklaracji funkcji finish()
load_resources	public void loadResources() {	deklaracja funkcji loadReasources(), służącej do wczytywania plików
load_image %1 %2;	loadImage(%2);	operacja wczytania obrazka z pliku
end_load_resources;	}	zakończenie deklaracji funkcji loadResources()
agent %1	public class %1 extends Agent {	deklaracja klasy Agent %1 - nazwa klasy agenta
end_agent	}	koniec deklaracji agenta
cyclic_behaviour %1	class %1 extends CyclicBehaviour {	deklaracja zachowania cyklicznego %1 - nazwa klasy
delayed_cyclic_behaviour %1	class %1 extends DelayedCyclicBehaviour {	deklaracja działania cyklicznego z opóźnieniem %1 - nazwa klasy
one_shot_behaviour %1	class %1 extends OneShotBehaviour {	deklaracja zachowania cyklicznego %1 - nazwa klasy
delayed_one_shot_behaviour %1	class %1 extends DelayedOneShotBehaviour{	deklaracja zachowania cyklicznego z opóźnieniem %1 - nazwa klasy
listener_behaviour %1	class %1 extends ListenerBehaviour {	deklaracja zachowania nasłuchiwania %1 - nazwa klasy
sleep_behaviour %1	class %1 extends SleepBehaviour {	deklaracja zachowania uśpienia %1 - nazwa klasy
end_behaviour	}	zakończenie deklaracji zachowania
add_behaviour %1	addBehaviour(%1);	dodanie nowego zachowania

		%1 - obiekt zachowania
add_anonymous_behaviour %1	addBehaviour(new %1() {	deklaracja i dodanie anonimowego obiektu zachowania
add_anonymous_behaviour %1 %2	addBehaviour(new %1(%2) {	deklaracja i dodanie anonimowego obiektu zachowania z parametrem
end_add_anonymous_behaviour	}	koniec deklaracji anonimowego zachowania
remove_behaviour %1	remove_behaviour(%1);	usunięcie istniejącego zachowania %1 - obiekt zachowania
constructor %1 %2	public %1 (%2) {	deklaracja konstruktora klasy %1 nazwa klasy %2 - parametr
constructor %1	public %1 () {	deklaracja konstruktora klasy bez parametrów %1 nazwa klasy
end_constructor	}	zakończenie konstruktora
super %1	super(%1);	konstruktor klasy bazowej z parametrem %1
super %1 %2	Super(%1, %2);	konstruktor klasy bazowej z parametrami %1, %2
super %1 %2 %3	Super(%1, %2, %3);	konstruktor klasy bazowej z parametrami %1, %2, %3
action	public void action(){	deklaracja kodu zachowania
end_action	}	zakończenie kodu zachowania
send %1 %2	send(%1, %2);	wysyła wiadomość %2 (String) do agenta o ID %1 (String)
register %1	register(%1);	rejestruje agenta pod globalnym id %1 (String) - globalne id
unregister	unregister();	zwalnia globalne id
receive %1	receive(%1);	odbiera wiadomości z kolejki wiadomości %1 - maksymalna ilość wiadomości
receive %1 %2	receive(%1 , %2);	odbiera wiadomości z kolejki wiadomości %1 - maksymalna ilość wiadomości %2 (String) - filtr wiadomości (wyrażenie regularne)
wake_up %1	wakeUp(%1);	sygnał budzenia do agenta %1 (String) - id agenta
kill %1	kill(%1);	sygnał unicestwienia do agenta %1 (String) - id agenta
get_args	getArgs();	pobiera argumenty, z jakimi stworzony był agent (w formie tablicy String'ów)
console_log %1	consoleLog(%1);	wypisuje wiadomość na konsolę (domyślnie dokleja czas symulajci i id agenta) %1 (String) - treść wiadomości
console_log %1 %2 %3	consoleLog(%1, %2, %3);	wypisuje wiadomość na konsolę %1 (String) - treść wiadomości %2 - true/false - czy dokleić czas symulacji %3 - true/false - czy dokleić id agenta
file_log %1	fileLog(%1);	loguje wiadomość do pliku (domyślnie dokleja czas symulajci i id agenta) %1 (String) - treść wiadomości
file_log %1 %2 %3	fileLog(%1 , %2 , %3);	loguje wiadomość do pliku %1 (String) - treść wiadomości %2 - true/false - czy dokleić czas symulacji %3 - true/false - czy dokleić id agenta
log %1	log(%1);	wypisuje wiadomość na konsolę (domyślnie dokleja czas symulajci i id agenta) %1 (String) - treść wiadomości
log %1 %2 %3	log(%1 , %2 , %3);	loguje wiadomość do pliku i na konsolę %1 (String) - treść wiadomości %2 - true/false - czy dokleić czas symulacji %3 - true/false - czy dokleić id agenta
set_visual_marker %1	setVisualMarker(%1);	ustawia znacznik wizualizacji %1 - obiekt znacznika
get_visual_marker	getVisualMarker();	zwraca aktualny znacznik lub null
get_id	getId();	zwraca id agenta (globalne, lub w

		przypadku jego braku unikalne)
get_global_id	getGlobalId();	zwraca globalne id agenta
get_unique_id	getUniqueId();	zwraca unikalne id agenta
get_status	getStatus();	zwraca aktualny status agenta
get_printable_status	getPrintableStatus();	zwraca status agenta w formie String'a

Tabela : API Agentów

Java Code Preview zapisuje i prezentuje przetłumaczony do Javy kod z zakładki Agent Editor. Jeśli występują w nim błędy, są one wyświetlane na początku kodu.



Rysunek : Przetłumaczony kod agenta

3. Agents in Simulation.

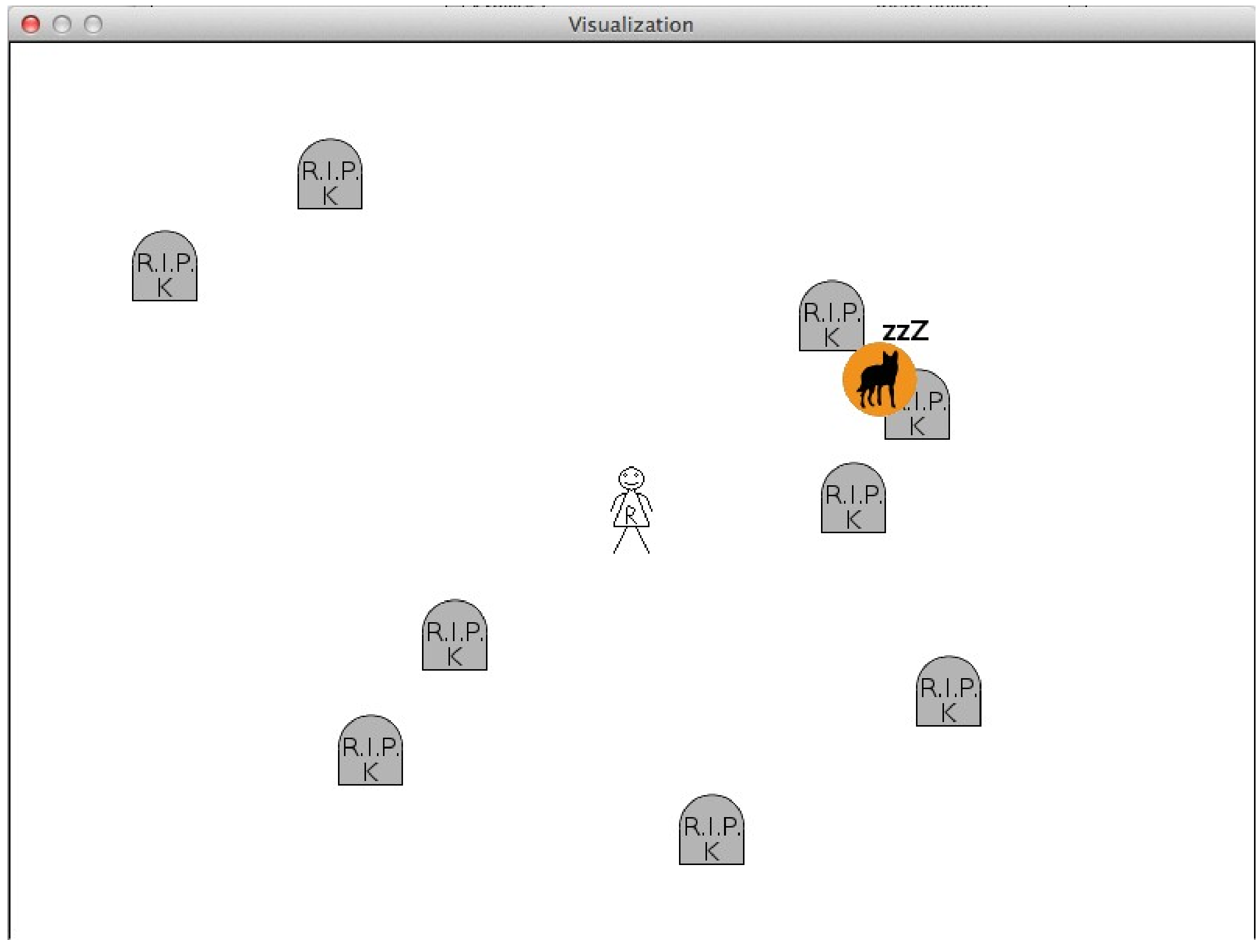
AGENTS IN SIMULATION

Id	Global Id	Status
GlupiKrolik#1		
Krolik#1		
Rolnik#1		
Wilk#1		
Spioch#1		delayed (51)
Krolik#2		delayed (1200)
Krolik#3		delayed (2450)
Krolik#4		delayed (2850)
Krolik#5		delayed (4050)
Krolik#6		delayed (4600)
GlupiKrolik#2		delayed (4750)
Krolik#7		delayed (5350)

Rysunek : Agenci uczestniczący w symulacji

Panel ten prezentuje agentów biorących udział w symulacji, podzielony jest na 3 sekcje. Pierwsza prezentuje aktywnych agentów, druga opóźnionych, natomiast trzecia nieżywych. Przed uruchomieniem symulacji można usunąć zaznaczonego agenta za pomocą przycisku . Po uruchomieniu, w trakcie symulacji można wysłać do zaznaczonego agenta sygnał budzenia , sygnał unicestwienia lub wiadomość . Sygnał budzenia zadziała jedynie, gdy agent aktualnie jest uśpiony. Sygnał unicestwienia zadziała zawsze. Wiadomość będzie dołączona do kolejki wiadomości agenta, ale musi zostać odebrana przez agenta, żeby do niego dotarła. Wszystkie sygnały przetworzone będą w kolejnym cyklu, tak że jeśli wysyłane są podczas pauzy symulacji, ich efekt widoczny będzie dopiero po jej wznowieniu.

4. Wizualizacja.



Rysunek : Wizualizacja agentów Wilka, Rolnika i Królików

Ważnym w symulacji panelem jest panel Visualization. W głównym oknie programu dostępny jest podgląd w zbliżeniu 2x, natomiast klikając przycisk mo-żna otworzyć zewnętrzne okno wizualizacji w pełnym wymiarze. Zawartość okna wizualizacji jest w pełni programowalna przez użytkownika przy użyciu funkcji setVisualMarker().

5. Konsola.

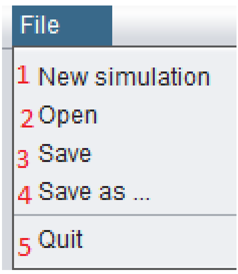
Ostatnim elementem GUI jest konsola. Oprócz podstawowych wiadomości diagnostycznych znajdą się tutaj wszystkie wiadomości, które zostały zalogowane w kodzie agentów.

```
[00:00:00.000] SIMULATION STARTED
[00:00:00.000](          Rolnik) melduje sie na farmie.
[00:00:00.000](        Krolik#1) melduje sie na farmie.
[00:00:00.000](            Wilk) melduje sie na farmie.
[00:00:00.000](   GlupiKrolik#1) melduje sie na farmie.
[00:00:01.050](        Krolik#1) starzeje sie... mam 2 lat
[00:00:01.050](   GlupiKrolik#1) skraszowal, java.lang.ArithmeticException: / by zero
[00:00:01.050](   GlupiKrolik#1) padl, status: dead (crashed)
[00:00:01.100](          Rolnik) 100 lat niech zyje Krolik#1
[00:00:01.200](        Krolik#2) melduje sie na farmie.
[00:00:02.100](        Krolik#1) starzeje sie... mam 3 lat
```

Rysunek : Okno konsoli

6. Menu.

Aplikacja posiada także swoje menu, dostępne pod przyciskiem "File", w lewym górnym rogu okna. Pojawia się wówczas:



Rysunek : Menu

Widocznych jest kilka opcji:

- **New simulation** - tworzy nową symulację, jednak jeśli nie została ona wcześniej zapisana, pojawia się okno pytające czy ją zapisać.
- **Open** - otwiera zapisaną wcześniej symulację.
- **Save** - zapisuje aktualną symulację nazwą domyślną. Zapis i odczyt jest możliwy tylko przed startem bądź też po resecie symulacji. Zapisana zostanie konfiguracja symulacji oraz lista agentów wraz z parametrami.
- **Save as** - zapisuje symulację w określonym przez użytkownika miejscu, domyślnie jest to folder "simulations", ważne jest także rozszerzenie, otworzyć będzie można jedynie te symulacje, które mają rozszerzenie **.sim**.
- **Quit** - kończy działanie programu.

Format zapisu symulacji

Symulacja zapisywana jest do pliku w czytelnym dla człowieka formacie. Pierwsze siedem linijek tekstu zajmuje konfiguracja symulacji, następnie, po linijce przerwy, znajdują się wpisy odpowiadające agentom w symulacji. Każda linijka ma następujący format:

NazwaAgenta<spacja>*Opóźnienie*<spacja>*Argumenty* (opcjonalne).

Dopuszczalne jest ręczne wprowadzenie do pliku listy agentów, jeśli tylko spełnia ona powyższe warunki. W ten sposób można definiować symulacje z większą ilością agentów, np. za pomocą skryptu.

Struktura katalogów po instalacji

W głównym folderze instalacji znajduje się wykonywalny plik JAR oraz dokumentacja – instrukcje użytkowania oraz pliki *javadoc* z dokumentacją API oraz aplikacji.

Po pierwszym uruchomieniu aplikacji, w katalogu użytkownika zostanie stworzony folder „**versasim**”, w którym znajdą się trzy podfoldery:

- **agents –** tutaj zapisywane są pliki z kodem agentów. Tylko ten folder będzie przeglądany w poszukiwaniu tych plików, więc nie należy ich zapisywać gdzie indziej. Uwzględniane będą tylko pliki z rozszerzeniem .agt.
- **simulations –** tutaj zapisywane będą pliki symulacji. Nie jest wymagane, aby znajdowały się one koniecznie w tym miejscu, ale bardzo ułatwi to zarządzanie plikami.
- **logs –** tutaj znajdą się logi wygenerowane podczas symulacji.
- **Images –** tutaj należy umieszczać obrazki, które będą używane w symulacji. Ścieżki do plików podawane w kodzie agentów będą uznawane za względne do folderu images.

Autorzy:
Opioła Łukasz
Trzeciak Piotr
Reinstein Bazyl
Michałek Marcin
Pałucki Tomasz
Czajkowki Jakub
Goliński Łukasz
Gwoździewicz Adrian