Radosław Ilnicki
Tomasz Kurowski
Arkadiusz Lisiecki
Paweł Pańków
Przemysław Sperzyński

Raport końcowy projektu:
Symulator zachowania w grupie, gra poznawcza

Wrocław, 21 stycznia 2008

1 Wstęp

W niniejszym dokumecie, zostaje podsumowana całość prac nad projetem. Zostanie przedstawiony schemat aplikacji, opis zaimplementowanych algorytmów oraz GUI.

Przekazane zostają:

- sprawozdanie podsumowujące
- program (na CD, źródła, statyczne kompilacje Linux, Windows)
- zrzut strony web (na CD)
- instrukcja użytkownika (na CD)

2 Opis projektu

Produktem końcowym projektu jest gra-symulator mająca naśladować zachowania zwierząt stadnych (drapieżników i ofiar) lub też zachowania pojedynczego osobnika w grupie w zależności od przyjętego 'charakteru'. Aplikacja jest kierowania głównie do młodzieży i przedstawia model robota jako zwierzę. Program pozwala użytkownikowi na eksperymentowanie z wielkościami fizycznymi takimi jak: prędkość, przyspieszenie, energia oraz pozwala na obserwację, jak wybór strategii ma wpływ na zachowanie osobników.

Program jest napisany w oparciu o dwa programy pisane w ramach studiów na kierunku eka-AiR na PWr. Bezinterfejsowy symulator zachowania drapieżników i ofiar (algorytmy pogoni i ucieczki) oraz graficzną prezentację zachowania się osobników (algorytm pogoni i ucieczki) oraz graficzną prezentację zachowania się osobników. Szczegółowe raporty z prac nad wymienionymi projektami źródłowymi znajdują się w repozytorium CVS w module *Inwentaryzacja*.

Stworzone narzędzie, to program komputerowy napisany w języku C++, z użyciem biblioteki QT4 dla potrzeb interfejsu i prezentacji graficznej zagadnienia. Program kompiluje się zarówno w środowisku Windows jak również w systemie operacyjnym Linux.

Licencja programu: GNU/GPL.

2.1 Budżet i użyte narzędzia

Projekt został w całości napisany przy użyciu darmowego oprogramowania i prywatnego sprzętu komuterowego.

Narzędzia użyte podczas prac nad projektem:

- kompilator: GNU gcc 4.1
- biblioteka QT: 4.3.3
- edytory: vim, kwirte, SciTE
- wspomaganie monitoringu prac: GNOME Planner
- zdalne repozytorium CVS (*http://sourceforge.net/zwierzaki*)
- generator dokumentacji technicznej: Doxygen
- edytor schematów UML: ArgoUML
- inne: OpenOffice.org, GIMP

Każdy z członków zespołu został zobowiązany samodzielnie zaopatrzyć się w niezbędne narzędzia do wykonania projektu.

2.2 Zespół projektowy

Skład zespołu projektowego to pięciu specjalistów:

- Radosław Ilnicki – tester, poszukiwacz algorytmów
- Tomasz Kurowski – projektant-analityk
- Arkadiusz Lisiecki – lider, programista
- Paweł Pańkow – dokumentujący, poszukiwacz algorytmów
- Przemysław Sperzyński – główny programista

którzy stanowią *de facto* złączenie doświadczeń dwóch grup pracujących nad programami wspomnianymi w opisie (por. 2), tj. nad interfejsem graficznym i algorytmami pogoni/ucieczki.

Każdy z członków zespołu potrafi posługiwać się systemem kontroli wersji CVS.

3 Aplikacja

Aplikacja spełnia przyjęte założenia implementacyjne, jest możliwie obiektowa, modułowa oraz zawiera komentarze interpretowane przez Doxygen. Nazwy zmiennych, metod i klas są w języku angielskim natomiast komentarze w języku polskim. W niniejszym dokumencie zostanie przybliżona struktura oraz działanie aplikacji. Dokumentacja techniczno-implementacyjna stanowi oddzielne opracowanie i jest przeznaczona dla developerów bądź osób pragnących szczegółowo zgłębić działanie aplikacji.

Schemat ideowy aplikacji:

3.1 Core Classes

Podstawowe klasy działania programu, zostały zgrupowane pod wspólną nazwę *Core Classes*. Należą do nich:

- Animal
- Obstacle
- Behavior

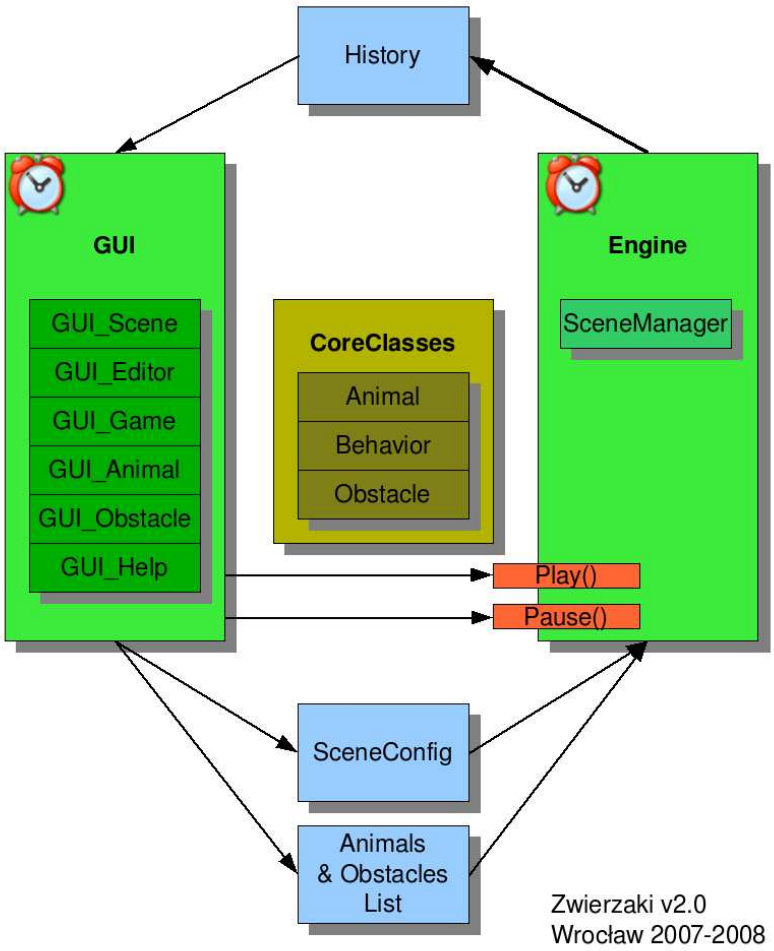
Powyższe klasy bezpośrednio przekładają się na poziom uwzględnianych szczegółowości w symulacji. To właśnie w tych klasach ustala się m.in. liczbę parametrów zwierzęcia (*Animal*), rodzaje przeszkód (*Obstacle*), algorytmy zachowań (*Behavior*).

W programie zaimplementowano tylko jeden rodzaj przeszkody i zamknięto ją w klasie: *Obstacle_Circle*. Klasa musi obsłużyć następujące metody wirtualne:

- Vector2D **getCross**(Vector2D &vec, double orient); - pobiera punkt przecięcia z przeszkodą
- double **getAngle**(Vector2D &vec, double orient); - wyznacza kąt 'natarcia' na przeszkodę

Klasa *Animal* jest dość rozbudowana, składa się z dwóch podklas:

- *Animal_* - podstawowe parametry zwierzęcia, bez uwzględniania strategii i hierarchi w stadzie
- *AnimalParam* - bieżące parametry zwierzęcia, zmieniające się w trakcie symulacji



Rysunek 1: Schemat ideowy aplikacji

Klasa *Behavior* sama w sobie nie zawiera żadnej strategii, stanowi pewne zestandaryzowanie, do którego muszą się dostosować wszystkie algorytmy mające być zaimplementowane w aplikacji. Kluczową metodą jest:

- AnimalParam **action**(Animal_ &anim, list¡Animal_¿ &family, list¡Animal_¿ &opponent, Obstacles &obs);

Powyższa metoda otrzymuje od Managera Sceny tyle informacji, na ile po-

zwala percepcja zwierzęcia, na tej podstawie algorytm musi podjąć decyzję.

3.2 Zaimplementowane algorytmy

Rodzicem wszystkich zaimplementowanych algorytmów jest klasa *Behavior*. Zosta/ly przeniesione wszystkie algorytmy z poprzednich wersji aplikacji. Przyjęto, że nazwy plików algorytmów zachowań (strategie), zaczynają się od 'b_', natomiast nazwy klas od 'B_'. W aplikacji działa 10 algorytmów, z czego 3 stanowią algorytmy podstawowe, 6 algorytmów pogoni/ucieczki oraz 1 omijania przeszkód (o zmiennym priorytecie). Bardziej szczegółowe opisy algorytmów znajdują się na stronie WWW projektu bądź w komentarzach w każdej z zaimplementowanych klas.

3.2.1 Algorytmy podstawowe

(*Zaimplementowane w jednym z projektów źródłowych [1], ale przepisane i dostosowane do nowego silnika*):

- B_Wander - chaotyczne ruchy
- B_Cohesion - 'grupowanie'
- B_Separation - odizolowanie

3.2.2 Algorytmy pogoni i ucieczki

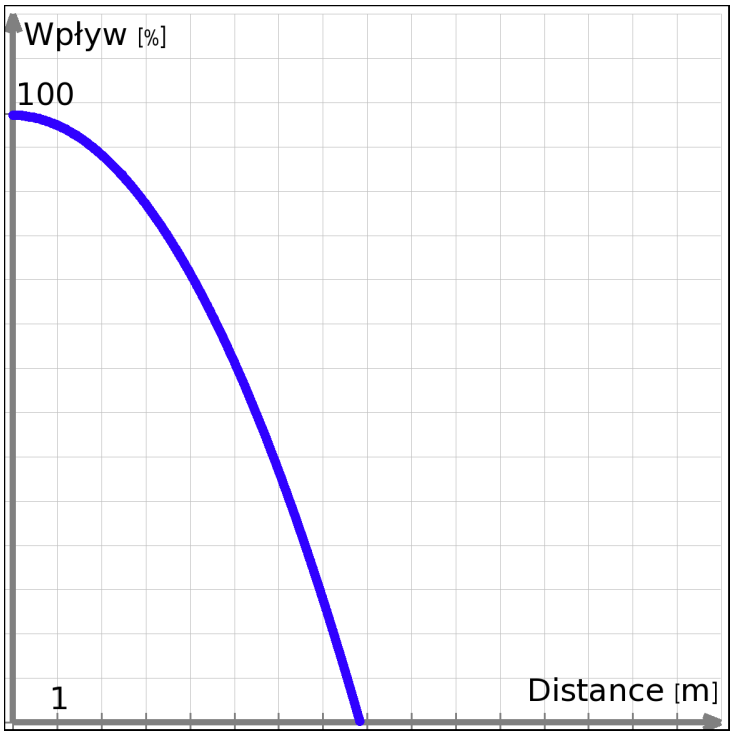
- B_Encirclement - pogoń za pierwszą ofiarą na liście (w kierunku ofiary)
- B_EncirclementLux - pogoń za pierwszą ofiarą na liście (w kierunku ofiary), wersja z modyfikacją
- B_Tommy - pogoń za pierwszą ofiarą na liście (w kierunku punktu przecięcia z ofiarą)
- B_TommyLux - pogoń za pierwszą ofiarą na liście (w kierunku punktu przecięcia z ofiarą), wersja z modyfikacją
- B_FlightMulti - ucieczka jak najdalej od wszystkich drapieżników z kryterium hiperbolicznym
- B_Flee - ucieczka (kryterium średniego kwadratu odległości)

Modyfikacja polega na tym, że do wyliczonej, nowej orientacji, dodaje się 0.35 rad (ze znakiem +, gdy ofiara jest po lewej stronie drapieżcy, ze znakiem -, gdy ofiara jest po prawej). Pozwala to uzyskać ciekawy efekt 'osaczenia'.

3.2.3 Algorytm omijania przeszkód

Algorytm omijania przeszkód, jest zaszyty w swego rodzaju 'podświadomości' (tzn. jest uruchamiany i uwzględniany przez Managera Sceny).

- B_AvoidObstacles - omijanie przeszkód



Rysunek 2: Przykład funkcji wpływu na wyliczaną w zalezności od odległości od przeszkody (względem bieżącej orientacji)

3.3 Manager sceny

Manager Sceny stanowi trzon aplikacji, odpowiada za położenia obiektów, przekazuje zwierzakom tyle informacji, na ile wskazuje ich ustalona percepcja w danych warunkach. Łączy cechy obiektów (np. zbudowane przez Edytor z GUI) z ożywiającymi je algorytmami (Strategiami).
Zadanie managera sceny:

- Utrzymanie osi czasu
- Uzupełnianie historii zdarzeń
- Uzupełnianie historii położeń zwierząt
- Wykonywanie algorytmu Managera Sceny

Ponieważ "życie"w wirtualnym świecie zwierząt miało być możliwie podobne do rzeczywistego, to oprócz wymaganej kontroli czasu i przechowywania informacji o obiektach, powinna istnieć sekwencja narzucająca kolejność wykonywania operacji, dlatego też warto wyróżnić algorytm Managera Sceny:

1. Ustalenie parametrów zwierzaków (uwzględnienie tolerancji)
2. Dla każdego zwierzaka wylicz i przekaż do Strategii: widoczne pozycje drapieżników, ofiar, przeszkód, bieżącą zwrotność, bieżącą prędkość, kierunek
3. Odbierz od Strategii nowy kierunek, nową prędkość
4. Uwzględnij ograniczenia
5. Zmień energię
6. Sprawdź czy doszło do chwycenia ofiary

3.3.1 Zdarzenia

Za sprawdzenie wystąpienie zdarzenia, oraz na reakcję na zdarzenie reaguje Manager Sceny, m.in. odnotowując je w historii zdarzeń. Przewidziano następujące zdarzenia:

- HUNTED - drapieżca dopadł ofiarę
- FRIEND_CRASH - zderzenie ze zwierzęciem tego samego rodzaju
- CRASH - zderzenie z przeszkodą

- KILLED - zwierzę zginęło

Wystąpienie zdarzenia HUNTED jest ściśle wykorzystane przez tryb gry, do zliczania zwierząt upolowanych, dodatkowo zdarzenie to determinuje również m.in.:

- wyliczenie prawdopodobieństwa zabicia ofiary (tzn. czy ofiara faktycznie zostanie upolowana)
- wyliczenie spadków energii ofiary i drapieżnika
- wyliczenie spadków prędkości i przyspieszeń ofiary i drapieżnika

3.3.2 Podejmowanie decyzji

Podejmowanie decyzji zależy, w przypadku ofiar, od odległości od drapieżnika. Jeśli ofiara 'poczuje' zagrożenie, kontrolę przejmue ustalony algorytm ucieczki. W przypadku braku zagrożenia, decyzje o położeniu są wyznaczane ze średniej ważonej algorytmów *Wander, Cohesion, Separation*. Wagi stanowią parametry zwierzęcia.

W przypadku drapieżników, decyzje są podejmowane przez liderów grup. Wybierana jest ofiara, która ma największy współczynnik *szansy* na pojmanie.

$$Szansa = \frac{|roznica\ orientacji|}{odleglosc}$$

3.3.3 Przywództwo

Wśród ofiar istnieje jeden przywódca stada, który może nakazać wszystkim zwierzakom ucieczkę w przypadku zagrożenia.

Drapieżniki mogą mieć co najmniej jednego lidera. Pzostali drapieżnicy są tzw. pomocnikami, którzy muszą być przypisani do jednego z liderów. Lider wskazuje ofiarę i wydaje komende do ataku. Atak następuje po przekroczeniu wartości progowej wskaźnika *Szansa* (por. 3.3.2).

Możliwe rozkazy:

- ORDER_NOTHING, brak rozkazów
- ORDER_RUN, ucieczka albo atak na zwierzaka w skazanego w *_OrderParam*
- ORDER_FORMGROUP, grupowanie wokól lidera
- ORDER_DISP, rozproszenie (w praktyce 'ucieczka' od lidera)

3.3.4 Energia, siła, przyspieszenie

Przyspieszenie bieżące jest wyliczane wg zależności:

$$a = \frac{V_{max}}{\Delta t}$$

co oznacza, że bieżąca siła zwierzęcia wynosi:

$$F = m * a = m * \frac{V_{max}}{\Delta t}$$

natomiast straty energii podczas ucieczki/pościgu:

$$\Delta E = \frac{m * V^2}{2}$$

straty energii podczas ataku:

$$\Delta E_{drap} = 0.9 * E_{drap}$$

$$\Delta E_{ofiara} = \frac{m_{drap}}{m_{ofiara}} * \Delta E_{drap}$$

3.4 Parametry sceny

Lista parametrów sceny:

- _stepTime - elementarny czas kroku symulacji [ms]
- _minReactionMultiplicity - najmniejsza krotność czasu kroku symulacji oznaczająca minimalny czas reakcji
- _minReactionTime - minimalny czas reakcji [ms] wynikający z krotności
- _e_phi - dokładność kątowa [rad]
- _e_len - dokładność odległości [m]
- _warningDistance - odleglość od przeszkody wyzwalająca funkcję wpływu przeszkody na kierunek ruchu zwierzaka

- _radiusCohesion - promień działania algorytmu *Cohesion* [m]
- _levelCohesion - mnożnik algorytmu *Cohesion*
- _radiusSeparation - promień działania algorytmu *Separation* [m]
- _levelSeparation - mnożnik algorytmu *Separation*
- _radiusBasicFlee - promień działania algorytmu *BasicFlee* [m]
- _levelBasicFlee - mnożnik algorytmu *BasicFlee*
- _radiusAvoidObstacles - najm. promień odległości od przeszkody
- _levelAvoidObstacles - mnożnik omijania przeszkód

3.5 GUI

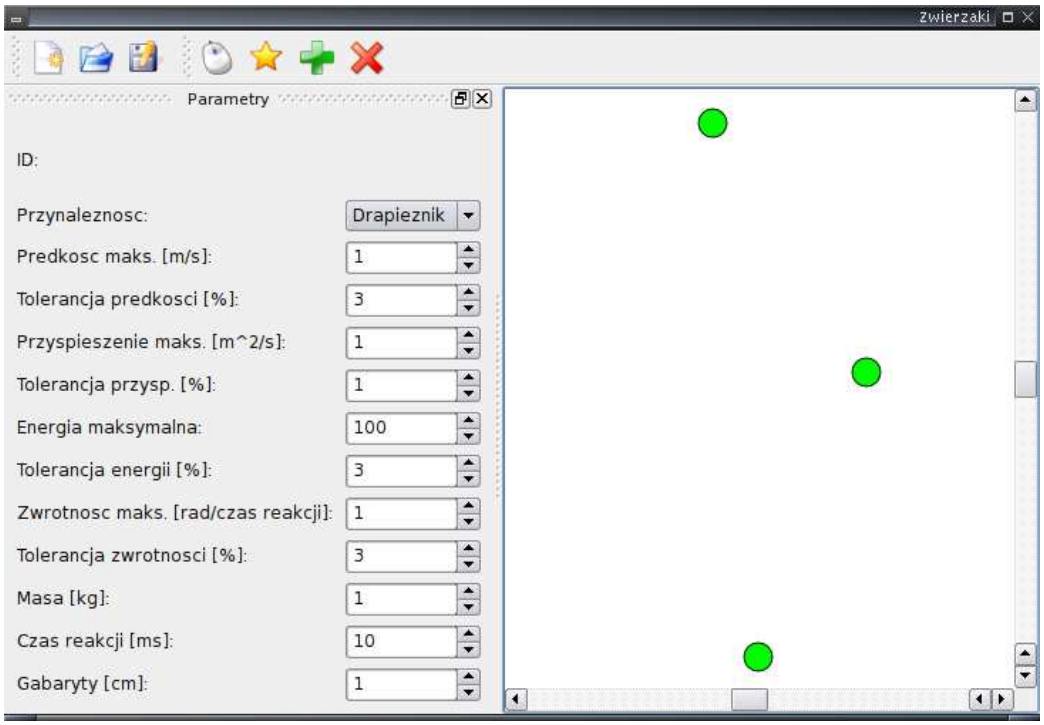
Interfejs użytkownika nie został ukończony zgodnie z planem. Nie ukończony edytor sceny, stanowi punkt krytyczny stagnacji prac w zakresie GUI. Nie mniej poniżej są przedstawione zrzuty ekranu tych fragmentów systemu, które zostały zwizualizowane.

Rysunek 3: Menu startowe programu

W przykładowej scenie (rys. 5) działają następujące funkcje:

- Tworzenie nowej sceny

Rysunek 4: Okienko otwarcia nowej sceny



Rysunek 5: Przykład sceny

- Zapis sceny do pliku (parametry zwierzaków)
- Wczytanie sceny z pliku (parametry zwierzaków)
- Dodanie nowego zwierzaka
- Ustalenie parametrów zwierzaka
- Przesuwanie zwierzaka po scenie (przy użyciu klawisza *Ctrl*)

4 Podsumowanie

Projekt mimo włożonego sporego wysiłku nie został w pełni ukończony. Istota projektu, tj. tryb gry z GUI nie został poprawnie zaimplementowany, co tymsamym tworzy projekt bezużytecznym. Korzyści jakie płyną z tego projektu, to rozbudowany silnik stworzony w oparciu o doświadczenia z poprzednimi projektami jak również nowymi przemyśleniami i znalezionymi artykułami.

Przyczyna niepowodzenia, to przede wszystkim zbyt wysoko postawiona poprzeczka w formaliźmie zachowań zwierząt, należało implementować rozwiązania sprawdzone i ew. dokonywać modyfikacji.

Literatura

[1] P. Sperzyński, P. Pańkow, A. Koreń, K. Kowalski, *Raport: Symulacja zachowań grup robotów*, ARE4350P materiały niepublikowane, PWr, 2007

[2] Strona projektu Zwierzaki w systemie SourceForge.net:
https://sourceforge.net/projects/zwierzaki

[3] Źródła projektów bazowych:
http://zwierzaki.cvs.sourceforge.net/zwierzaki/inwentaryzacja

[4] Najswieższe źródła projektu:
http://zwierzaki.cvs.sourceforge.net/zwierzaki/devel/CVSROOT/Zwierzaki

[5] Steering Behaviors Project