Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej
Informatyka 2018/2019, rok II

Symulacja dyskretna systemów złożonych

Temat projektu: Lisy i zające - zależność między ich liczebnościami

Skład osobowy grupy:

Anna Nagi

Oskar Pawica

Konrad Polarczyk

Spis treści

1.1 Opis problemu

W wielu ekosystemach na Ziemi naturalnie występuje model drapieżnik-ofiara. Może być opisany za pomocą równania Lotki-Volterry - stanowi wtedy układ dynamiczny.[1] W naszym przypadku może być wykorzystywany do symulacji zmian w populacji lisów-drapieżników i zajęcow-ofiar.

Według teoretycznych praw ekologii wiele zależy od gatunkowych cech zarówno drapieżnika, jak i ofiary oraz od środowiska, w którym żyją. [2]

Stosunki między populacjami możemy podzielić na antagonistyczne i nieantagonistyczne.
Drapieżnictwo jest rodzajem antagonistycznym, ponieważ za jego sprawą jedna strona zyskuje (drapieżnik), a druga traci (ofiara).

Drapieżnik musi dogonić, schwytać i zabić swoją ofiarę. Jest do tego celu przystosowany między innymi dzięki swojej budowie, która zazwyczaj ułatwia mu upolowanie zdobyczy.
Może się wydawać, że ofiara jest bezbronna wobec ataku drapieżnika, nie zawsze jest to jednak prawdą. Niektóre ofiary mogą być wręcz niebezpieczne dla swoich potencjalnych wrogów, ze względu na budowę swojego organizmu, czy behawioralne przystosowania.[3] Mogą się więc przyczynić do spadku wielkości populacji drapieżnika.W przypadku zająca przydatne mogą okazać się na przykład jego skoczne kończyny.

---

[1] The spreading speed for a predator-prey model with one predator and two preys
Author(s): Chin-Chin Wu

[2] Study of a Leslie–Gower predator-prey model with prey defense and
mutual interference of predators
Author(s): P. Mishra, S.N. Raw, B. Tiwari

[3] Predator–Prey Dynamics. The Role of Olfaction
Author(s): Michael R. Conover

1.2 Potencjalne możliwości rozwiązania - analiza wstępna

Symulacja oparta na systemie wieloagentowym, z zaimplementowanymi zachowaniami odpowiednimi dla lisów i zajęcy wraz z reprezentacją graficzną oraz panelem umożliwiającym ustawienie takich zmiennych jak:

- Aktualna liczba lisów i zajęcy
- Warunki środowiskowe:
    - Ilość pożywienia dla zajęcy (może być zależna od pory roku czy położenia geograficznego)
- Wskaźniki przyrostu naturalnego lisów i zajęcy

Każdy agent będzie posiadał kilka właściwości, które będą determinowały jego zachowania i stan fizyczny.[4]

Pożądane efekty symulacji powinny dać się opisać przez równanie Lotki-Volterry.[5]

[4] An individual-based evolving predator-prey ecosystem simulation using a fuzzy cognitive map as the behavior model.
Author(s): Robin Gras, Didier Devaurs, Adrianna Wozniak, Adam Aspinall.

[5] Modelling and Analysis of a Predator-Prey System with Pulses and Time Delay
Author(s): Lingshu Wang, Guanghui Feng

2. Przegląd literatury - “state-of-the-art”

Zarówno lisy, jak i zające posiadają szereg przystosowań ułatwiających im przetrwanie.

Przystosowania lisa do polowania:
- dobrze rozwinięte: słuch, wzrok, węch, dotyk
- szybkie i zwinne ruchy, skoczność
- umiejętność pływania
- umiejętność polowania w nocy
- ostre i silne kły oraz pazury - zabija poprzez ukąszenie w kark

Przystosowania zająca do ochrony przed drapieżnikami:
- tworzenie nor w głębi ziemi w celu zabezpieczenia siebie oraz potomstwa
- maskujący kolor futra
- dobrze rozwinięte narządy zmysłów, m.in. słuch - duże uszy
- skoczne tylne kończyny ułatwiające ucieczkę

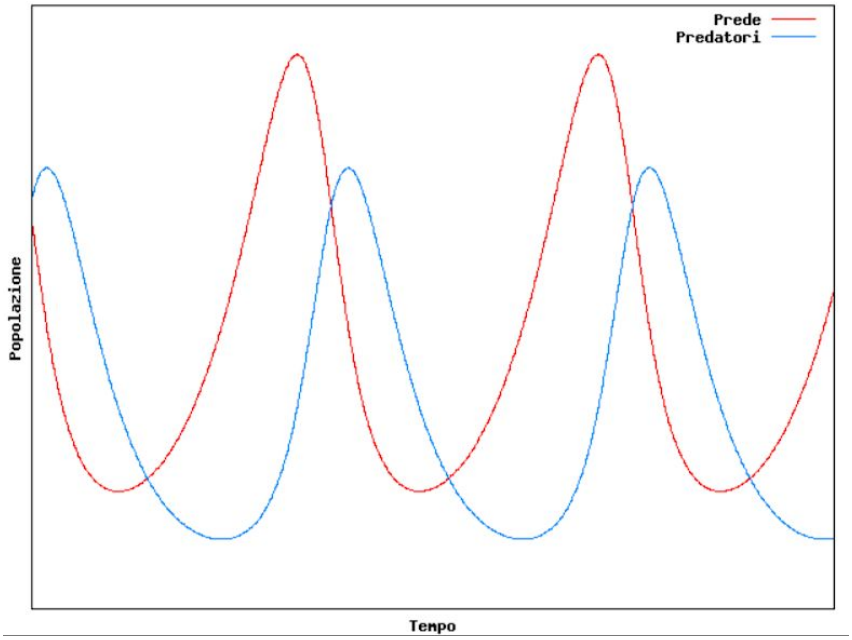
Strategie obronne zajęcy:
- kamuflaż
- życie w stadach
- szybkie przemieszczanie się

Zazwyczaj łupem lisów padają zające chore, bardzo młode lub w inny sposób osłabione, ponieważ są najłatwiejsze do schwytania. Wbrew pozorom jest to pozytywne dla populacji zajęcy, ponieważ mogą przetrwać jedynie najsilniejsze osobniki i mają one większy dostęp do pożywienia oraz innych zasobów potrzebnych do rozmnażania, np.terytorium do kopania nor dla potomstwa. [6] Czasami jednak populacja drapieżników rozrasta się tak bardzo, że ofiar zaczyna brakować. Ich mała ilość powoduje, że słabsze i gorzej przystosowane drapieżniki głodują i w efekcie umierają, co powoduje zmniejszenie ich populacji. Dzięki temu, że wywierają mniejszą presję na populację ofiar, te mogą się w spokoju rozmnażać i ich populacja na nowo się powiększa.[7] Powoduje to po pewnym czasie wzrost ilości drapieżników i taka sytuacja powtarza się

[6] Can Foxes Regulate Rabbit Populations?
Author(s): Peter B. Banks

[7] Behavioural, Morphological and Dietary Response of Rabbits to Predation Risk from Foxes
Author(s): Peter B. Banks, Ian D. Hume and Olivia Crowe

cyklicznie. Oprócz tych oddziaływań, na wielkości populacji ma wpływ także środowisko, w którym żyją. [8]



Wykres ze strony:
https://pl.wikipedia.org/wiki/R%C3%B3wnanie_Lotki-Volterry

Zależności pomiędzy populacjami zajęcy i lisów można opisać za pomocą równania różniczkowego Lotki-Volterry.[9]

Podstawowy opis modelu Lotki-Volterry

Z - populacja ofiar
L - populacja drapieżników

[8] Invasive prey controlling invasive predators? European rabbit abundance does not determine red fox population dynamics
Author(s): Michael P. Scroggie, David M. Forsyth, Steven R. McPhee, John Matthews, Ivor G. Stuart, Kasey A.Stamation, Michael Lindeman, David S.L. Ramsey
[9] Modelling prey in discrete time predator-prey systems
Rory Mullan, David H. Glass and Mark McCartney

Rozważamy środowisko, w którym występują wyłącznie 2 gatunki zwierząt - w naszym przypadku zając i lis. Wprowadzenie dodatkowych gatunków[10] prowadzi do zwiększenia stopnia skomplikowania modelu.[11]

Sytuacje skrajne:

a) brak lisów - sytuacja korzystna dla zajęcy, mają swobodę życia i rozmnażania się, ich liczba zwiększa się według równania Malthusa $Z'(t) = rO * Z$
b) brak zajęcy - sytuacja niekorzystna dla lisów, głodują, ich populacja się zmniejsza według równania $Z'(t) = rOZ$

Uwagi odnośnie modelu:

- polowanie lisów na zające powoduje spadek liczby zajęcy, bo są zabijane
- wzrost liczby zajęcy powoduje wzrost liczby lisów, bo mają więcej pożywienia
- spadek liczby zajęcy powoduje spadek liczby lisów, bo lisy zaczynają głodować
- spadek liczby lisów powoduje wzrost liczby zajęcy, bo mniej drapieżników na nie poluje[12]
- zakładamy równomierne rozmieszczenie zwierząt na terenie

Klasyczny model Lotki-Volterry[13] uzyskany po zastosowaniu tych uwag:

$$Z'(t) = (rO - uOL)Z$$
$$L'(t) = (rDZ - uD)L$$

gdzie:

Z - wielkość populacji zajęcy

L - wielkość populacji lisów

[10] Periodic Solutions of Discrete Time and Ratio-Dependent System with One Predator and Two Cooperative Preys
Author(s): Li Yanhong, Zhao Ming, Liu Taihui, Cheng Rongfu

[11] Dynamics of predator–prey system with fading memory
Author(s): Banshidhar Sahoo, Swarup Poria

[12] Predator-prey dynamics in an ecosystem context
Author(s): James F.Kitchell, Lisa A.Eby, Xi He, Daniel E.Schindler, Russell A.Wright

[13] Analysis of a Predator-Prey System Concerning Impulsive Effect
Author(s): Lingshu Wang, Guanghui Feng

rO - współczynnik rozrodczości ofiar

uO - współczynnik sukcesu polowań drapieżników (umierania ofiar w wyniku drapieżnictwa)

rD - współczynnik rozrodczości drapieżników

uD - współczynnik śmiertelności drapieżników

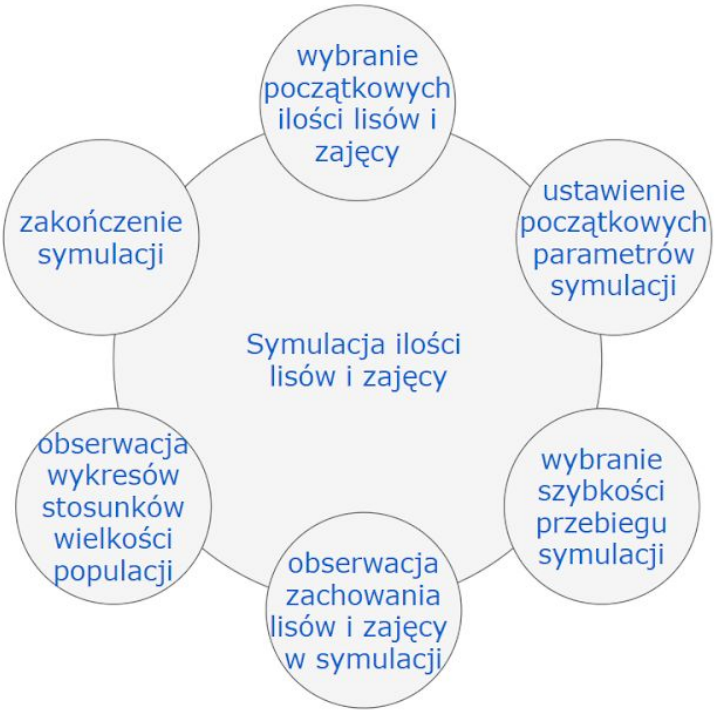
3.1 Cele modelu

Celem modelu jest stworzenie symulacji, która jak najdokładniej przedstawia rzeczywiste procesy zachodzące w przyrodzie - zależności pomiędzy populacjami drapieżników i ofiar na przykładzie lisów i zajęcy. Rozwój, rozmnażanie i umieranie zwierząt zależą od wielu czynników, co postaraliśmy się zawrzeć w algorytmie, aby był możliwie najbardziej zbliżony do świata realnego.

3.2 Założenia algorytmu

Zarówno lisy, jak i zające mogą się rozmnażać. Lisy umierają z głodu lub starości, króliki tylko ze starości. Rozmnażają się oba gatunki - osobniki, które ukończyły 2 rok życia. Mogą rozmnażać się tylko względnie najedzone osobniki w przypadku lisów (poziom najedzenia niezbędny do rozmnażania użytkownik może regulować suwakiem). Po skutecznej lub nieskutecznej próbie rozmnażania, osobniki muszą poczekać, aby rozmnożyć się ponownie - nie zawsze się to udaje. Lisy są szybsze, ale zające bardziej zwrotne podczas polowania/ucieczki.

3.3 Diagram przypadków użycia



4. Algorytm i ważne części programu

4. 1. Język programowania

4.1.1 Java

Językiem programowania, który został przez nas wybrany do wykonania projektu jest Java. Wybraliśmy ten język, ze względu na prostotę pisania kodu, kompilowania oraz debugowania. Dodatkowym atutem jest możliwość programowania obiektowego. Dzięki automatycznej alokacji pamięci i garbage collector Java jest znacznie łatwiejsza w użytkowaniu niż przykładowo C++. Dodatkowo, kod jest uniwersalny i łatwo przenaszalny pomiędzy różnymi urządzeniami. Funkcjonalność Javy można rozszerzyć za pomocą gotowych i własnych bibliotek, ułatwiających pracę z tym językiem programowania.

4.2. Biblioteki wykorzystane w projekcie

4.2.1 JADE

JADE, czyli Java Agent DEvelopment Framework jest open-source'ową platformą dla aplikacji wieloagentowych w pełni zaimplementowaną w języku Java. Dzięki temu kod może być przenaszalny pomiędzy różnymi platformami. JADE posiada wbudowane narzędzia graficzne ułatwiające debugowanie oraz rozwój projektu. Dodatkowo pozwala na wykonywanie tasków i komunikację pomiędzy agentami.

4.2.2 Vecmath

Vecmath to paczka służąca do obsługi wektorów 3D w Javie. W projekcie służy do graficznej reprezentacji agentów - zajęcy i lisów w przestrzeni oraz ich otoczenia. Zawiera m.in. wektory, punkty przestrzenne o wielu koordynatach, osie liczbowe.

4.2.3 Swing

Biblioteka graficzna - nowa wersja AWT.

4.2.4 AWT

AWT (inaczej Abstract Window Toolkit) - standardowa biblioteka graficzna języka Java. Ponadto, umożliwia przechwytywanie zdarzeń wprowadzonych przez użytkownika.

4.3. Implementacja - szczegółowy opis

Projekt składa się z następujących klas:

Pakiet **engine**:

AnimationAgent - Dziedziczy po klasie *Agent* z biblioteki JADE. Implementuje interfejs *GraphicComponent*. Odpowiada za tworzenie agentów, dodanie zachowania (behaviour), uruchomienie zachowań u zwierząt oraz dodanie komponentów graficznych do symulacji.

AnimationThread - Dziedziczy po klasie *Thread*. Odpowiada za uruchomienie i nowe instancje wątków koniecznych do utworzenia agentów. Jest także odpowiedzialna za aktualizację symulacji.

ControlPanel - Dziedziczy po klasie *UIPanel*. Pozwala na utworzenie nowego panelu sterowania dla symulacji. Zawiera przyciski: start, pauza, stop, a także suwak prędkości zachodzenia procesu symulacji.

Debug - Klasa odpowiedzialna za rysowanie linii i siatki. Zawiera klasę *Line* oraz *Grid*, która implementują interfejs *GraphicComponent*.

GraphicComponent (interfejs) - Pochodzi z biblioteki AWT. Jego metoda paintComponent jest rozszerzana przez inne klasy.

GUI - Dziedziczy po klasie *JFrame*. Odpowiada za graficzny interfejs użytkownika - jego wyświetlanie oraz budowanie.

MonoBehaviour - Dziedziczy po klasie *Behaviour*, pochodzącej z biblioteki JADE. Dzięki niej do symulacji można dodawać, modyfikować i usuwać zachowania agentów.

PlotPanel - Dziedziczy po klasie *JPanel*. Odpowiada za wykres liczebności lisów i zajęcy - jego wygląd, wyświetlanie poszczególnych wielkości oraz aktualizację wraz z procesem symulacji.

SettingsPanel - Dziedziczy po klasie *UIPanel*. Zawiera klasę *Slider*, która umożliwia tworzenie suwaków, służących do manipulacji przebiegiem symulacji. Przykładowe parametry, które użytkownik może samodzielnie zmieniać to: szybkość rozmnażania lisów i zajęcy, proporcje płciowe zwierząt w populacji, częstość rozmnażania.

SimulationManager - Dziedziczy po klasie *Agent*. Jest odpowiedzialna za zbudowanie animacji, inicjalizację symulacji, tworzenie zwierząt - agentów.

SimulationPanel - Dziedziczy po klasie *JPanel*. Pozwala na dodawanie i usuwanie graficznych komponentów aplikacji

Time - Odpowiada za czas symulacji oraz jej ciągłą aktualizację.

Timer - Reguluje i uaktualnia czas w trakcie działania aplikacji.

UIPanel - Dziedziczy po klasie *JPanel*. Jest odpowiedzialna za wyświetlanie komponentów graficznych wraz z ich opisem.

Utils - Wyszukuje w symulacji agentów o określonym typie, określa ich kolor (zielone-zające, pomarańczowe-lisy).

Viewport - Odpowiada za widok symulacji.

Pakiet **extensions**:

Vector2d - Rozszerza klasę *Vector2d* z biblioteki vecmath. Odpowiada za graficzne wektory tworzące symulację i ustawienie ich w przestrzeni.

Pakiet **main**:

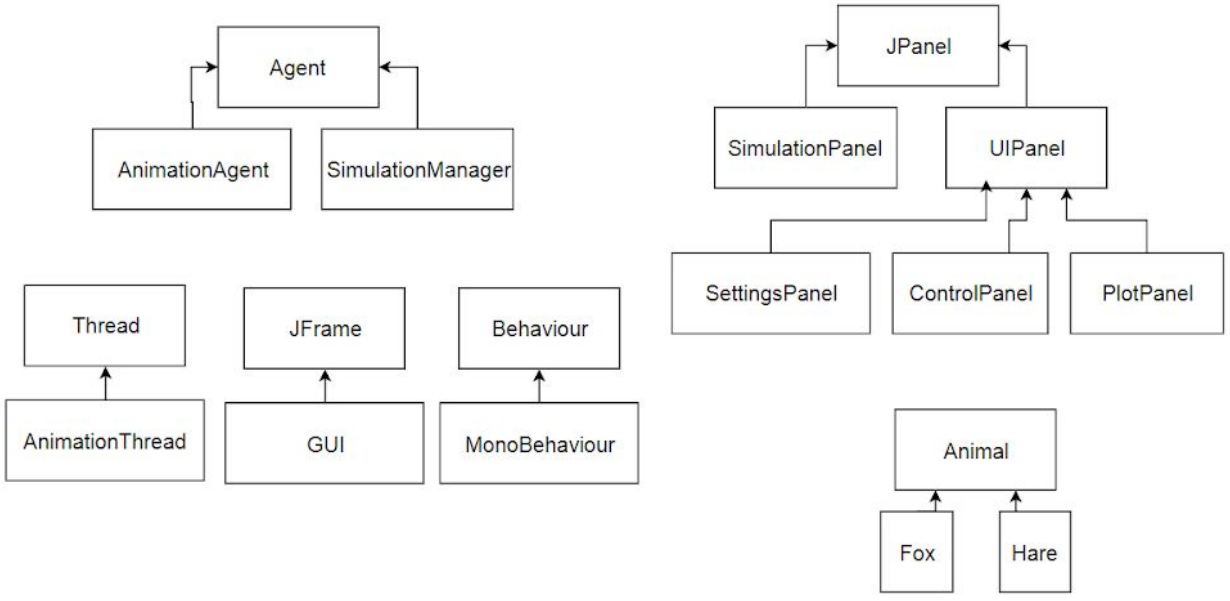
Animal - Zawiera funkcje odpowiedzialne za m.in. wiek zwierzęcia i starzenie się (większe osobniki są bardziej widoczne dla przeciwnego gatunku), rozmnażanie, płeć, umieranie, pozycję zwierzęcia w symulacji oraz poruszanie się (kierunek ruchu, zwrotność, prędkość). Wewnętrzna klasa *VisionCone* odpowiada za pole widzenia zwierzęcia (konieczne do polowania, uciekania i rozmnażania).

DataBase - Służy do zbierania i magazynowania danych na temat agentów - zwierząt i przebiegu symulacji. Zapewnia modyfikację i dostęp do danych oraz minimalne i maksymalne możliwe wartości niektórych parametrów.

Fox - Dziedziczy po klasie *Animal*. Odpowiada za tworzenie lisa, polowanie i jedzenie zająca, głód i umieranie. Wewnętrzna klasa *AnimalMovementController* umożliwia wyszukiwanie i pogoń za ofiarą.

Hare - Dziedziczy po klasie *Animal*. Odpowiada za tworzenie zająca i jego funkcje życiowe, np. umieranie. Wewnętrzna klasa *HareMovement* pozwala na ruch i uciekanie przed drapieżnikami w najbardziej optymalny sposób.

4.4 Diagramy UML wybranych klas



4.5 Opis funkcji symulacji

Wpływ użytkownika na symulację
Niektóre elementy symulacji mogą być modyfikowane przez użytkownika. Mają one wpływ na zmianę przebiegu symulacji i zachowania lisów i zajęcy, a także wzajemny stosunek ilościowy obu populacji.

Polowanie
Lisy i zające w normalnym trybie symulacji poruszają się w sposób losowy. Gdy zając znajdzie się w polu widzenia lisa(oznaczonym jako błękitny stożek), rozpoczyna się polowanie. Lisa zaczyna gonić zająca - porusza się szybciej w jego kierunku. Drapieżnik jest szybszy, ale mniej zwrotny od swojej ofiary (w symulacji promień skrętu lisa jest większy niż zająca). Ponadto, zając ucieka według pewnego algorytmu - w stronę, która jest najdalej od wszystkich lisów w zasięgu jego pola widzenia.

Rozmnażanie
Jeśli 2 osobniki tego samego gatunku znajdą się w swoim polu widzenia, istnieje szansa na rozmnażanie. Następuje wtedy sprawdzenie płci (niezbędny jest 1 samiec i 1 samica) oraz wieku (mogą rozmnażać się jedynie osobniki powyżej 2 roku życia). Jeśli warunki będą sprzyjające może dojść do rozmnażania, którego skuteczność jest modyfikowalna przez użytkownika.

Głód/sytość
Wystąpienie głodu może prowadzić do śmierci lisów. Podczas głodowania zwierzęta nie mogą się rozmnażać, skupiają się wtedy na polowaniu.Gdy zwierzęta są całkiem najedzone, nie polują i częściej się rozmnażają. Szybkość zmiany współczynnika sytości/głodu również może być modyfikowana.

Suwaki

Simulation Speed - szybkość przebiegu symulacji

Hare/Fox Initial Population - ilość początkowa lisów/zajęcy na mapie. Możliwość edytowania jest zablokowana po rozpoczęciu symulacji.

Hare/Fox Birth Rate - współczynnik narodzin zwierząt. Wartości między 1 a 100. Jest to procentowa szansa na to, że przy spotkaniu samca i samicy danego gatunku pojawi się nowy zwierzak (dziecko).

Maximum Hunger - u drapieżników wprowadziliśmy wskaźnik głodu, który z czasem spada. Ten suwak oznacza jego maksymalną wartość liczbową, czyli "pełne najedzenie". Jeśli głód spadnie do zera, lis umiera. Dlatego im większy maximum hunger, tym dłużej lis wytrzyma bez jedzenia.

Hunger Per Meal - określa ile "punktów najedzenia" odnawia się lisowi po zjedzeniu zająca. Im większa wartość tego parametru, tym lepiej lis zaspokaja głód poprzez zjedzenie jednej ofiary.

Hunger Loss (per sec) - określa ilość traconych punktów najedzenia na sekundę. Wartości między 0.1 a 1.0

min BreedHungerPercentage - wskaźnik wpływu głodu na rozrodczość lisów. Im mniejsza wartość, tym częściej lisy będą się rozmnażać mimo bycia głodnymi.

Max HuntHunger Percentage - stosunek najedzenia lisa do tego, czy mu się będzie chciało polować, pomimo tego że jest najedzony. Im większy ten współczynnik, tym częściej będzie polował, bo częściej będzie chciał zaspokajać głód. Im mniejszy, tym większa szansa, że przejdzie obojętnie, mimo że niedaleko niego jest zajac.

Gender Max Percentage - górna granica maksymalnego procentu samców w populacji. Przy wartości 0.5 liczba samców nie może przekroczyć 50% ogólnej populacji. Przy wartości 1.0 będą mogły być same samce(100%), jeśli akurat tak się wylosuje, bo algorytm nie będzie ograniczał tej wielkości. Wartości między 0.01 a 1.0

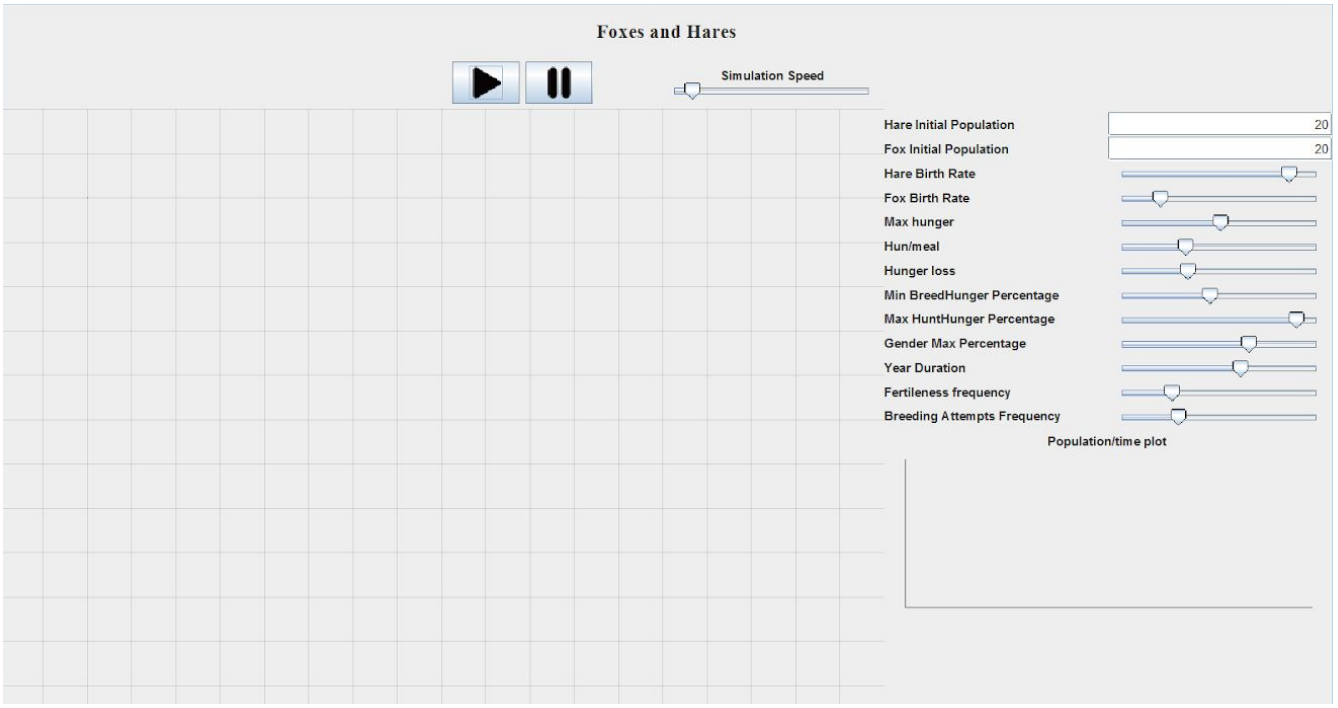
Year Duration - wyrażona w sekundach wartość określająca ile sekund symulacji odpowiada jednemu rokowi. Mamy ustawioną (wziętą z wikipedii uśrednioną) długość życia zwierząt na 13 lat. Czyli im większy będzie ustawiony ten parametr, tym dłużej zwierzęta pożyją zanim umrą ze starości. Wartości między 30 a 80.

Fertileness Frequency - określa po ilu sekundach od narodzin ostatniego małego zwierzaka samica staje się znowu płodna. Im mniejsza wartość, tym częściej zwierzęta będą się rozmnażać.

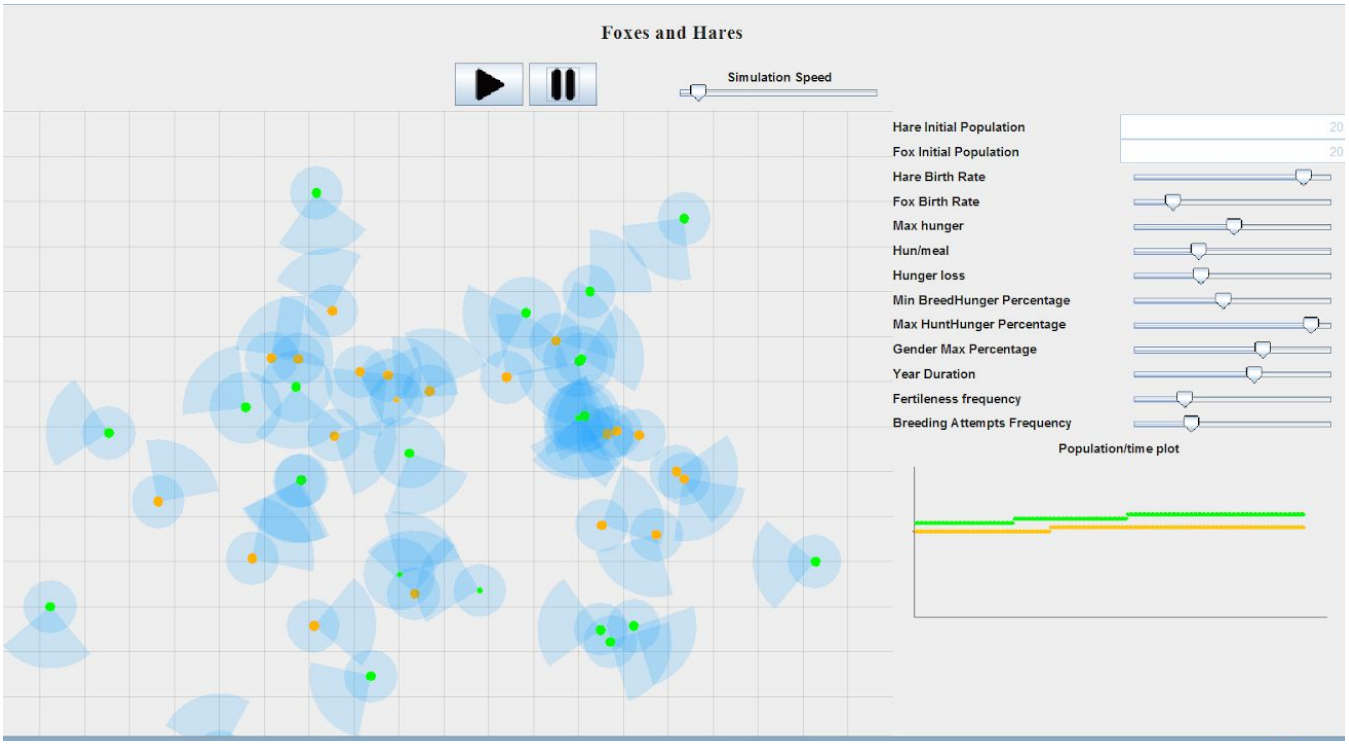
Breeding Attempts Frequency - określa po ilu sekundach po nieudanej próbie rozmnożenia się między samcem a samicą mogą podjąć kolejną próbę. Wartości między 1.0 a 8.0

5. Aplikacja i wyniki symulacji - przykładowe uruchomienie aplikacji

Tuż po uruchomieniu:

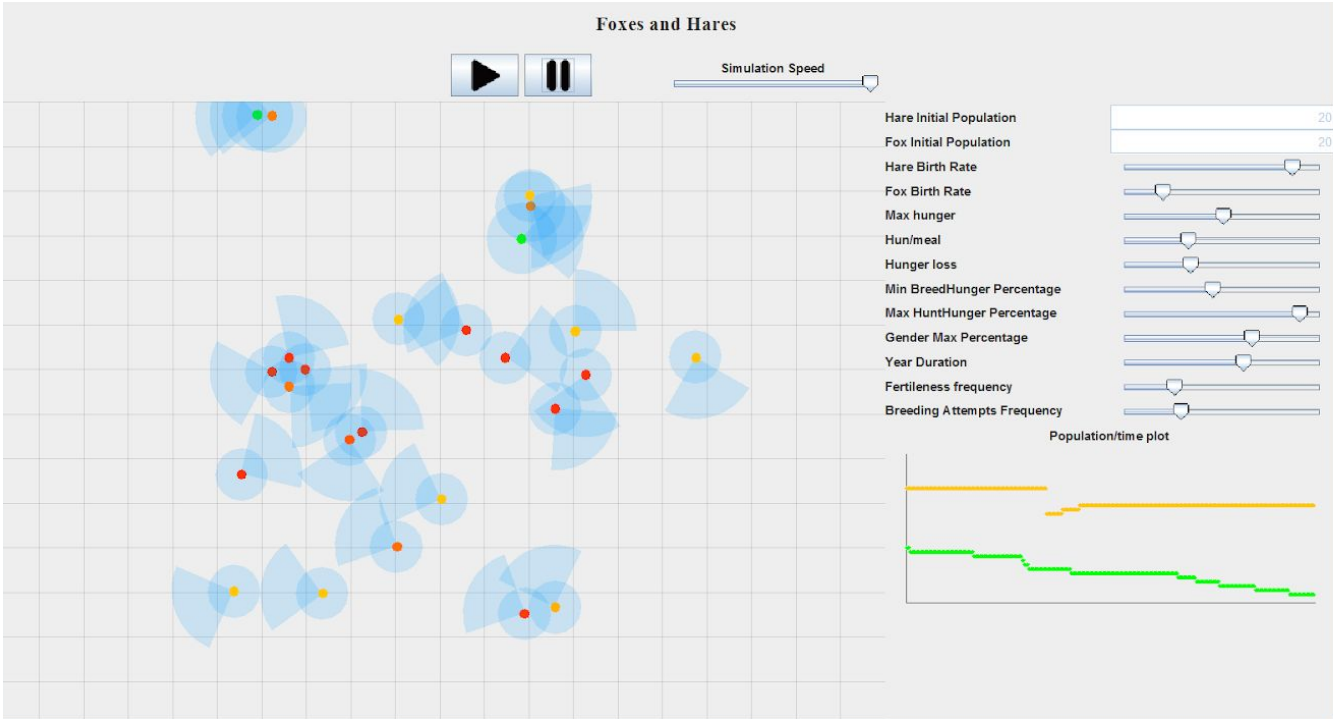


Pomarańczowe kropki i wykres - lisy, zielone kropki i wykres - zające
Po zainicjalizowaniu procesu symulacji:

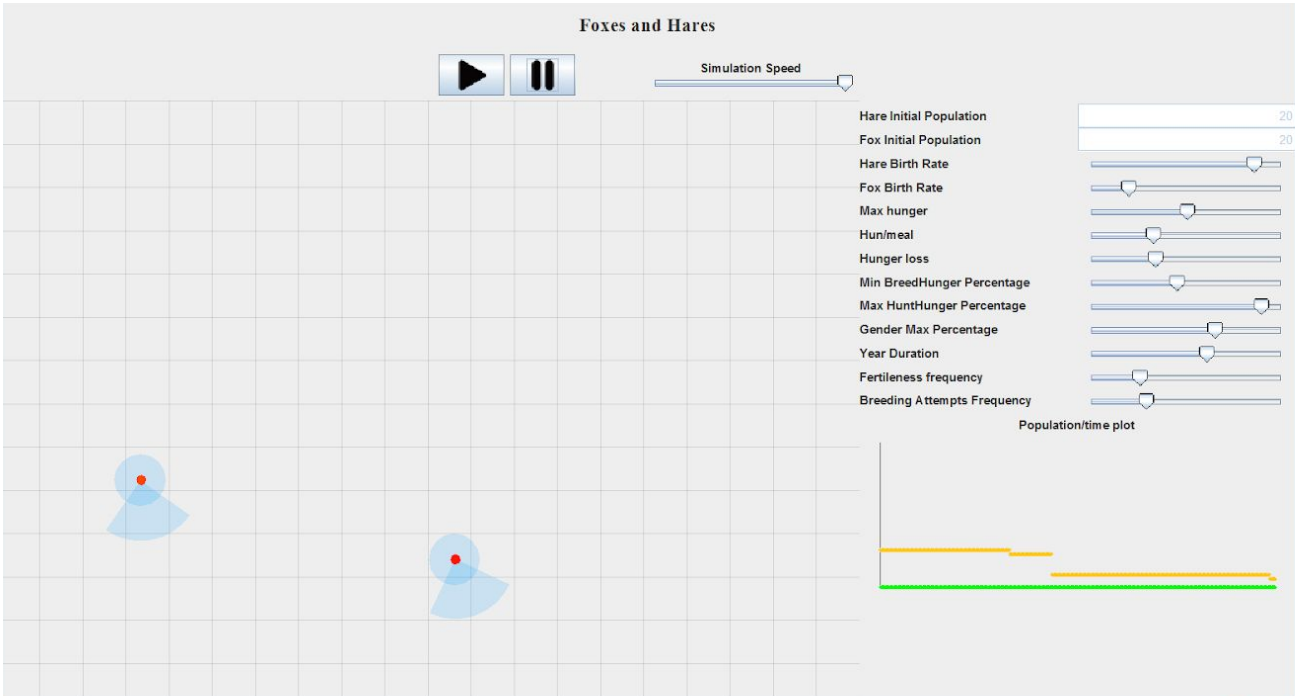


Jak widać obie populacje utrzymują się na podobnym poziomie liczebności.

Po pewnym czasie liczba lisów znacznie wzrasta,a liczba zajęcy maleje:



Ze względu na brak pożywienia, liczba lisów również zaczyna spadać, a później maleje do zera:



6.Wnioski

6.1 Wnioski ogólne:

Aplikacja jest w stanie przybliżyć w dość dokładny sposób zależności pomiędzy populacjami drapieżników i ofiar. Przy odpowiednio ustawionych parametrach, zachodzą zjawiska opisane równaniem Lotki-Volterry. Udało się w sprawny sposób pokazać zależności pomiędzy liczebnościami zwierząt - znacznie ułatwił to wyświetlany obok symulacji wykres. Można dążyć do lepszej dokładności obliczeń poprzez modyfikację parametrów za pomocą suwaków.

6.2 Future Works

W przyszłości można rozbudować aplikację o nowe gatunki zwierząt, zwiększając w ten sposób bioróżnorodność i dokładność symulacji. Można także wprowadzić elementy świata nieożywionego, jak np. kamienie, kryjówki,

rośliny, jeziora, z którymi zwierzęta mogłyby wchodzić w interakcję jak w prawdziwym życiu.