# Projekt PO

# Etap V

# „Symulacja interakcji między zwierzętami”

Grupa 6:

Bartosz Hornicki

Tomasz Grochowski

# Opis programu w języku naturalnym

Na polanie żyją dwa gatunki zwierząt: reprezentanci grupy roślinożerców – zające oraz reprezentanci grupy mięsożerców – wilki.

Każde zwierzę skupia się na zaspokajaniu swoich podstawowych potrzeb, by móc przeżyć jak najdłużej. W przypadku zajęcy jest to zależne wyłącznie od ukształtowania terenu, gdyż pragnienie zaspokajają pijąc wodę z pobliskich źródeł, a pożywieniem dla nich jest trawa rosnąca pod ich nogami. Dostęp do trawy nie jest jednak nieograniczony, gdyż nie rośnie ona na piasku, który otacza zbiorniki wodne. Zdecydowanie łatwiej jest im jednak zaspokoić swój głód niż wilkom, które by móc owy głód zaspokoić muszą najpierw znaleźć niedawno zmarłego zająca. Z tego też powodu wilki gdy poczują głód stają się agresywne i atakują pobliskich przedstawicieli grupy roślinożernej. Gdy dane zwierzę nie je, bądź nie pije przez długi czas staje się ono coraz słabsze, co ostatecznie kończy się śmiercią.

W zaspokajaniu tych potrzeb jednak przeszkadza konieczność zwalczanie zmęczenia, które wraz z upływem czasu staje się coraz bardziej uciążliwe. Gdy zwierzę nie jest głodne ani spragnione może ono w każdej chwili ułożyć się do snu, gdy jednak w trakcie poszukiwań pokarmu i wody zmęczenie danego osobnika osiągnie poziom wyczerpania, mimo głodu czy pragnienia może ono opaść z sił i zasnąć, co jest bardzo niebezpieczne i może kończyć się śmiercią głodową, lub w przypadku zająca również atakiem.

Dodatkowym utrudnieniem przy odpowiednim funkcjonowaniu jest podeszły wiek osobnika. Stare osobniki poruszają się wolniej i dłużej zajmuje im wykonywanie poszczególnych czynności. Oba gatunki cechują się określoną średnią wieku, możliwego do osiągnięcia, nie każde jednak zwierzę po jego osiągnięciu umiera. Zdarzają się i przypadki w których dany osobnik może przeżyć, jednak z reguły nie jest to wiek przekraczający średnią w znacznym stopniu.

Zwierzęta mogę poruszać się swobodnie po terenie polany, poza sytuacjami, w których drogę zablokuje im woda, jakiś obiekt (np. drzewo) lub inny osobnik.

# Spis narzędzi użytych przy tworzeniu projektu

* GitHub (<https://github.com/Horochov/pwrSymulatorZwierzat>)
* Algorytm OpenSimplexNoise (<https://gist.github.com/KdotJPG/b1270127455a94ac5d19>)
* Biblioteki języka Java
* Microsoft Word
* Narzędzie do tworzenia diagramów UML: (<https://online.visual-paradigm.com>)
* Dokumentacja Javy (<https://docs.oracle.com/en/java/>)
* Stack Overflow (<https://stackoverflow.com>)

# Wymagania funkcjonalne

1. Rozpoczęcie symulacji (użytkownik)
2. Ustawienie szybkości symulacji (użytkownik)
3. Przerwanie symulacji (użytkownik)
4. Poruszanie się zwierząt po siatce
5. Decyzja zwierząt o czynności

5.1. Poszukiwanie za pożywieniem

5.2. Atakowanie innego zwierzęcia

5.3. Ucieczka przed innym zwierzęciem

5.4. Swobodne poruszanie się

1. Dorastanie zwierząt
2. Zamiana zwierząt w padlinę po śmierci
3. Pola dostępne i zabronione dla zwierząt
4. Losowe zdarzenia

# Analiza czasownikowo-rzeczownikowa

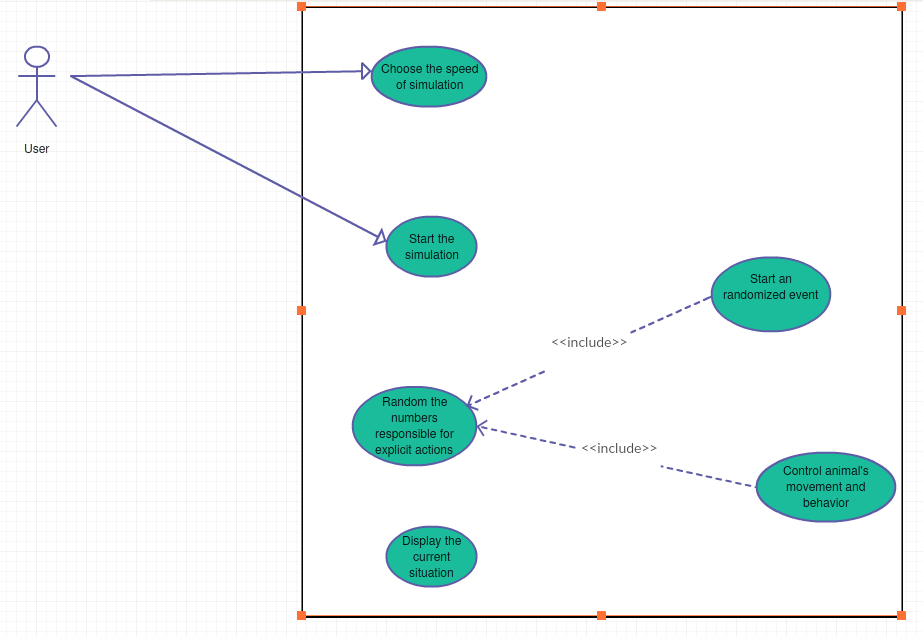
Celem projektu było stworzenie turowej symulacji zależności między roślinożercami (króliki) a drapieżnikami (wilki). Użytkownik może włączyć symulację, ustawić prędkość jej wykonywania i przerwać ją.

Zwierzęta poruszać się będą po terenie złożonym z kwadratowych pól. Zwierzę na podstawie swoich potrzeb zadecyduje jaką czynność wykonać - szukać za pożywieniem, atakować inne zwierzę bądź uciekać, swobodnie poruszać się lub odpoczywać. Każde zwierzę posiada wiek, który wpływa na jego statystyki. Zwierzę może umrzeć ze starości lub zostać zabite – zamienia się wtedy w padlinę, która jest pożywieniem.

Teren składa się z kwadratowych pól. Pola mogą być dostępne dla zwierząt, bądź nie, np. drzewo, kamień. Na polu może znajdować się surowiec - trawa, padlina lub woda.

W symulacji mogą wystąpić losowe zdarzenia mające wpływ na teren, grupę zwierząt lub pojedynczego osobnika.

# Diagram UML przypadków użycia



# Karty CRC

|  |  |
| --- | --- |
| Class name:Main | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Contains main function * Manages user input * Keeps game running | Collaborators:   * Game |

|  |  |
| --- | --- |
| Class name: Game | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Creates terrain * Manages animal placement | Collaborators:   * Animal * Tile * Textures * Position * OpenSimplexNoise |

|  |  |
| --- | --- |
| Class name: IAnimal interface | |
| Superclasses: | |
| Subclasses: Animal | |
| Responsibilities:   * Declares common animal functions | Collaborators: |

|  |  |
| --- | --- |
| Class name: AnimalStats | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Stores information about animal statistics. | Collaborators: |

|  |  |
| --- | --- |
| Class name: Animal abstract | |
| Superclasses: IAnimal | |
| Subclasses: Wolf, Hare | |
| Responsibilities:   * constructs "animal" objects * generally exhibits animals capabilities | Collaborators:   * AnimalStats |

|  |  |
| --- | --- |
| Class name: Hare | |
| Superclasses: Animal | |
| Subclasses: | |
| Responsibilities:   * defines animal's statistics * defines animal's reactions and behaviour * defines animal's needs | Collaborators: |

|  |  |
| --- | --- |
| Class name: Wolf | |
| Superclasses: Animal | |
| Subclasses: | |
| Responsibilities:   * defines animal's statistics * defines animal's reactions and behaviour * defines animal's needs | Collaborators: |

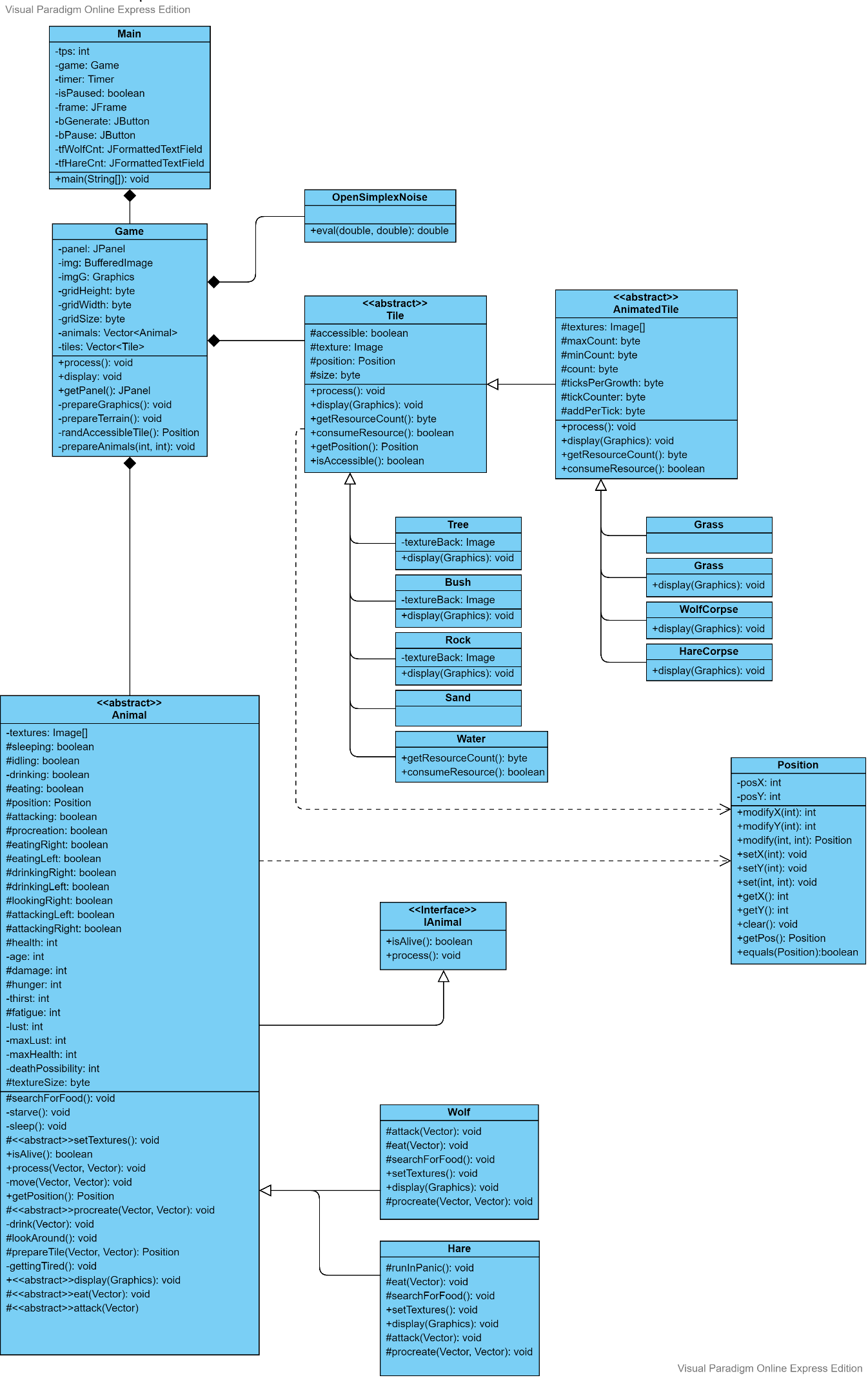
|  |  |
| --- | --- |
| Class name: Tile | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * displays the type of ground within * shows the resources specific tile contains * allows animals to move around | Collaborators:   * Position * Textures * ResourceType |

|  |  |
| --- | --- |
| Class name: OpenSimplexNoise | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Generates random noise map | Collaborators: |

|  |  |
| --- | --- |
| Class name: Position | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Encapsulates coordinates | Collaborators: |

|  |  |
| --- | --- |
| Class name: Textures | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Stores terrain tile textures. | Collaborators: |

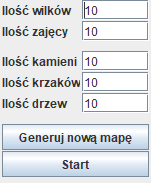
|  |  |
| --- | --- |
| Class name: ResourceType enumeration | |
| Superclasses: | |
| Subclasses: | |
| Responsibilities:   * Informs about resource type | Collaborators: |

Diagram UML klas

# Opis interfejsu użytkownika



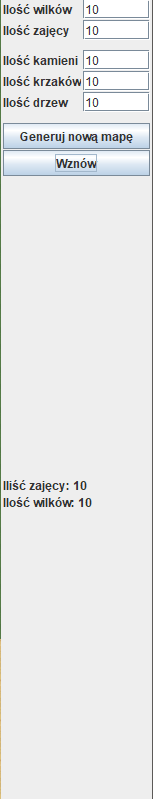
Wygląd okna. Większą część zajmuje plansza składająca się z terenów (trawy, piasku, wody), przeszkód (kamienie, krzaki, drzewa) i zwierząt (wilki i zające). Plansza generuje się z domyślnymi wartościami – po 10 przeszkód i zwierząt każdego rodzaju.



Liczbę zwierząt i przeszkód można zmienić poprzez wpisanie wartości w odpowiednie pole. W takim wypadku należy także kliknąć „Generuj nową mapę”.

Przycisk „Start” włącza symulację, zmieniając wyświetlany przez siebie tekst na „Pauza”. Po ponownym kliknięciu przycisku symulacja zostanie wstrzymana, a tekst przycisku zmieni się na „Wznów”.

Symulację można przerwać w dowolnym momencie także poprzez kliknięcie przycisku „Generuj nową mapę”. Jest to równoznaczne z przygotowaniem kolejnej symulacji do uruchomienia.



Wraz z uruchomieniem symulacji pojawią się dwa pola z liczbą żyjących zwierząt danego gatunku.

Symulacja zakończy się, gdy pozostanie tylko jedno zwierzę.



W momencie zakończenia symulacji liczba zwierząt zamienia się w powyższą wiadomość – w sytuacji gdy wygrał wilk wiadomość zostaje adekwatnie zmieniona.

# Uwagi i wnioski

Do napisania programu i ukończenia projektu niezbędne okazało się zaczerpnięcie z wiedzy i doświadczenia użytkowników strony <https://stackoverflow.com>, co pomogło nam przy rozwiązaniu wszelkich napotkanych problemów. Równie przydatnym okazało się składowanie dokumentacji w repozytorium, co w znacznym stopniu ułatwiło wykonanie projektu. Niezwykle pomocne okazało się również wykorzystanie odpowiedniego środowiska programistycznego, które oferowało liczne podpowiedzi i pomoce przy pisaniu kodu, oraz zapewniało dostęp do publicznych dokumentacji Javy, by móc jednak skutecznie znaleźć w niej co było potrzebne i móc to odpowiednio wykorzystać konieczna była znajomość podstaw tego języka programowania i znajomość języka angielskiego. Kluczowym okazało się być odpowiednie zaimplementowanie w klasach funkcji „process”, która zbierała w sobie całą niezbędną funkcjonalność danych klas.