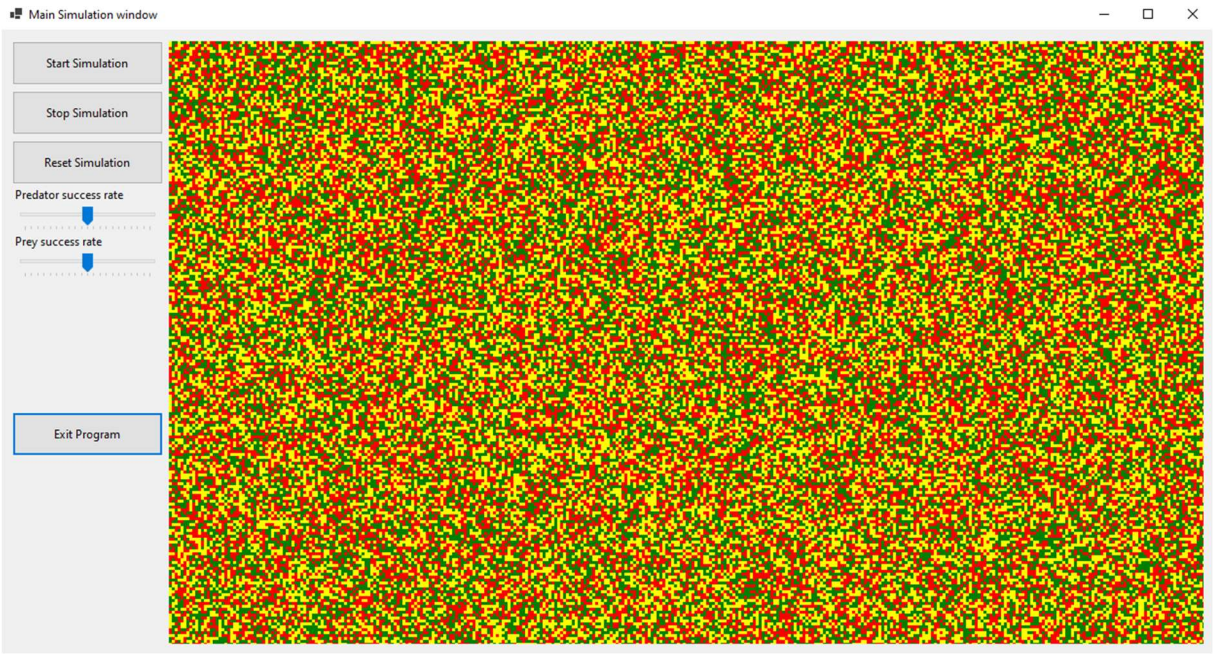
Symulacja Drapieżnik-Ofiara

Bartłomiej Konecki s203967
Informatyka i ekonometria zaocznie semestr 5
20.11.2023

Celem symulacji jest wizualne przedstawienie interakcji między drapieżnikami i ofiarami w środowisku naturalnym.

Po uruchomieniu aplikacji symulacji powita nas okno symulacji.



W oknie wygenerowane jest środowisko naturalne sawanny wypełnione losowo zwierzętami w postaci kwadratów, kolory symbolizują różne stany danej przestrzeni. Legenda środowiska, znaczenie kolorów kwadratów:

Kolor kwadratu	Znaczenie
zielony	Lokalizacja zawierająca roślinożercę, pomyślmy o zebrze albo antylopie
czerwony	Lokalizacja zawierająca drapieżnika, pomyślmy o lwie albo gepardzie
żółty	Lokalizacja pusta, pomyślmy o żółtej, wypalonej przez słońce, trawie sawanny

W oknie dostępne są przyciski:

1. ‘Start simulation’ – rozpoczęcie albo kontynuacja działania symulacji;
2. ‘Stop simulation’ – wstrzymanie działania symulacji;
3. ‘Reset simulation’ – wstrzymanie działania symulacji i ponowne wypełnienie środowiska losowymi zwierzętami;
4. Pasek ‘Predator success rate’ – reguluje skuteczność z jaką drapieżnicy polują na inne zwierzęta;

5. Pasek ‘Prey success rate’ – miał regulować skuteczność roślinożerców, ale nie został zaimplementowany;
6. ‘Exit program’ – wyłączenie aplikacji.

Symulacja działa na podstawie siatki obiektów typu `enum CellType`. Obiekty te mogą mieć stan Empty, Predator albo Prey.

```
enum CellType
{
    Empty,
    Predator,
    Prey
}
```

Gdy symulacja pracuje komputer iteruje po wszystkich polach w poszukiwaniu obiektów typu Predator i Prey. Jeżeli znajdzie zwierzę to uruchamia funkcje, które symulują zachowania dla danego typu zwierzęcia w danej lokalizacji, czyli PredatorBehavior() albo PreyBehavior(). Po wdrożeniu zmian wywołanych logiką zwierząt, aplikacja na nowo rysuje środowisko naturalne, czyli DrawGrid(). Na koniec odczekujemy jakąś porcję czasu, żeby łatwiej było obserwować zmiany w symulacji, domyślnie 0,5 sekund.

```
private async void StartSimulation()
{
    isRunning = true;
    while (isRunning)
    {
        for (int i = 0; i < gridSize; i++)
        {
            for (int j = 0; j < gridSize; j++)
            {
                if (grid[i, j] == CellType.Predator)
                {
                    PredatorBehavior(i, j);
                }
                else if (grid[i, j] == CellType.Prey)
                {
                    PreyBehavior(i, j);
                }
            }
        }

        DrawGrid();
        await Task.Delay(delayMs);
    }
}
```

O logice zwierząt.

Do zbudowania logiki zwierząt niezbędna była funkcja pomocnicza FindNerby() która zwraca listę sąsiadujących z głównym zwierzęciem kratek.

```
private List<Point> FindNearby(int x, int y, CellType type)
{
    List<Point> nearby = new List<Point>();

    for (int i = -1; i <= 1; i++)
    {
        for (int j = -1; j <= 1; j++)
        {
            int newX = x + i;
            int newY = y + j;

            if (newX >= 0 && newX < gridSize && newY >= 0 && newY < gridSize)
            {
                if (grid[newX, newY] == type)
                {
                    nearby.Add(new Point(newX, newY));
                }
            }
        }
    }

    return nearby;
}
```

Życie zwierząt zawarte jest w metodach symbolizujących elementy ich życia:

1. `AnimalWander – zwierzę wędruje w losowym kierunku do wolnego pola mu przyległego`

```
private void AnimalWander(int x, int y, CellType animalType)
{
    List<Point> nearbyEmpty = FindNearby(x, y, CellType.Empty);
    if (nearbyEmpty.Count > 0)
    {
        int index = random.Next(nearbyEmpty.Count);
        Point emptyLocation = nearbyEmpty[index];
        grid[emptyLocation.X, emptyLocation.Y] = animalType;
        grid[x, y] = CellType.Empty;
    }
}
```

2. AnimalReproduce – zwierzę losowa rozmnaża się pod warunkiem, że wokoło znajdzie się wolne miejsce

```
private void AnimalReproduce(int x, int y, CellType animalType)
{
    float random_reproduce_chance = 0; ;
    if (animalType == CellType.Predator)
    {
        random_reproduce_chance = random_predator_reproduce_chance;
    } else if(animalType == CellType.Prey)
    {
        random_reproduce_chance = random_prey_reproduce_chance;
    }
    List<Point> nearbyEmpty = FindNearby(x, y, CellType.Empty);
    if(nearbyEmpty.Count > 0)
    {
        if (random.NextDouble() < random_reproduce_chance)
        {
            int index = random.Next(nearbyEmpty.Count);
            Point emptyLocation = nearbyEmpty[index];
            grid[emptyLocation.X, emptyLocation.Y] = animalType;
        }
    }
}
```

3. AnimalDie – zwierzę losowa umiera lub żyje dalej, szansa śmierci jest stałą i może być modyfikowana do obliczeń

```
private void AnimalDie(int x, int y, double extra_chance)
{
    if(random.NextDouble() < random_death_chance + (random_death_chance * extra_chance))
    {
        grid[x, y] = CellType.Empty;
    }
}
```

4. PreyLackFood – jeżeli roślinożerca jest otoczony przez zbyt dużą ilość innych roślinożerców to nie wystarczyło jedzenia i zwierzę umiera

```
private bool PreyLackFood(int x, int y)
{
    List<Point> nearbyPrey = FindNearby(x, y, CellType.Prey);
    if (nearbyPrey.Count > max_prey_in_9x9)
    {
        grid[x, y] = CellType.Empty;
        return true;
    }
    return false;
}
```

5. PredatorHunt – drapieżnik losowo próbuje polować na roślinożercę, jeżeli nie ma roślinożercy to na innego drapieżnika

```
private bool PredatorHunt(int x, int y)
{
    double huntChance = predator_success_rate;
    if (random.NextDouble() < huntChance)
    {
        List<Point> nearbyPrey = FindNearby(x, y, CellType.Prey);
        List<Point> nearbyPredator = FindNearby(x, y, CellType.Predator);

        if (nearbyPrey.Count > 0)
        {
            int index = random.Next(nearbyPrey.Count);
            Point preyLocation = nearbyPrey[index];
            grid[preyLocation.X, preyLocation.Y] = CellType.Empty;
            return true;
        }
        else if(nearbyPredator.Count > 0)
        {
            int index = random.Next(nearbyPredator.Count);
            Point predatorLocation = nearbyPredator[index];
            grid[predatorLocation.X, predatorLocation.Y] = CellType.Empty;
            return true;
        }
    }
    return false;
}
```

Roślinożercy mają następującą logikę:

```
private void PreyBehavior(int x, int y)
{
    if(!PreyLackFood(x, y))
    {
        AnimalWander(x, y, CellType.Prey);
        AnimalReproduce(x, y, CellType.Prey);
        AnimalDie(x, y, 0);
    }
}
```

1. PreyLackFood() – sprawdź czy nie ma za dużo roślinożerców w otoczeniu, jeżeli jest za dużo to zabrakło pożywienia i zwierzę ginie;
2. AnimalWander() – zwierzę wykonuje krok w losowym kierunku, pod warunkiem że znajdzie wokół siebie wolne pole;
3. AnimalReproduce() – zwierzę losowo się rozmnaża, pod warunkiem że jest wolne miejsce na młode;
4. AnimalDie() – zwierzę losowo umiera lub przeżywa.

Drapieżnicy mają następującą logikę:

```
private void PredatorBehavior(int x, int y)
{
    if(FindNearby(x, y, CellType.Prey).Count == 0)
    {
        AnimalWander(x, y, CellType.Predator);
    }
    if(PredatorHunt(x, y))
    {
        AnimalReproduce(x, y, CellType.Predator);
        AnimalDie(x, y, bonus_death_chance_if_fed);
    } else
    {
        AnimalDie(x, y, bonus_death_chance_if_hungry);
    }

}
```

1. Jeżeli wokół drapieżnika nie ma roślinożercy to zwierzę wędruje o 1 krok metodą AnimalWander(), warunkiem jest znalezienie wolnego pola wokoło;
2. Zwierzę próbuję polować na roślinożercę, w przeciwnym razie na innego drapieżnika;
    a. Jeżeli udało się upolować;
        i. Zwierzę próbuje się rozmnożyć AnimalReproduce();
        ii. Sprawdzamy czy zwierzę losowo umiera, jest na to mniejsze prawdopodobieństwo niż zwykle; prawdopodobieństwo jest zmodyfikowane o wartość `bonus_death_chance_if_fed`;
    b. Jeżeli nie udało się upolować;
        i. Sprawdzamy czy zwierzę losowo umiera, jest na to większe prawdopodobieństwo niż zwykle; prawdopodobieństwo jest zmodyfikowane o wartość `bonus_death_chance_if_hungry`;