Raport: Symulacja Ekosystemu - Trawa, Króliki i Lisy

Autor: Zofia Pietrucha 291734
Grupa: 4
Data: Czerwiec 2025

Wprowadzenie

Zadanie polegało na stworzeniu symulacji ekosystemu z trzema elementami: trawą, królikami i lisami. Zdecydowałam się na kod z biblioteką Ebiten.

Celem było pokazanie, jak w naturze działają relacje drapieżnik-ofiara i jak populacje wpływają na siebie nawzajem. Chciałam, żeby można było obserwować cykle populacyjne.

Jak zrozumiałam zadanie

Świat to siatka 80×60 komórek, gdzie każda może zawierać:

- **Puste pole** - czarna przestrzeń
- **Trawa** - rośnie, ma różne poziomy dojrzałości (różne odcienie zieleni)
- **Królik** - biała kropka (żółta jak się właśnie urodził dla lepszej identyfikacji nowych królików)
- **Lis** - czerwona kropka

Każdy element ma swoje zachowania - trawa rośnie, króliki jedzą trawę i się rozmnażają, lisy polują na króliki (też się rozmnażają).

Architektura

Główne struktury

```
type World struct {
    Grid         [][]EntityType
    Grass        map[Position]*Grass
    Rabbits      []*Rabbit
    Foxes        []*Fox
    Tick         int             // licznik czasu
    smartHunting bool           // czy lisy mogą widzieć na więcej kratek
}
```

Podzieliłam kod na pliki, dla łatwiejszej nawigacji:

- **main.go** - główna pętla i interfejs użytkownika
- **world.go** - świat i jego inicjalizacja
- **animals.go** - logika zwierząt
- **grass.go** - system trawy
- **rendering.go** - rysowanie wszystkiego
- **constants.go** - wszystkie parametry w jednym miejscu

Jak działa trawa

Trawa to fundament całego ekosystemu. Zrobiłam ją tak:

```
func (w *World) updateGrass() {
    // Istniejąca trawa rośnie
    for _, grass := range w.Grass {
        if grass.Amount < 100 {
            grass.Amount += 2  // rośnie o 2 punkty na tick
        }
    }

    // Nowa trawa ma 1% szansy pojawić się na pustym polu
    for attempts := 0; attempts < 10; attempts++ {
        x := rand.Intn(gridWidth)
        y := rand.Intn(gridHeight)

        if w.Grid[x][y] == Empty && rand.Float64() < 0.01 {
            // Nowa trawa!
        }
    }
}
```

Parametry trawy:

- Maksymalna "dojrzałość": 100 punktów
- Tempo wzrostu: 2 punkty na tick
- Szansa na nową trawę: 1% na każde puste pole
- Im więcej punktów, tym bardziej zielona

Króliki - trudniejsze niż myślałam

Króliki to było wyzwanie, bo mają dość skomplikowane zachowania:

Poruszanie się i jedzenie:

- 70% szansy na ruch każdy tick
- Mogą iść w 8 kierunkach (też na skos)
- Jedzą trawę, która ma minimum 5 punktów dojrzałości
- Za trawę dostają 40 energii
- Tracą 1 energię co sekundę

Rozmnażanie

Najpierw miałam bug, w którym króliki rozmnażały się wielokrotnie w jednym ticku i populacja eksplodowała. Musiałam zrobić system śledzenia par:

```
func (w *World) handleRabbitReproduction() {
    processedPairs := make(map[string]bool) // żeby nie przetwarzać tej samej pary dwa razy

    for _, rabbit := range w.Rabbits {
        if rabbit.Energy < 60 || rabbit.ReproduceCD > 0 {
            continue // za mało energii lub za wcześnie po ostatnim rozmnażaniu
        }

        partner := w.findPartner(rabbit, processedPairs)
```

```
        if partner != nil {
            w.makeBaby(rabbit, partner) // nowy królik!
        }
    }
}
```

Parametry królików:

- Próg energii do rozmnażania: 60
- Cooldown po rozmnażaniu: 180 ticków (30 sekund)
- 30% szansy na próbę rozmnażania
- Limit populacji: 50 (żeby nie było ich miliony)
- Nowo narodzone są żółte przez jakiś czas

Lisy - implementacja dwóch trybów

Tu było ciekawie, bo zrobiłam dwa tryby polowania:

Tryb podstawowy - "głupie" lisy:

```
func (w *World) moveFoxHunting(fox *Fox) {
    // Sprawdź sąsiednie pola
    // Jeśli jest królik obok - idź tam
    // Jeśli nie - ruszaj się losowo
}
```

Tryb zaawansowany - "mądre" lisy:

```
func (w *World) moveFoxSmart(fox *Fox) {
    targetRabbit := w.findNearestRabbit(fox.Position) // patrz w promieniu 3 pól

    if targetRabbit != nil {
        newPos := w.moveTowardsTarget(fox.Position, *targetRabbit)
        log.Printf("Lis zobaczył królika i go goni!")
    } else {
        // Nie ma królików w zasięgu, ruszaj się losowo
    }
}
```

Różnica jest ogromna! Mądre lisy są znacznie skuteczniejsze i populacja królików spada szybciej.

Parametry lisów:

- Zasięg widzenia (tryb smart): 3 komórki
- Próg energii do rozmnażania: 70
- Energia za zjedzenie królika: 50
- Limit populacji: 15
- 60% szansy na ruch każdy tick

Interfejs użytkownika

Uważam, że interfejs jest całkiem ładny i z możliwościami sterowania rozgrywką:

Kontrolki:

- **Spacja** - pauza/play (bardzo przydatne!)
- **1** - tryb rysowania królików (kliknięcie myszą)
- **2** - tryb rysowania lisów
- **0** - normalny tryb
- **V** - przełącz między trybami polowania lisów
- **S** - zapisz dane do CSV

Wizualizacja:

- Górna część - sama symulacja
- Dolna część - wykres populacji w czasie rzeczywistym
- Panel z przyciskami i statystykami

Wykres był chyba najtrudniejszy do zrobienia, ale było warto, poniważ od razu widać cykle populacyjne.

Eksport danych

Zrobiłam funkcję, która zapisuje wszystko do CSV:

```
func exportPopulationData(history []PopulationData) {
    // Zapisz z timestampem
    filename := fmt.Sprintf("ecosystem_data_%s.csv", timestamp)

    // Dodaj metadane jako komentarze
    file.WriteString("# Parametry symulacji:\n")
    file.WriteString(fmt.Sprintf("# - Rozmiar siatki: %dx%d\n", gridWidth,
gridHeight))
    // ... więcej info

    // Główne dane
    file.WriteString("Tick,Rabbits,Foxes,Grass,Timestamp\n")
    for _, data := range history {
        line := fmt.Sprintf("%d,%d,%d,%d,%s\n", ...)
        file.WriteString(line)
    }
}
```

Dodatkowo można zapisać sekwencję obrazków pokazującą ewolucję wykresu - przydatne do pokazania jak przebiegała symulacja czy stworzenia późniejszej animacji

Co odkryłam - wyniki eksperymentów

Cykle populacyjne - działają!

Rzeczywiście można zaobserwować te klasyczne cykle:

1. Dużo trawy → króliki się mnożą
2. Dużo królików → lisy mają co jeść i się mnożą
3. Dużo lisów → króliki giną
4. Mało królików → lisy umierają z głodu
5. Mało lisów → króliki znów się mnożą

I tak w kółko! Czasami jeden z gatunków całkiem wymiera i trzeba go "dosiać" ręcznie.

Balansowanie parametrów

Za małą szansę na trawę - króliki umierają z głodu. Za dużą - ich populacja eksploduje. Za mądre lisy - wybijają wszystkie króliki. Za głupie - nie mogą ich złapać.

Finalne parametry (po wielu próbach):

```
grassSpawnChance = 0.01    // 1% - trawa się pojawia, ale nie za szybko
rabbitMoveChance = 0.7     // 70% - króliki dość aktywne
foxMoveChance = 0.6        // 60% - lisy trochę mniej aktywne
reproduceChance = 0.3      // 30% - rozmnażanie nie za częste
maxRabbits = 50            // limit żeby nie było ich tysiące
maxFoxes = 15              // proporcjonalnie mniej lisów
```

Tryby polowania

Różnica między "głupimi" a "mądrymi" lisami jest ogromna:

- Głupie lisy: polują tylko w bezpośrednim sąsiedztwie, czasem można bezpiecznie przejść obok
- Mądre lisy: widzą w promieniu 3 pól i aktywnie gonią króliki

Z mądrymi lisami cykle są bardziej dramatyczne - populacja królików spada znacznie szybciej.

Największe problemy, które napotkałam

1. Bug z rozmnażaniem

Początkowo króliki mogły się rozmnażać wielokrotnie w jednym ticku. Populacja rosła wykładniczo i program się wieszał. Rozwiązałam to mapą `processedPairs`, która śledzi, które pary już się rozmnożyły w tym ticku.

2. Wydajność

Chciałam 60 FPS animacji, ale symulacja nie musi być tak szybka. Zrobiłam tak:

- Animacja: 60 FPS (płynnie)
- Logika symulacji: 6 FPS (co 10. klatkę)
- Zapisywanie danych: co 30 tików symulacji

3. Zarządzanie pamięcią

Dla długich symulacji musiałam:

- Ograniczyć historię do 150 punktów
- Używać map tylko tam gdzie trzeba (trawa jest "sparse")
- Uważać na garbage collection

4. Debugging

Dodałam dużo logów, szczególnie dla narodzin i śmierci:

```
log.Printf("Nowy królik urodził się na (%d,%d)! Łącznie królików: %d", x, y,
len(w.Rabbits))
log.Printf("Lis umarł z głodu! Zostało lisów: %d", len(w.Foxes)-1)
```

To bardzo pomagało zrozumieć, co się dzieje.

Ciekawe odkrycia

Scenariusze ekstremalne

1. **Wyginięcie lisów**: Króliki rosną do limitu, potem jest wielki kryzys żywnościowy
2. **Wyginięcie królików**: Lisy szybko umierają, trawa rośnie wszędzie
3. **Równowaga**: Czasami udaje się osiągnąć stabilny stan (rzadko!)

Wpływ kształtu

Początkowo króliki i lisy pojawiały się losowo, ale potem dodałam grupowanie - zwierzęta pojawiają się w małych grupach. To sprawia, że częściej się spotykają i mogą się rozmnażać.

Interesujące wzorce

- Czasami powstają "oazy" - obszary z dużą ilością trawy i królików, odizolowane od lisów
- Lisy czasem "wypasają" obszar z królików i muszą szukać nowych terenów łowieckich
- Przy bardzo niskiej populacji lisów króliki rozprzestrzeniają się szybko

Wnioski

Czego się nauczyłam

1. **Balansowanie to sztuka** - każdy parametr wpływa na wszystkie inne
2. **Debugging w symulacjach to wyzwanie** - trudno złapać co poszło nie tak
3. **Wizualizacja jest kluczowa** - bez wykresu ciężko było zobaczyć zmiany w symulacji
4. **Go + Ebiten to przyjemna kombinacja** - dość proste w użyciu, wydajne

O programowaniu

- Modułowość się opłaca - dzięki podziałowi na pliki kod był czytelny
- Dobre nazwy zmiennych to podstawa
- Logi są nieocenione przy debugowaniu
- Warto od razu myśleć o wydajności

Możliwe rozszerzenia (gdybym miała więcej czasu)

1. **Sezony** - zima = mniej trawy, lato = więcej
2. **Choroby** - epidemie redukujące populacje
3. **Grupowe zachowania** - stada królików, sfory lisów
4. **Genetyka** - szybsze króliki, lepsze lisy
5. **Więcej gatunków** - ptaki, owady, rośliny

Jak uruchomić symulację

Wymagania

Żeby uruchomić program, potrzebujesz:

- Go w wersji 1.21 lub nowszej (ja używałam 1.24)

- Linux (projekt był testowany pod Linuxem, zgodnie z wymaganiami)
- Biblioteki systemowe dla Ebiten (zazwyczaj są już zainstalowane)

Uruchamianie

```
# Najpierw pobierz zależności
go mod download

# Uruchom symulację
go run .
```

Podsumowanie

Ten projekt był naprawdę fajny! Na początku wydawał się prosty, ale każdy szczegół okazał się ważny. Najbardziej podobało mi się obserwowanie, jak symulacja "żyje" - czasem byłam w stanie przewidzieć co się stanie (np. że za chwilę będzie kryzys królików), a czasem mnie zaskakiwała.

Największą satysfakcją było zobaczenie pierwszych prawidłowych cykli populacyjnych jak w naturze.

P.S. Gdyby chciał Pan pobawić się parametrami to można je łatwo zmienić w `constants.go` i obserwować jak to wpływa na symulację!