Dokumentacja wewnętrzna projektu oraz struktura systemu

Inżynieria Oprogramowania

Garden Planner

Kierunek: Informatyka

Członkowie zespołu:

Jakub Darul

Mateusz Lamla

Michał Kierzkowski

Gliwice, 2025/2026

Spis treści

1 Struktura systemu **2**
1.1 Drzewo plików 2
1.2 Schemat graficzny struktury systemu 3
1.3 Opis systemu 4

2 Dokumentacja wewnętrzna **5**
2.1 Opis przebiegu prac nad projektem 5
2.2 Opis działania projektu 6

3 Instrukcja obsługi programu **7**

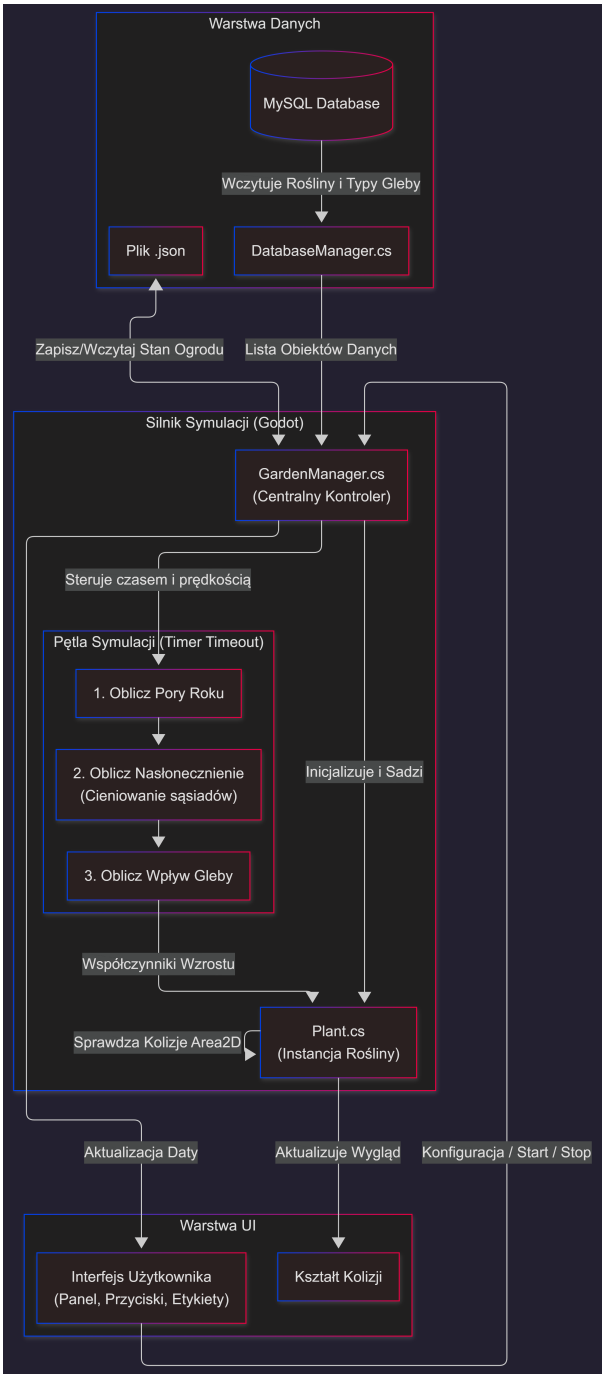
1 Struktura systemu

1.1 Drzewo plików

```
GardenPlanner/
├── Project/
│   ├── Database/
│   │   ├── DatabaseManager.cs          Obsługa bazy danych
│   │   ├── docker-compose.yml          Konfiguracja kontenera DB
│   │   └── init.sql                    Struktura tabel
│   ├── Data/
│   │   └── DataModels.cs               Definicje struktur
│   ├── Scenes/                         Sceny
│   │   ├── Main.tscn                   Główna scena gry
│   │   └── Plant.tscn                  Szablon rośliny
│   ├── Scripts/                        Klasy
│   │   ├── GardenManager.cs            Obsługa działania ogrodu
│   │   └── Plant.cs                    Wzrost pojedynczej rośliny
│   ├── Textures/                       Zasoby graficzne
│   │   └── Plants/                     Tekstury roślin
│   ├── project.godot                   Konfiguracja Godot
│   └── GardenPlanner.csproj            Konfiguracja .NET
└── README.md
```

Struktura projektu została zaprojektowana w taki sposób, aby prace nad warstwą wizualną, logiczną oraz bazodanową były niezależne od siebie. Dzięki temu dwie osoby mogą pracować nad różnymi częściami projektu bez obaw, że będą sobie przeszkadzały. Zadbaliśmy o wyraźny podział na logikę, dane i zasoby.

1.2 Schemat graficzny struktury systemu



Rysunek 1: Schemat graficzny struktury

Powyższy schemat obrazuje przepływ danych oraz zależności, które występują w naszym systemie symulacji. Cały system możemy podzielić na 3 warstwy:

- **Warstwa Danych**: część odpowiedzialna za zarówno dane pochodzące z utworzonej przez nas bazy, jak i za zapisy stanu symulacji.
- **Warstwa Logiki**: część przechowująca główny element całego naszego systemu (GardenManager). Zarządza ona pętlą czasu, steruje cyklem życia roślin oraz przekazuje informacje o warunkach dla roślin, dzięki czemu mogą rosnąć zgodnie ze swoimi wymogami.
- **Warstwa Prezentacji**: część odpowiedzialna za widok całej symulacji. Odświeża i wyświetla zmiany zachodzące w warstwie logiki.

1.3 Opis systemu

Cały system, który utworzyliśmy w ramach projektu, możemy streścić do paru najważniejszych jego cech:

- **Silnik i język**: Projekt oparliśmy na silniku Godot 4.3 z wykorzystaniem języka C# (.NET) dla lepszej wydajności obliczeń symulacyjnych.
- **Baza danych**: Wykorzystaliśmy relacyjną bazę danych MySQL do przechowywania stałych parametrów i danych (gatunki roślin, typy gleby).
- **Konteneryzacja**: Użyliśmy Docker oraz Docker Compose do łatwego i szybkiego uruchamiania środowiska bazodanowego.
- **Format zapisów**: Stan symulacji można zapisać w formacie JSON.
- **Wymagania**: System do działania wymaga zainstalowanego środowiska .NET oraz Dockera.

2 Dokumentacja wewnętrzna

2.1 Opis przebiegu prac nad projektem

Prace nad systemem realizowaliśmy z wykorzystaniem techniki Kanban. Dzięki temu podejściu, mogliśmy na bieżąco monitorować postępy oraz zadbać o równomierny podział zadań w zespole.
Całość pracy nad projektem podzieliliśmy na łącznie 5 sprintów. Progres był regularnie aktualizowany za pomocą strony **Notion.so**.

Krótki opis każdego ze sprintów:

- **Sprint 1**: zadania organizacyjne i przygotowawcze do dalszej pracy nad projektem (instalacja i konfiguracja narzędzi).
- **Sprint 2**: przygotowanie najbardziej podstawowych elementów programu takich jak np. panele czy sceny.
- **Sprint 3**: dopracowanie kwestii roślin, praca nad działaniem symulacji i tworzenia ogrodu, poprawa błędów.
- **Sprint 4**: dodanie drobnostek estetycznych, dodanie opcji wczytywania i zapisywania stanu ogrodu, usprawnienie kwestii kolizji roślin.
- **Sprint 5**: implementacja pór roku, sporządzenie dokumentacji, przeprowadzanie testów, ostateczne modyfikacje części wizualnej.

Prowadziliśmy również arkusz, który służył nam do monitorowania estymat i faktycznego czasu pracy nad poszczególnymi zadaniami ze sprintów. Do każdego ze sprintów dołączyliśmy wykres, który przedstawia sumy godzin spędzonych oraz oczekiwanych przy kolejnych etapach tworzenia projektu. Dodatkowo, przygotowaliśmy tabelkę, która służy jako końcowe podsumowanie czasu pracy.

ZESPÓŁ	SPRINT 1		SPRINT 2		SPRINT 3		SPRINT 4		SPRINT 5		SUMA	
	Estymata	Wykonanie	Estymata	Wykonanie	Estymata	Wykonanie	Estymata	Wykonanie	Estymata	Wykonanie	Estymata	Wykonanie
Jakub Darul	5	5	8	7	10	10	6	5,5	2	2	31	29,5
Michał Kierzkowski	8	8	8	7,5	9	7,5	5	4	4	4	34	31
Mateusz Lamla	4	5	11	9	9	7,5	6	6	4	4	34	31,5
Wszyscy							3	3	14	14	17	17

Rysunek 2: Końcowe podsumowanie ilości godzin

Na potrzeby przechowywania i aktualizowania projektu na bieżąco wraz z postępem prac, utworzyliśmy repozytorium **GitHub**. Struktura repozytorium została przedstawiona na schemacie w podpunkcie **1.1**.

2.2 Opis działania projektu

Cały system opiera się na pętli symulacyjnej zarządzanej przez klasę **GardenManager**, która koordynuje rozwojem całego ogrodu.

Opis paru kluczowych mechanik:

- **Mechanika czasu i pór roku**: symulowanie rozrostu ogrodu przebiega za sprawą funkcji *SimulateStep* oraz timera. Główna klasa koordynująca ogrodem przechowuje globalny licznik miesięcy. Przy każdej zmianie miesiąca, wysyłany jest sygnał do wszystkich roślin. Zmiana pory roku wpływa na współczynniki wzrotu roślin oraz na warstwę wizualną.
- **Mechanika zacienienia i kolizji**: każda z roślin posiada swój promień, który symbolizuje szerokość rozrostu. Za pomocą $ColissionShape2D$ wykrywane jest sąsiedztwo. Jeśli obszar korony rośliny nakłada się na obszar innej rośliny, następuje obliczenie "tłoku". Każda z roślin posiada swój wzrost, dzięki czemu możemy stwierdzić, jaki cień rzucają. Obniża to parametr *SunLevel* dla niższych sąsiadów, co zmienia wartość rozrostu.
- **Parametryzacja gleby**: właściwości każdej z możliwy gleb zapisane są w bazie danych. Charakteryzują je parametry takie jak $RetencjaWody$, $PoziomOdzywczy$ czy $Zyznosc$. W zależności od typu gleby w ogrodzie, rośliny mają modyfikowane mnożniki prędkości wzrostu.
- **Zapis i wczytywanie zapisów**: program umożliwia zapisywanie aktualnego stanu ogrodu w pliku JSON, który następnie możemy wczytać z powrotem do programu. Zapewnia nam to możliwość późniejszych modyfikacji ogrodu bez konieczności tworzenia go od nowa.
- **Mechanika dewastacji**: w trakcie trwania symulacji, istnieje szansa na dewastacje losowego obszaru naszego ogrodu poprzez na przykład rycie dzików czy podkopy kretów. W wyniku takiej dewastacji, usuwane są rośliny rosnące na danym obszarze. Teren zdewastowany utrzymuje się przez 3 miesiące.

3 Instrukcja obsługi programu

Aby uruchomić symulator, użytkownik potrzebuje na swoim komputerze mieć zainstalowanego Docker'a oraz Godot'a.
Interfejs symulatora jest graficzny. Komunikacja z użytkownikiem przebiega jedynie za pomocą myszki, jako że wszystkie dostępne opcje mają swoje przypisane przyciski.

- Na początku należy wybrać wymiary ogrodu (zakładamy, że teren jest prostokątem) oraz typ gleby.
- Następnie użytkownik, aby móc rozpocząć symulację, musi umieścić na planszy ogrodu minimalnie jedną roślinkę.
- Symulacja rozpoczyna się po naciśnięciu odpowiedniego guzika, co powoduje rozrost rośliny oraz upływ czasu (na górze ekranu znajduje sie licznik miesięcy i lat od rozpoczęcia symulacji).
- W każdej chwili użytkownik ma możliwość zatrzymania symulacji, w celu np. dodania/usunięcia sadzonek lub przeanalizowania rozrostu obecnych już roślin.

Po zakończeniu symulacji przez użytkownika, ma on możliwość zapisania do pliku JSON aktualnego stanu ogrodu, aby móc swój plan zastosować w swoim prawdziwym ogrodzie z możliwością modyfikacji planu w przyszłości.