**Міністерство освіти і науки України**

**Національний технічний університет**

**«Дніпровська політехніка»**

|  |
| --- |
| Інститут електроенергетики |
| (інститут) |

|  |
| --- |
| Факультет інформаційних технологій |
| (факультет) |

|  |  |
| --- | --- |
| **Кафедра** | Програмного забезпечення комп’ютерних систем |
|  | (повна назва) |

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

**кваліфікаційної роботи ступеня**

|  |
| --- |
| *бакалавра* |

(назва освітньо-кваліфікаційного рівня)

|  |  |  |  |
| --- | --- | --- | --- |
| **студента** | | *Бондаря Андрія Павловича* | |
|  | | (ПІБ) | |
| **академічної групи** | | | *122-17-3* |
|  | | | (шифр) |
| **напряму підготовки** | | | *122 Комп’ютерні науки* |
|  | | | (код і назва напряму підготовки) |
| **на тему:** | *Розробка ігрового додатку для адаптації дій гравця* | | |
| *з використанням бібліотек RayLib v3.5, OpenMP 2.0 на мові C++.* | | | |
|  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Керівники** | **Прізвище, ініціали** | **Оцінка за шкалою** | | **Підпис** |
| **рейтинговою** | **інституційною** |
| кваліфікаційної роботи | *ПІБ* |  |  |  |
| **розділів:** |  |  |  |  |
| спеціальний | *ПІБ* |  |  |  |
| економічний | *Касьяненко Л.В.* |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Рецензент** |  |  |  |  |
| **Нормоконтролер** | *Гуліна І.Г.* |  |  |  |

**Дніпро**

**2021**

**Міністерство освіти і науки України**

**НТУ «Дніпровська політехніка»**

**ЗАТВЕРДЖЕНО:**

Завідувач кафедри

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| програмного забезпечення комп’ютерних систем | | | | | | | | |
| (повна назва) | | | | | | | | |
|  | | |  | І.М. Удовик | | | | |
| (підпис) | | |  | (прізвище, ініціали) | | | | |
| « » |  |  | | | 20 |  | року |

**ЗАВДАННЯ**

**на дипломний проект**

|  |
| --- |
| *бакалавра* |

(назва освітньо-кваліфікаційного рівня)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| студенту | *122-17-3* |  | *Бондару Андрію Павловичу* | | | | | | |
| (група) | | | | (прізвище та ініціали) | | | | |
| **Тема дипломного проекту** | | | | | *Розробка ігрового додатку для адаптації* | | | | | | |
| *дій гравця з використанням бібліотек RayLib v3.5, OpenMP 2.0 на мові C++* | | | | | | | | | | | |
|  | | | | | | | | | | | |
|  | | | | | | | | | | | |
| затверджена наказом ректора ДВНЗ «НГУ» від | | | | | | *. .2021 р.* | № | *-л* | | |

|  |  |  |
| --- | --- | --- |
| **Розділ** | **Зміст виконання** | **Термін виконання** |
| ***Спеціальний*** | ***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.***  ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми*** | ***12.05.2021 р.*** |
| ***Економічний*** | ***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки*** | ***26.05.2021 р.*** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Завдання видав | |  | | | |  | | *ПІБ* |
|  | (підпис) | | | |  | | (посада, прізвище, ініціали) | |
| Завдання прийняв до виконання | | |  |  | | | *Бондар А.П.* | |
|  | | | (підпис) | |  | | (прізвище, ініціали) | |

Дата видачі завдання:  *. .2021 р.*

Термін подання дипломного проекту до ДЕК ***\_ \_. .2021 р.\_\_***

**РЕФЕРАТ**

Пояснювальна записка: \_\_\_ с., \_\_\_ рис., \_\_\_ табл., \_\_ дод., \_\_\_ джерел.

Об'єкт розробки: ігровий додаток-стратегія з використанням машинного навчання.

Мета дипломного проекту: створення ігрового додатку зі штучним інтелектом що зможе адаптуватися під довільну ігрову ситуацію та допоможе розвинути стратегічне мислення гравця.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні програмного додатка, що надає можливість покращити стратегічні та тактичні навички.

Актуальність інформаційної системи визначається збільшеною тенденцією на розробки з використанням машинного навчання, що оптимізують дії щодо адаптації до довільних ситуацій, демонструють можливості та гнучкість алгоритмів.

Список ключових слів: ІГРОВИЙ ДОДАТОК, КОМП’ЮТЕР, МАШИННЕ НАВЧАННЯ, ПЕРСЕПТРОН, АЛГОРИТМ, ШТУЧНИЙ ІНТЕЛЕКТ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ.

**(ДОБАВИТЬ ПЕРЕВОД НА ИНГЛИШ)**

**ЗМІСТ**

РЕФЕРАТ…………………………………………………………………………….

ABSTRACT………………………………………………………………………….

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ……………………………………………...

ВСТУП……………………………………………………………………………….

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

* 1. Загальні відомості з предметної галузі…………………………………
  2. Призначення розробки та галузь застосування………………………..
  3. Підстава для розробки…………………………………………………..
  4. Постановка завдання……………………………………………………….
  5. Вимоги до програми або програмного виробу………………………….
     1. Вимоги до функціональних характеристик………………………………
     2. Вимоги до інформаційної безпеки…………………………………………
     3. Вимоги до складу та параметрів технічних засобів……………………….
     4. Вимоги до інформаційної та програмної сумісності ……………………

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ..

* 1. Функціональне призначення системи……………………………………..
  2. Опис застосованих математичних методів………………………………
  3. Опис використаних технологій та мов програмування…………………
  4. Опис структури системи та алгоритмів її функціонування ……………….
  5. Обґрунтування та організація вхідних та вихідних даних програми….
  6. Опис розробленої системи .............…………………………………………
     1. Використані технічні засоби…….………………………………………….
     2. Використані програмні засоби……………………………………………..
     3. Виклик та завантаження програми…………………………………………
     4. Опис інтерфейсу користувача………………………………………………

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ………………………………………………

* 1. Розрахунок трудомісткості та вартості розробки програмного продукту.
  2. Рахунок витрат на створення програми……………………………………

ВИСНОВКИ…………………………………………………………………………..

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………………….

Додаток А. Код програми……………………………….…………………………..

Додаток Б. Відгук керівника економічного розділу……………………………….

Додаток В. Перелік файлів на диску……………………………………………….

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ**

ООП – об’єктно-орієнтоване програмування;

ШІ – штучний інтелект;

API – прикладний програмний інтерфейс;

ПК – персональний комп’ютер;

ПЗ – програмне забезпечення;

IDE – інтегроване середовище розробки

ЕОМ – електронно-обчислювальна машина

**(ДОБАВИТЬ ВСТУП)**

**РОЗДІЛ 1**

**АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ**

* 1. **Загальні відомості з предметної галузі**

Комп`ютерні ігри міцно закріпили за собою звання відмінного засобу не тільки відпочинку, але і навчання. Вони здатні сформувати у гравця принципи мислення для вирішення поставлених завдань, які потім можливо застосувати для вирішення реальних проблем, і все це - в доступній, ігровій формі, за якої за виконання завдань гравець отримує задоволення, що закріплює отриманий досвід.

Один з жанрів комп’ютерних ігор, що дає творчу свободу та широкий набір інструментів для реалізації власних ідей – Sandbox, («пісочниця» у перекладі). Яскраві представники цього жанру:

* Minecraft (з англ. «шахтарське ремесло») – пісочниця з виглядом від першої/третьої особи. Гравець отримує у своє розпорядження трьохмірний відкритий, процедурно-генерований світ, що повністю складається з кубічних «блоків», які можна видобути та створювати власні структури у світі, або перетворити на блоки іншого типу. Можливість змінити гру за допомогою модифікацій розширюють її ще більше.



Рис. 1.1. Ігровий процес Minecraft

* Factorio – пісочниця, що виконана в жанрі ізометричного економічного симулятора з елементами стратегії. Основою ігрового процесу є створення великих виробничих ланцюжків і повсюдна автоматизація – персонаж гравця хоч і може виробляти необхідні компоненти та пристрої самостійно, але швидкість його виробництва не йде ні в яке порівняння з виробничими можливостями налагодженої фабрики. Забруднення, яке розповсюджується фабрикою, турбує мешканців планети – вигаданих членистоногих істот, та змушує збиратися в групи і нападати на джерело забруднення.



Рис. 1.2. Ігровий процес Factorio

Стратегія, як жанр ігор, існував задовго до створення перших комп’ютерів. Яскравий приклад – шахи, абстрактна стратегічна покрокова гра. Стратегії у реальному часі (RTS) відрізняються від покрокових тим, що події у них розвиваються одночасно, що вимагає від гравців швидкої реакції та стратегічного мислення.



Рис. 1.3. Ігровий процес StarCraft 2

RTS зазвичай відтворюють військові дії та розбудову військових баз для здобуття перемоги над противником, мають специфічний інтерфейс. Їм притаманні численні ігрові умовності, такі як врівноваження можливостей протиборчих сторін, нереалістично короткі терміни проведення боїв і будівництва, видобування абстрактних ресурсів що потребуються для створення бойових одиниць та будівель.



Рис. 1.4. Ігровий процес Supreme Commander

У більшості стратегій гра проходить як протистояння двох або більше сторін. Гравці-люди не завжди можуть скласти компанію в грі, через особливості механіки гри або з інших причин. Для подібних випадків передбачено заміщення реального гравця комп'ютерним, ШІ-суперником.

Ігровий ШІ в основному займається вибором дій в залежності від поточних умов. У традиційній літературі по ШІ це називається "інтелектуальними агентами". Агентом зазвичай є персонаж гри, але це може бути і машина, робот або навіть щось більш абстрактне - ціла група сутностей, країна або цивілізація. У будь-якому випадку це об'єкт, що стежить за своїм оточенням, що приймає на підставі нього рішення і діє відповідно до цих рішень.

Задачі ШІ реального світу, особливо ті що є актуальними сьогодні, зазвичай зосереджені на «сприйнятті». Наприклад, безпілотні автомобілі повинні отримувати зображення знаходиться перед ними дороги, комбінуючи їх з іншими даними (радара й лідара) та намагаючись інтерпретувати те, що вони бачать. Зазвичай це завдання вирішується машинним навчанням, яке особливо добре працює з великими масивами зашумлених даних реального світу (наприклад з фотографіями дороги перед автомобілем або декількома кадрами відео) і надає їм якесь значення, отримуючи семантичну інформацію, наприклад «через 10 метрів переді мною є ще одна машина».

Ігри відрізняються тим, що для вилучення інформації їм не потрібна окрема система, оскільки вона є невід'ємною частиною симуляції. Немає необхідності виконувати алгоритми розпізнавання зображень, щоб виявити ворога перед собою: гра знає, що там є супротивник і може передати цю інформацію безпосередньо процесу прийняття рішень.

Зазвичай передбачається, що гра повинна розважати і кидати гравцеві виклик, а не бути «оптимальною» – тому навіть якщо і можна навчити агентів протистояти гравцям найкращим чином, то найчастіше гейм дизайнерам потрібно від них зовсім іншого. Яскравим прикладом цього принципу є ігри жанру шутер, де рефлекси навіть професійних геймерів-людей ледве зможуть протистояти ботам-«шахраям», що заздалегідь знають де знаходиться опонент та ціляться з точністю, недосяжною для людей.

Розробники мають великий арсенал інструментів, щоб зробити дії ІШІ більш схожими на людські, без необхідності повністю копіювати людську поведінку. Для жанру стратегій це можуть бути спеціальні маркери (AI markers) на ігровій мапі, за допомогою яких ШІ може направляти свої юніти, жорстко прописані скрипти опису дій, що необхідно виконати, або ж трохи «шахрайства», як-то видача додаткових ігрових ресурсів або відсутність туману війни (область що приховує юніти супротивника та потребує розвідки) для «зору» ШІ.



Рис. 1.5. Графи маршрутів юнітів (AI markers) для мапи гри Supreme Commander

**1.2. Призначення розробки та галузь застосування.**

Розробка ігрового додатку «Стратегія у реальному часі з використанням машинного навчання для адаптації до дій гравця».

Основні терміни та ключові слова:

Машинне навчання – підгалузь штучного інтелекту в галузі інформатики, яка застосовується для надання комп'ютерам здатності «навчатися», покращуючи свою продуктивність у певній задачі з поточних даних, замість того, щоби бути програмованими явно.

Штучний інтелект – здатність системи здобувати, обробляти та застосовувати знання та вміння подібно до того, як це виконує людина.

**1.3. Підстави для розробки.**

Згідно наказу ректора Національного технічного університету «Дніпровська політехніка» № \_\_\_\_ від \_\_.\_\_ 2021 р.

**1.4. Постановка завдання.**

Мета завдання – розробити стратегію у реальному часі, що розрахована на одного користувача, та буде адаптуватися під дії гравця. Для цього програма має використовуючи алгоритми машинного навчання, такі як штучні нейронні мережі та пошук оптимального шляху з врахуванням вагів.

Ландшафт ігрової мапи має створюватися процедурною генерацією на основі шуму Перліна.

Пошук шляху юнітів ШІ-супротивника має базуватися на цільовому векторному полю, що генерується з врахуванням типу ландшафту та співвідношенням отриманої та нанесеної шкоди у обчислювальних тайлах.

Економічна складова стратегії має залежати від кількості тайлів (клітин мапи ігрового поля), що захоплено гравцем та його супротивником. Захоплення тайлів має відбуватися поступово, шляхом створення допоміжних юнітів-будівель.

Області мапи, що ще не розвідані юнітами, мають бути покриті «туманом війни» та підлягают розвідці як зі сторони гравця, так і зі сторони ШІ-супротивника.

Розподілення видобутих ресурсів супротивника на створення нових юнітів має здійснюватися ШІ-агентом на основі штучних нейронних мереж, що мусить самостійно навчатись згідно з поточними обставинами та результуючою ефективністю своїх юнітів.

Розроблений ігровий додаток спрямований на розвиток стратегічного мислення у гравця, шляхом пошуку оптимальної стратегій та адаптації до мінливої ситуації.

**1.5. Вимоги до програми або програмного виробу**

**1.5.1. Вимоги до функціональних характеристик.**

Процес введення даних та команд до програми проводиться за допомогою клавіатури та миші. Результуючі значення, що розраховані на основі введених користувачем даних, мають бути приведені до діапазону можливих значень: як-то, користувач не може створити юніта за межами ігрового поля.

Цикл обчислення положень та атрибутів об’єктів також має враховувати ці діапазони. Наприклад, юніти не можуть вийти за межі ігрового поля, а більшість атрибутів не можуть дорівнювати від’ємному значенню.

Виведення даних проводиться у графічному вигляді за допомогою монітору комп’ютера.

**1.5.2. Вимоги до інформаційної безпеки.**

Програма у фінальному вигляді скомпільована та не потребує зв’язку з мережею Інтернет, тому додаткових вимог до безпеки не потребує. Супутні файли (DLL-бібліотеки, файли графічних ресурсів тощо) мають зберігати своє положення відносно виконуваного файлу програми задля уникнення можливих несправностей програми.

**1.5.3. Вимоги до складу та параметрів технічних засобів**

Для користувача є важливим мати систему з відеокартою, що сумісна з OpenGL версії 3.3 або вище. Операційною системою має бути Windows 10 версії 64 bit. Мінімально можлива комплектація ЕОМ:

* CPU: сімейства Intel Celeron;
* GPU: 3D адаптер Nvidia, Intel HD або AMD/ATI;
* RAM: 2048 МБ.
* VRAM (відеопам’ять): 128 МБ;
* Вільне місце на накопичувачі: 128 МБ;

**1.5.4. Вимоги до інформаційної та програмної сумісності.**

Програма має бути реалізована на мові C++ за допомогою IDE Visual Studio 2019. Допоміжні бібліотеки, використані для розробки:

* RayLib v3.5, безкоштовна бібліотека на мові C з вільним вихідним кодом, що спрямована на допомогу у створені відеоігр.
* OpenMP 2.0, відкритий стандарт та бібліотека для створення застосунків з використанням паралельних обчислень.

**РОЗДІЛ 2.**

**ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ**

**2.1. Функціональне призначення системи**

Результатом даної кваліфікаційної роботи має бути ігровий застосунок сумісний з десктопними пристроями. Функціональним призначенням готового додатку є демонстрація можливостей використання машинного навчання у створенні ігрових застосунків зі стратегічною складовою для розвитку мислення гравця.

**2.2. Опис застосованих математичних методів**

(Расписать использованные геом. формулы)

Функції оновлення стану юнітів застосовують формули Евклідової відстані (2.1) для пошуку найближчої доступної цілі з наявних.

|  |  |
| --- | --- |
|  | (2.1) |

де – відстань між точками *p* та *q.*

Ландшафт ігрової мапи процедурно генерується за допомогою шуму Перліна, що є математичним алгоритмом з генерування процедурної текстури псевдо-випадковим методом (рис 2.2.1).

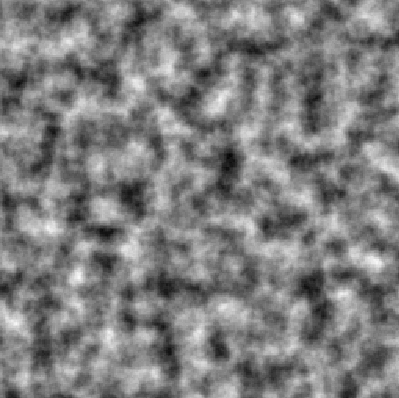


Рис. 2.2.1 Приклад згенерованої чорно-білої текстури карти висот

Юніти, підконтрольні ШІ, рухаються у напрямі, що задається векторним полем, специфічним для кожного типа юнітів. Векторне поле, у свою чергу, генерується залежно від положення головної споруди гравця (знищення якої означає для нього поразку), прохідності ландшафту, стан розвідки, вплив типу ландшафта на швидкість юніта, а також минулих успішних та невдалих бойових дії супроти юнітів гравця.

Ігровий додаток також використовує штучну нейронну мережу на основі персептрона, що приймає на вхідне значення процентне співвідношення знайдених ШІ бойових юнітів гравця, та видає співвідношення юнітів, що будуть створені ШІ. Для кожного типу юнітів ШІ-супротивника зберігаються статистичні дані, такі як отримані та завдані втрати у боях, що дозволяє кожний умовний ігровий цикл вираховувати бойову ефективність окремих типів юнітів. Ці дані персептрон застосовує для зворотного поширення помилки, задля корекції та оптимізації вихідних значень.

**2.3. Опис використаних технологій та мов програмування**

Даний ігровий додаток був розроблений з використанням таких технологій:

* C++
* RayLib 3.5
* siv::PerlinNoise
* STL

C++ – об’єктно-орієнтована мова програмування зі статичною типізацією.

C++ базується на ідеї забезпечення як безпосередньої відповідності вбудованих операцій і типів апаратних засобів для забезпечення ефективного використання пам'яті та ефективних низькорівневих операцій, так і доступних та гнучких механізмів абстракції для забезпечення типів, що визначаються користувачем, з однаковою підтримкою нотацій, спектром використання та продуктивністю порівнянною зі вбудованими типами. З роками застосування цих простих ідеалів призвело до більш загального, ефективного та гнучкого набору засобів. Результат підтримує синтез стилів програмування, які можуть бути одночасно ефективними та елегантними. Дизайн С ++ був зосереджений на техніках програмування, що займаються такими основними поняттями, як пам'ять, мінливість, абстракція, управління ресурсами, вираження алгоритмів, пошук помилок та модульність. Це найважливіші проблеми системного програміста і в цілому програмістів високопродуктивних та обмежених у ресурсах систем.

[**ДОБАВИТЬ НОМЕР ССЫЛКИ НА КНИГУ**]

Ця мова програмування була обрана за те, що вона дозволяє програмісту самостійно керувати виділенням та звільненням пам’яті, що дозволяє створювати високопродуктивний код (рис. 2.3.1.), проте це також потребує від програміста розуміння, що він збирається зробити. **[ДОБАВИТЬ ССЫЛКУ НА ГИТХАБ С СРАВНЕНИЕМ]**

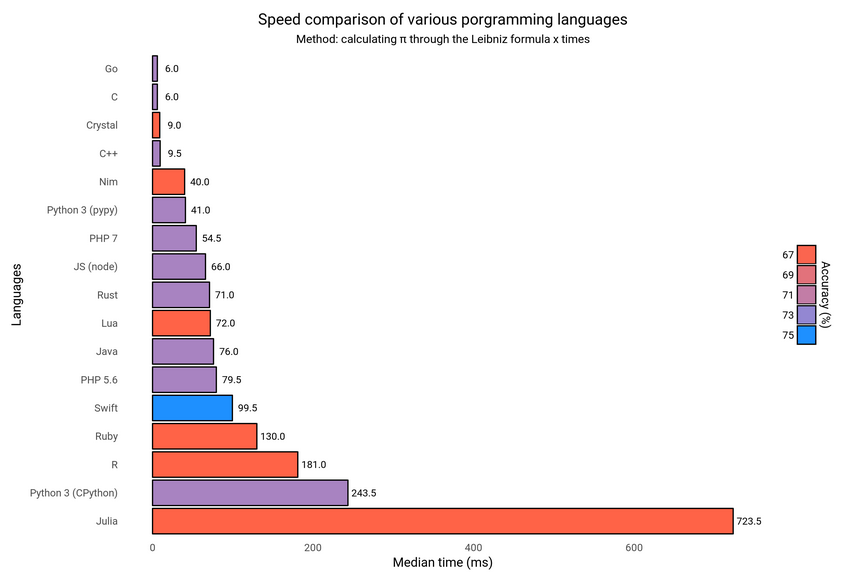


Рис. 2.3.1. Графік порівняння часу виконання формули Лейбніца на різних мовах програмування

Статична типізація, хоч і може здаватися непотрібним обмеженням, також є своєрідною першою лінією захисту програміста від очевидних помилок: середа розробки та компілятор не дадуть запустити цей код та явно підкажуть, де була зроблена помилка.

RayLib – кросс-платформенна бібліотека з відкритим вихідним кодом, що спрямована на полегшення розробки ігрових додатків, написана на мові C. Бібліотека має модульну структуру (рис 2.3.1.), кожен з модулів самодостатньо організований відповідно до свого призначення, що дозволяє уникнути створення небажаної рекурсивної залежності, що зазвичай притаманна бібліотекам, написаним на C.

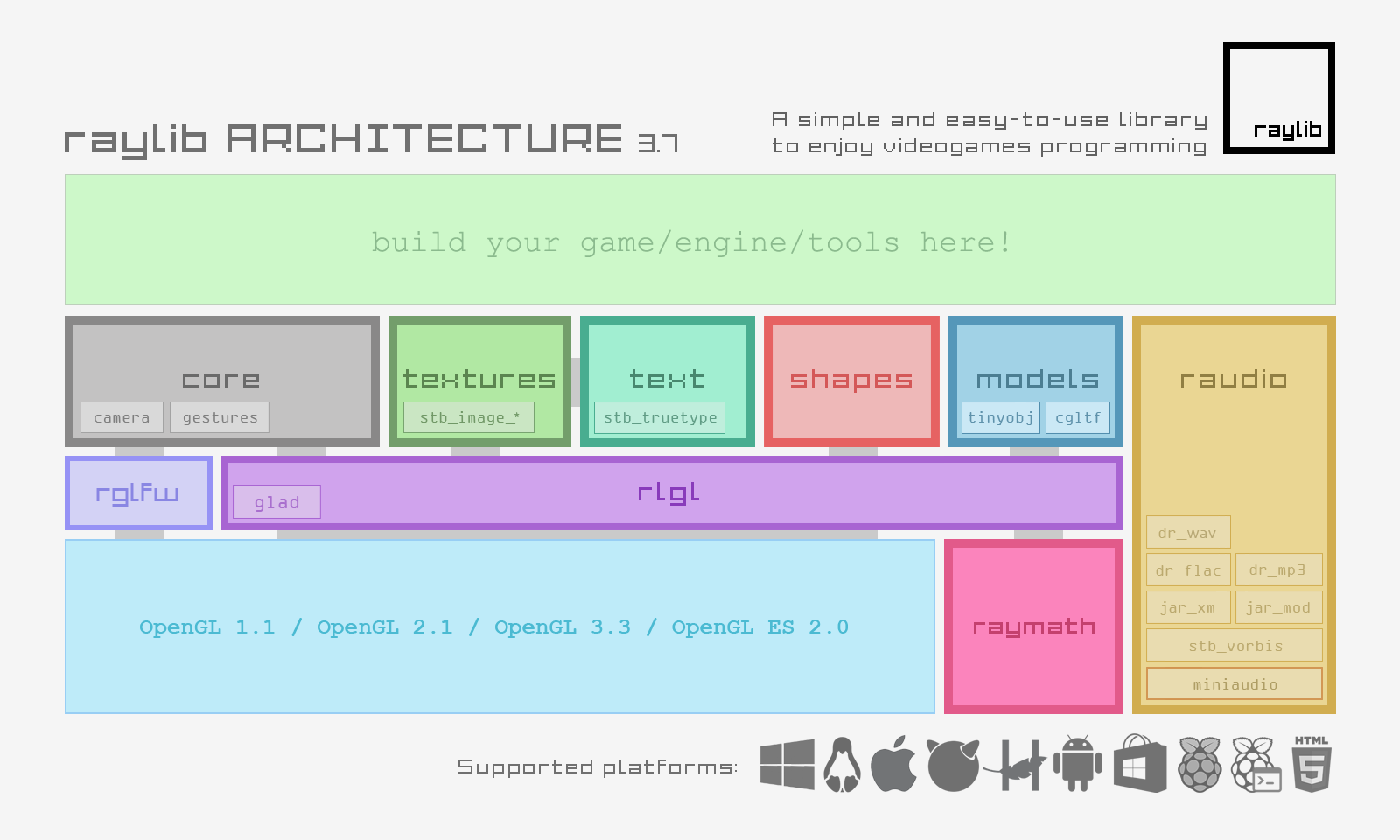


Рис 2.3.1. Архітектура бібліотеки RayLib

RayLib складається з семи основних модулів:

* core, управління вікном, графічним контекстом OpenGL та введенням команд;
* rlgl, обгортка-API над OpenGL;
* shapes, функції візуалізація двомірних примітивів;
* textures, завантаження та управління текстурами та зображеннями;
* text, завантаження шрифтів та візуалізація тексту;
* models, завантаження та візуалізація тривимірних моделей;
* raudio, управління аудіопристроями та завантаження звуків.

Ці сім модулів мають спільний заголовок, який називається raylib.h. Усі функції API визначені в цьому заголовку, незважаючи на те, що вони внутрішньо розділені на сім модулів, тому користувачеві потрібно лише включити raylib.h, щоб отримати доступ до всіх функцій raylib. Інші бібліотеки часто використовують заголовок для кожного модуля (тому користувачі можуть вибрати ті, які їм потрібні), але це ускладнює залежності. Це простий підхід, який робить raylib легшим для початківців та досвідчених користувачів.

Крім основних модулів, бібліотека містить також допоміжні модулі:

* raymath, функції обчислень векторів, матриць та кватерніонів.
* camera, система тривимірної камери, що може працювати у режимах: вільному, від першої особи, від третьої особи та розширеному.
* gestures, проста система IMGUI (Immediate Mode GUI) з декількома елементами управління.
* physac, двовимірна фізична бібліотека з функціями для обчислення процесів зіткнень, розширення, динаміки тощо.

[**ВСТАВИТЬ ССЫЛКУ НА ВИКИ РЕЙЛИБА**]

RayLib доступний для використання на великій кількості мов програмування, у тому числі: C, C++, C#, D, Go, Rust, Lua, Python, Java та інші. [**ВСТАВИТЬ ССЫЛКУ НА СПИСОК ЯЗЫКОВ**]

«OpenMP (Open Multi-Processing) - це набір директив компілятора, бібліотечних процедур і змінних оточення, які призначені для програмування багатопоточних додатків на багатопроцесорних системах із загальною пам'яттю (SMP-системах).

Перший стандарт OpenMP був розроблений в 1997 році як API, орієнтований на написання легко переносних багатопоточних додатків. Спочатку він був заснований на мові Fortran, але пізніше включив в себе мови С і С++.

Інтерфейс OpenMP став однією з найбільш популярних технологій паралельного програмування. OpenMP успішно використовується як при програмуванні суперкомп'ютерних систем з великою кількістю процесорів, так і в настільних користувальницьких системах або, наприклад, в Xbox 360.

Розробку специфікації OpenMP ведуть кілька великих виробників обчислювальної техніки і програмного забезпечення, чия робота регулюється некомерційною організацією «OpenMP Architecture Review Board» (ARB).

У OpenMP використовується модель паралельного виконання «розгалуження-злиття». Програма OpenMP починається як єдиний потік виконання, що зветься початковим потоком. Коли потік зустрічає паралельну конструкцію, він створює нову групу потоків, що складається з себе і деякого числа додаткових потоків, і стає головним у новій групі. Всі члени нової групи (включаючи головний) виконують код всередині паралельної конструкції. В кінці паралельної конструкції є неявний бар'єр. Після паралельної конструкції виконання призначеного для користувача коду продовжує тільки головний потік. В паралельний регіон можуть бути вкладені інші паралельні регіони, в яких кожен потік початкового регіону стає основним для своєї групи потоків. Вкладені регіони можуть в свою чергу включати регіони більш глибокого рівня вкладеності.»[**ВСТАВИТЬ ССЫЛКУ НА ОПЕНМП**]

OpenMP було обрано, як засіб управління паралельними обчисленнями, за відносну простоту використання та ефективний результат. Застосовуючи паралельні директиви з циклами for (рис 2.3.2.), можна досягти суттєвого приросту швидкості виконання коду, кратного кількості ядер у процесорі ЕОМ.

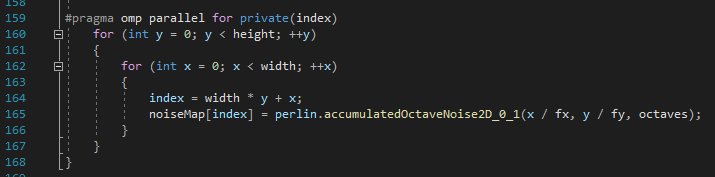


Рис 2.3.2. Приклад застосування OpenMP

siv::PerlinNoise – бібліотека що складається з одного файлу заголовка, що реалізує шум Перліна на сучасному C++. Базується на коді Improved Noise Кена Перліна [**ВСТАВИТЬ ССЫЛКУ НА ГИТХАБ ЛИБЫ**].

Бібліотека підтримує наступні можливості:

* псевдовипадкова генерація на основі зерна
* шум у 1, 2 та 3 вимірах
* застосування октав для генерації

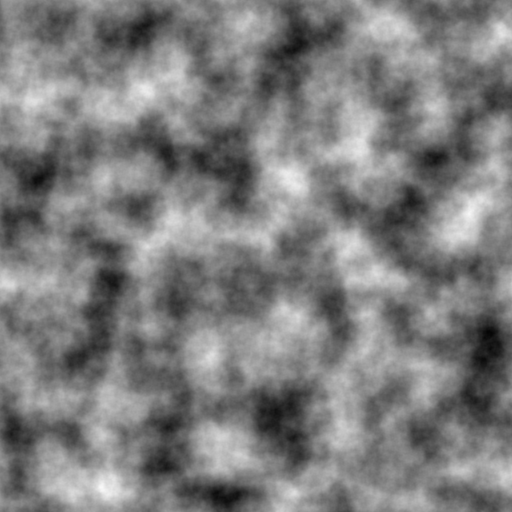


Рис 2.3.3. Приклад генерованого шуму з параметрами частоти(8.0), октав(8) та зерна(12345)

STL (Standard Template Library - Стандартна Бібліотека Шаблонів) - найбільша частина стандартної бібліотеки С++. Бібліотека STL містить п'ять основних видів компонентів:

* контейнер (container): керує набором об'єктів в пам'яті.
* итератор (iterator): забезпечує для алгоритму засіб доступу до вмісту контейнера.
* алгоритм (algorithm): визначає обчислювальну процедуру.
* функціональний об'єкт (function object): инкапсулирует функцію в об'єкті для використання іншими компонентами.
* адаптер (adaptor): адаптує компонент для забезпечення різного інтерфейсу.

Контейнерами є стандартні структури даних, такі як список (list), вектор (vector), словник (map) та багато інших. Формальні вимоги до контейнерів досить великі, але основним є правило доступу до елементів. Доступ до елементів контейнера здійснюється через спеціальні об'єкти – ітератори. Ви можете не знати, як розташовуються елементи контейнера в пам'яті, однак ви точно знаєте, що ітератори можна перебрати послідовно, і кожен з них надає доступ до елементу. Ітератор, який вказує на перший елемент, можна отримати за допомогою методу контейнера begin (). Ітератор, який вказує за останній елемент, можна отримати за допомогою методу контейнера end().

Для виконання дипломної роботи були використані контейнери vector та map, через їх зручність у ітеруванні кожного елементу та безпечне використання пам’яті.

**[ВСТАВИТЬ ССЫЛКУ НА STL]**

**2.4. Опис структури системи та алгоритмів її функціонування**

Основна механіка ігрового додатку з точки зору гравця: гравець, згенерувавши мапу, починає гру та виставляє свою Базу у довільному місці. У протилежній частині мапи створюється Вулик. Мета гравця – знищити Вулик та не допустити знищення своєї Бази. Для захисту та нападу йому потрібно створювати Турелі. Створення та заряджання Турелей потребує ресурсу – Енергії, що накопичується в залежності від того, скільки території захоплено експансією. Для цього потрібні споруди – Ядра, що допомагають розширювати територію експансії та будувати нові споруди. Ядра повинні бути підключені до Бази або іншого працюючого Ядра, щоб функціонувати.

Комахи також залежать від експансії, що розповсюджується за допомогою Пухлин. Чим більше території захоплено Пухлинами, тим більше накопичується Їжі, і тим більше Вулик може створити Комах, що відразу після створення прямують на розвідку або бій з юнітами гравця.

Створений ігровий додаток використовує поширену структуру роботи ігор (рис. 2.4.1.):

1. Ініціалізація ресурсів та вікна.
2. Основний цикл, що переривається лише тоді, коли користувач дає команду виходу:
3. Обробка вводу користувача – виконання дій в залежності від натиснутих клавіш та кнопок миші, таких як зміна положення камери, створення нових юнітів або закінчення гри. Оновлення положення курсору миші відносно екрану та ігрової мапи.
4. Cимуляція – обчислення положень юнітів, будівель, ігрових ресурсів, експансії.
5. Рендер – відображення усіх тайлів та юнітів, що входять до видимої зони.
6. Звільнення усіх ініціалізованих ресурсів та завершення виконання програми.

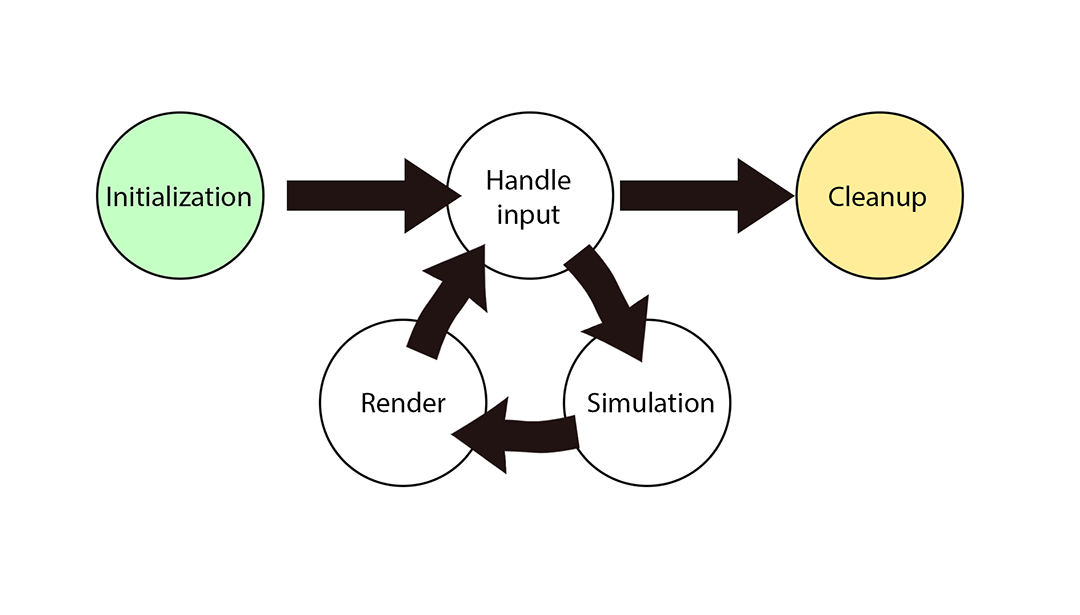


Рис.2.4.1. Цикл виконання

Спадкування та поліморфізм ООП дозволяє створювати класи з віртуальними методами, що мають бути перевизначені у класах-спадкоємцях. Так, у створеному ігровому додатку кожен клас-спадкоємець класу GameActor (рис. 2.4.2.) має методи Update() та Draw(), що викликаються у головних циклах симуляції та рендеру відповідно.

Назви та сигнатури методів однакові, однак для кожного класу ігрових юнітів ці методи перевизначені відповідно до їх функціоналу: виконуючи код Update(), ядра будують турелі та поширюють експансію, турелі шукають та наводяться на ворога, комахи зайняті розвідкою та пошуком турелей, вулик створює нових комах. А метод Draw() відображає юніта на ігровій мапі відповідно до його поточного стану (атакує, будує тощо), застосовуючи для цього відповідні текстури або графічні примітиви.

Окремо від інших класів є клас GameData. Цей клас містить у собі контейнери для зберігання усіх створених юнітів, матриці з даними про мапу у всіх її аспектах (тип місцевості, розповсюдження експансії, векторні поля для пошуку шляху тощо), супровідні функції-методи що застосовуються при обчисленні юнітів (пошук усіх юнітів в заданому радіусі, перевірка типу місцевості, розвіювання туману війни, вирахування вектора-контейнера з індексами усіх тайлів навколо, тощо) та головні цикли GameUpdate() та GameDraw(), які, крім ітеративного виклику відповідних методів юнітів симуляції та відображення, також симулюють процеси на мапі та відображують цю саму мапу.

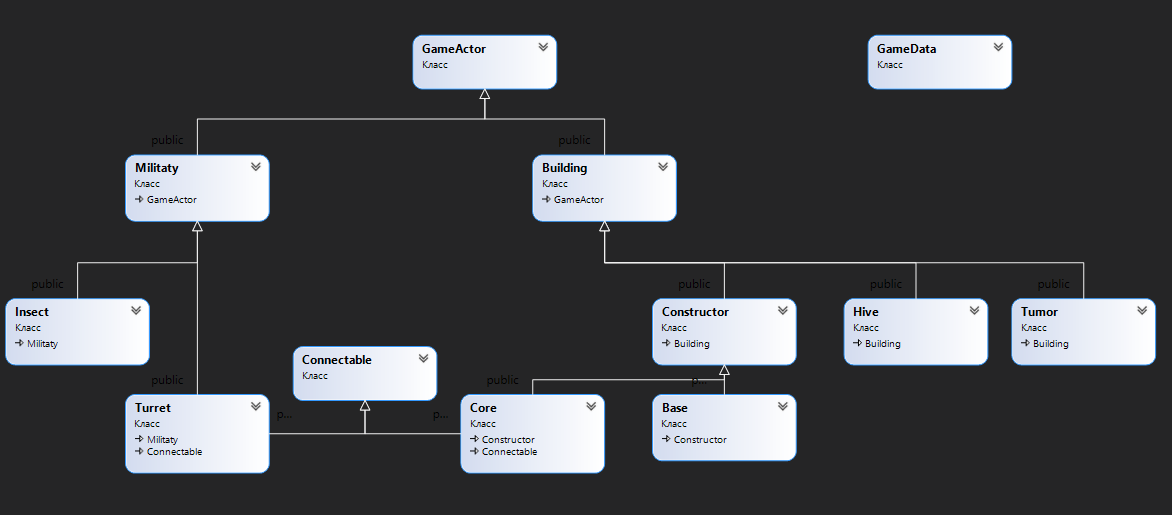


Рис. 2.4.2. Діаграма класів розробленої програми

Метод Update() оновлює показники юніта та стан відповідно до його поточного стану (рис. 2.4.3.). Наприклад, юніт «комаха» має поточний стан «Goes» («йде»), що зміниться на «атакує», якщо у радіусі виявлення буде знаходитися юніт супротивника. У стані «атакує» юніт «комаха» буде наближатися до виявленого ворога, поки той не наблизиться до радіуса атаки. Як тільки ворог буде знищений, стан «комахи» знову зміниться на стан «йде» з відповідною поведінкою.

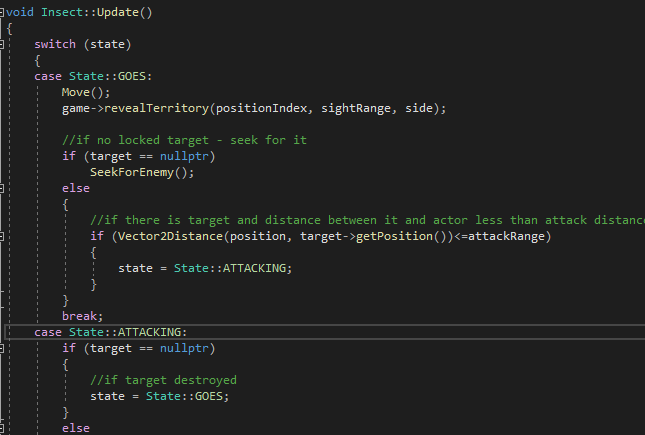


Рис. 2.4.3. Фрагмент коду поведінки юніта

Кожен юніт, чий клас є класом-нащадком класу Building («Будівля»), буде розповсюджувати навколо себе експансію (рис. 2.4.4-5). Від кількості тайлів (клітин ігрової мапи), захоплених експансією, залежить, скільки ресурсів за ігровий цикл отримує сторона, якій належить ця експансія.



Рис. 2.4.4. Новостворені Пухлини розповсюджують «слиз» – експансію Комах

Кожна будівля, щойно створюється, відмічає навколо себе область що буде покрита експансією та зберігає індекси цих тайлів. Це потрібно для повторного використання цих даних, без потреби в повторних розрахунках кожного циклу.

Кожен цикл будівлі перевіряють, чи є навколо тайл без експансії, що є сусідом до вже проведеної експансії. Якщо так – цей тайл відмічається експансією, якщо ні – будівля запам’ятовує, що вже перевірила всі тайли у радіусі експансії та більше не запускає цикл перевірки. Знищена або нефункціонуюча будівля не може підтримувати експансію навколо себе, тому ця експансія з часом зникне.

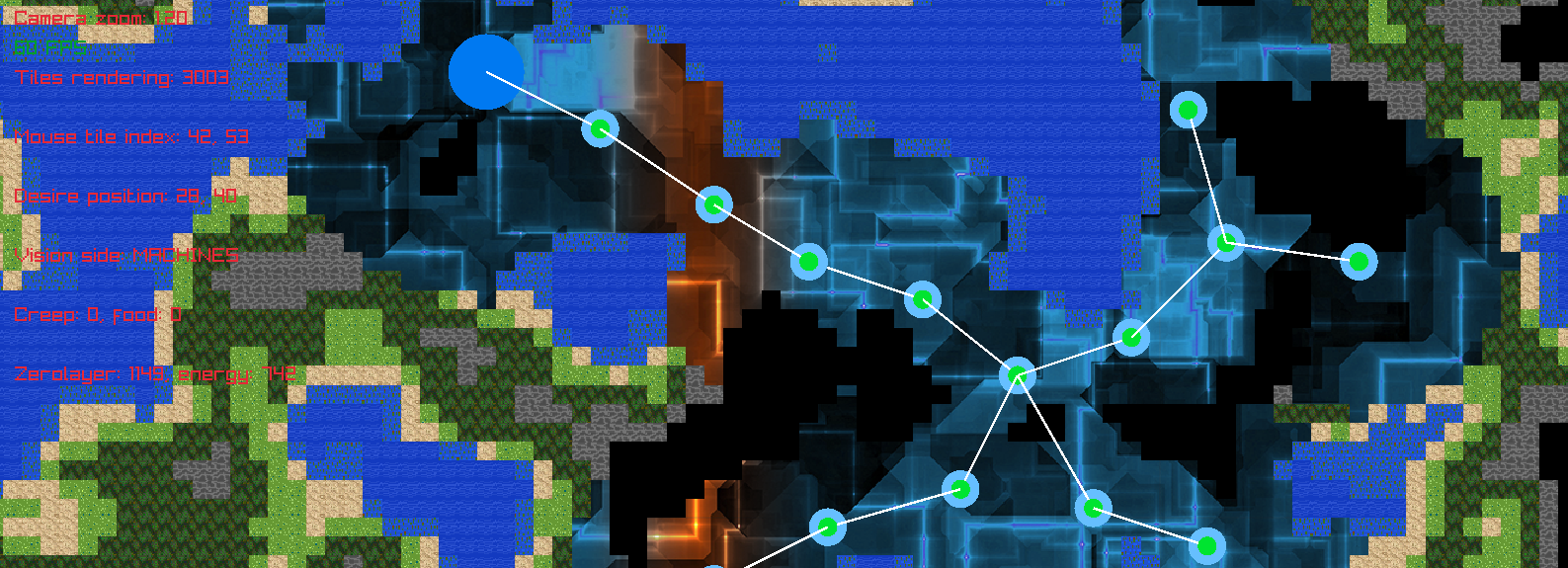


Рис. 2.4.5. База та Ядра гравця розповсюджує експансію Машин

Пошук шляху для комах базується на векторному полі, що генерується наступним чином:

1. Очистити векторне поле та теплову мапу від попередніх значень.
2. Виставити на координатах цільовій позиції значення теплової мапи 1.
3. Обчислити теплову мапу, перевіряючи кожен член масиву на перевірку:
4. Додати до масиву індексів на перевірку навколишні тайли, якщо їх координати не виходять за рамки мапи та вже не були перевірені.
5. Виставити значення теплової мапи, враховуючи вплив типу ландшафта на швидкість юніта та мапи отриманої та нанесеної шкоди. Якщо тайл знаходиться під туманом війни – виставити значення 1, незалежно від інших факторів.
6. Обчислити векторне поле: для кожного тайлу обчислити вектор, що буде спрямований на навколишній тайл з найменшим значенням теплової карти.

Коли векторне поле сформоване, юніту достатньо лише помножити свою швидкість на значення векторного поля у його поточній позиції, щоб отримати вектор руху у даний момент. Цей метод дозволяє за один цикл розрахунків отримати шлях для усіх юнітів з будь-якої позиції, що для ігор з великою кількістю юнітів набагато вигідніше, ніж розраховувати шлях для кожного з них окремо (як-то алгоритм .

Однак, також має місце очевидний недолік цього підходу: чим більше мапа, тим більше потрібно розрахунків, а постійно перераховувати векторне поле є витратним за процесорним часом. Хоча цю проблему можна обійти, розділяючи мапу на декілька зон й перераховуючи тільки ті, що потрібні для коректного пошуку шляху. **[ВСТАВИТЬ ССЫЛКУ НА ТУТОРИАЛ]**

Генерація ландшафту виконується наступним чином:

1. Звільняється пам’ять, якщо вона була зайнята попередньою версією ландшафту;

(ПРО ГЕНЕРАЦИЮ ЛАНДШАФТА ИЗ ШУМА – НА ПРЕДЫДУЩИЙ РАЗДЕЛ?)

**2.5. Обґрунтування та організація вхідних та вихідних даних програми**

**2.6. Опис розробленої системи**

**2.6.1. Використані технічні засоби**

При розробці та тестуванні системи була використана персональна ЕОМ з наступними характеристиками:

* Процесор AMD FX-4100 Quad-Core;
* Монітор Asus VE208N;
* Оперативна пам’ять 16 GB RAM;
* Накопичувач SSD 240GB Kingston SUV400S37240G;
* Клавіатура, миша;

**2.6.2. Використані програмні засоби**

Для розробки та компіляції вихідних кодів використовувалась IDE Microsoft Visual Studio 2019 Community – безкоштовне середовище розробки з підтримкою великої кількості мов програмування та технологій.

**2.6.3. Виклик та завантаження програми**

Для запуску програми треба виконати файл GameProject.exe.

**2.6.4. Опис інтерфейсу користувача**

Після запуску гри натисніть «Generate terrain» для запуску генератора ландшафту. У його правій частині доступні для зміни параметри генерації:

* розміри ландшафту в клітинах (тайла);
* зерно псевдослучайной генерації;
* кількість октав і частота
* межі висот різних типів ландшафту

Кнопки "Rerender" і "Regenerate" потрібні для застосування змін типу ландашфта і повторного створення карти висот.

Зліва і по центру відображені підсумковий ландшафт і чорно-біла карта висот відповідно.

Для застосування отриманого ландшафту в грі натисніть клавішу «Esc» або кнопку «Back» зліва вгорі, потім в меню натисніть «Play».

Спочатку екран в грі чорний - юніти не розвідали територію і вона покрита «Туманом війни».

Управління камерою: клавіші "WASD", коліщатко мишки для віддалення / наближення.

Зліва відображений параметр наближення ( "Camera zoom:"), якщо він менше 1.0, то карта відображається в спрощеному варіанті.

Для створення турелі натисніть «1», «2» або «3» (легка, важка та ПВО відповідно), наведіть курсор в бажану точку та натисніть лівою кнопкою миші.

Для створення ядра натисніть «Q» та натисніть на бажаній клітці. Ядра можуть будувати інші юніти та розповсюджують «заряджену» області.

Щоб відобразити векторне поле, по якому орієнтуються юніти, натисніть «F5».

Щоб видалити всі юніти з карти, натисніть "C".

Щоб змінити вид на «Туман війни» з боку Машин на Комах та навпаки, натиснять «F1».

Щоб згенерувати нову карту, натисніть «Esc» і задійте генератор ландшафту.

**РОЗДІЛ 3**

**ЕКОНОМІЧНИЙ РОЗДІЛ**

Вихідні дані:

1. передбачуване число операторів програми – 2379;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста – 154 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 13 грн/год.

**3.1. Розрахунок вартості розробки програмного продукту**

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

, людино-годин, (3.1)

де *to* – витрати праці на підготовку й опис поставленої задачі (приймається 50);

*tи* – витрати праці на дослідження алгоритму рішення задачі;

*tа* – витрати праці на розробку блок-схеми алгоритму;

*tп* – витрати праці на програмування по готовій блок-схемі;

*tотл* – витрати праці на налагодження програми на ЕОМ;

*tд* – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

, де (3.2)

*q* – передбачуване число операторів;

*C* – коефіцієнт складності програми;

*p* – коефіцієнт кореляції програми в ході її розробки.

;

Витрати праці на вивчення опису задачі *tи* визначається з урахуванням уточнення опису і кваліфікації програміста:

, людино-годин, (3.3)

де *B* – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

*K* – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

, людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

; (3.4)

*,* людино-годин.

Витрати на складання програми по готовій блок-схемі:

; (3.5)

*,* людино-годин.

Витрати праці на налагодження програми на ЕОМ:

* за умови автономного налагодження одного завдання:

; (3.6)

, людино-годин,

* за умови комплексного налагодження завдання:

; (3.7)

, людино-годин

Витрати праці на підготовку документації:

; (3.8)

де *tдр* – трудомісткість підготовки матеріалів і рукопису

; (3.9)

людино-годин.

*tдо*– трудомісткість редагування, печатки й оформлення документації

; (3.10)

, людино-годин.

, людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

t = 38,2 + + + + + = 1703,5, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1703,5 людино-годин для розробки даного програмного забезпечення.

**3.2. Розрахунок витрат на створення програми**

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

, грн, (3.11)

де *ЗЗП* – заробітна плата виконавців, яка визначається за формулою:

, грн, (3.12)

де *t* – загальна трудомісткість, людино-годин;

*СПР* – середня годинна заробітна плата програміста, грн/година

*,* грн.

*ЗМВ* – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

, грн, (3.13)

де *tотл* – трудомісткість налагодження програми на ЕОМ, год.

*СМЧ* – вартість машино-години ЕОМ, грн/год.

грн.

грн.

Очікуваний період створення ПЗ:

, мес. (3.14)

де Bk- число виконавців;

Fp – місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

міс.

**Висновки**. На розробку даного програмного забезпечення піде людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 9,67 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть грн.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

**(ДОБАВИТЬ ССЫЛКИ ДЛЯ ПЕРВОГО РАЗДЕЛА ИЗ ВИКИ)**

Шум Перлина URL: <https://habr.com/ru/post/342906/>

Bjarne Stroustrup. The C++ Programming Language, 4th Edition. Addison–Wesley, 2013, ‎ 1376 с.

Speed comparison of programming languages. URL: <https://github.com/niklas-heer/speed-comparison>

raylib architecture. URL: <https://github.com/raysan5/raylib/wiki/raylib-architecture>

raylib bindings. URL: <https://github.com/raysan5/raylib/blob/master/BINDINGS.md>

Параллельные заметки №1 – технология OpenMP. URL: <https://habr.com/ru/company/intel/blog/82486/>

siv::PerlinNoise. URL: <https://github.com/Reputeless/PerlinNoise>

Основные понятия стандартной библиотеки. С++ URL: <https://habr.com/ru/post/434986/>

Understanding Goal-Based Vector Field Pathfinding. URL: <https://gamedevelopment.tutsplus.com/tutorials/understanding-goal-based-vector-field-pathfinding--gamedev-9007>