**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) |
| **спеціальності** | *122 Комп’ютерні науки* |
|  | (код і назва спеціальності) |
| **освітньої програми** | *Комп’ютерні науки* |
|  | (назва освітньої програми) |
| **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 | *доц. Бердник М.Г.* |  |  |  |
| **розділів:** |  |  |  |  |
| спеціальний | *доц. Бердник М.Г.* |  |  |  |
| економічний | *доц. Касьяненко Л.В.* |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Рецензент** |  |  |  |  |
| **Нормоконтролер** | *доц. 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) | | | |
| « » |  |  | | | 2021 | року |

**ЗАВДАННЯ**

**на кваліфікаційну роботу**

|  |
| --- |
| *бакалавра* |

(назва освітньо-кваліфікаційного рівня)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **студента** | *122-17-3* |  | *Бондаря Андрія Павловича* | | | | | | |
| (група) | | | | (прізвище та ініціали) | | | | |
| **тема кваліфікаційної роботи** | | | | | *Розробка ігрового додатку для адаптації* | | | | | | |
| *дій гравця з використанням бібліотек RayLib v3.5, OpenMP 2.0 на мові C++* | | | | | | | | | | | |
|  | | | | | | | | | | | |
|  | | | | | | | | | | | |
| затверджена наказом ректора НТУ «ДП» від | | | | | | *07.06.2021р.* | № | *317-с* | | |

|  |  |  |
| --- | --- | --- |
| **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*** | ***13.05.2021 р.*** |
| ***Економічний*** | ***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки*** | ***27.05.2021 р.*** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Завдання видав | |  | | | |  | | *доц. Бердник М.Г.* |
|  | (підпис) | | | |  | | (посада, прізвище, ініціали) | |
| Завдання прийняв до виконання | | |  |  | | | *Бондар А.П.* | |
|  | | | (підпис) | |  | | (прізвище, ініціали) | |

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

**РЕФЕРАТ**

Пояснювальна записка: 47 с., 23 рис., 0 табл., 3 дод., 13 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, ПЕРЦЕПТРОН, АЛГОРИТМ, ШТУЧНИЙ ІНТЕЛЕКТ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ.

**ABSTRACT**

Explanatory note: 47 p., 23 figs, 0 tabl., 3 appx., 13 sources.

Object of development: strategy game application with the use of machine learning.

The purpose of the diploma project: to create a game application with artificial intelligence that can adapt to any game situation and help develop the strategic thinking of the player.

The introduction considers the analysis and current state of the problem, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic and clarifies the problem.

The first section analyzes the subject area, determines the relevance of the task and the purpose of development, formulates the task, specifies the requirements for software implementation, technology and software.

The second section analyzes the available solutions, selected platforms for development, designed and developed the program, describes the program, algorithm and structure of its operation, as well as calling and loading the program, determines the input and output data, describes the parameters of hardware.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

Of practical importance is the creation of a software application that provides an opportunity to improve strategic and tactical skills.

The relevance of the information system is determined by the increased trend of development using machine learning, which optimizes actions for adaptation to arbitrary situations, demonstrates the capabilities and flexibility of algorithms.

Keyword list: GAME APPLICATION, COMPUTER, MACHINE LEARNING, PERCEPTRON, ALGORITHM, ARTIFICIAL INTELLIGENCE, PROCEDURAL GENERATION.

**ЗМІСТ**

РЕФЕРАТ……………………………………………………………………………2

ABSTRACT………………………………………………………………………….3

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ……………………………………………...6

ВСТУП……………………………………………………………………………….7

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ 9

* 1. Загальні відомості з предметної галузі…………………………………….9
  2. Призначення розробки та галузь застосування…………………………..15
  3. Підстава для розробки……………………………………………....…......16
  4. Постановка завдання………………………………………………………16
  5. Вимоги до програми або програмного виробу…………………………..17
     1. Вимоги до функціональних характеристик………………………….…….17
     2. Вимоги до інформаційної безпеки………………………………………..17
     3. Вимоги до складу та параметрів технічних засобів………………....…..18
     4. Вимоги до інформаційної та програмної сумісності ……………....…….18

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ.19

* 1. Функціональне призначення системи……………………………………..19
  2. Опис застосованих математичних методів………………………………..19
  3. Опис використаних технологій та мов програмування…………………..22
  4. Опис структури системи та алгоритмів її функціонування ……………..29
  5. Обґрунтування та організація вхідних та вихідних даних програми……36
  6. Опис розробленої системи .............………………………………………37
     1. Використані технічні засоби…….………………………………………..37
     2. Використані програмні засоби……………………………………………37
     3. Виклик та завантаження програми……………………………………….37
     4. Опис інтерфейсу користувача…………………………………………….37

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ…………………………………………….39

* 1. Розрахунок трудомісткості та вартості розробки програмного продукту....39
  2. Рахунок витрат на створення програми……………………………………...42

ВИСНОВКИ………………………………………………………………………..45

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ………………………………………....46

Додаток А. Код програми……………………………….………………………..48

Додаток Б. Відгук керівника економічного розділу……………………………125

Додаток В. Перелік файлів на диску…………………………………………….126

**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

**ВСТУП**

На сьогодні технології машинного навчання набувають все більшого застосування. Створюються та покращуються нові типи штучних нейронних мереж, а це у наш час повсюдної автоматизації означає лише подальше прискорення темпів автоматизації навіть у тих областях, що раніше вважалися творчими та непридатними для застосування у них машин.

Гнучкі алгоритми, що не скуті жорстко продиктованими даними або ж навіть імітують будову нейронів живих істот, були створені ще у минулому столітті. Ці алгоритми, нехай навіть в силу своєї визначеності та природи як послідовності чітких інструкцій, здатні адаптуватися під, майже, будь-які ситуації та виконувати найрізноманітніші ситуації.

Штучні нейронні мережі, що створені задля автоматизації творчої роботи людини, навчаються на, відповідно, готових результатах людської праці. І це в них прекрасно виходить: машина вже зараз здатна: здолати професійного у логічну гру «Го», замінити обличчя людини у відео на інше, генерувати правдоподібні тексти та неймовірні картини, поєднувати декілька зображень в одне, діагностувати рак і так далі і тому подібне. Нехай це лише тільки імітація людської творчості – але імітація гідна. Зараз машини не розумні, але при належному навчанні здатні автоматизувати неймовірну кількість людської праці.

Метою даного проекту є створення адаптивного ігрового додатка для демонстрації можливостей методів машинного навчання та розвитку стратегічного мислення гравця.

Для досягнення цієї мети потрібно:

* аналізувати можливі варіанти реалізації машинного навчання;
* розробити базову ігрову механіку;
* поєднати ігрову механіку, баланс юнітів та машинне навчання.

На сьогодні поширюється тенденція використання машинного навчання у автоматизованих системах та додатках, тому дане програмне забезпечення є актуальним.

Завдання даної кваліфікаційної роботи та об'єкт його діяльності безпосередньо пов'язані з напрямом підготовки «Комп’ютерні науки» та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки» галузі знань 12 «Інформаційні технології». Виконання кваліфікаційної роботи надає можливості отримання автору кваліфікації «фахівець в галузі інформаційних технологій».

**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» (рис. 1.1.1.), 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. 1.1.1. 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 (рис.1.1.2.). Забруднення, яке розповсюджується фабрикою, турбує мешканців планети – вигаданих членистоногих істот, та змушує збиратися в групи і нападати на джерело забруднення.



Рис. 1.1.2. 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.1.1.3.).



Рис. 1.1.3. Ігровий процес StarCraft 2

RTS зазвичай відтворюють військові дії та розбудову військових баз для здобуття перемоги над противником, мають специфічний інтерфейс (рис. 1.1.4.). 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. 1.1.4. 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 [1]».

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 (рис.1.1.5), 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. 1.1.5. 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.**

Відповідно до ОКХ та ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою та затверджується наказом ректора.

Отже підставами для розробки (виконання кваліфікаційної роботи) є:

* ОКХ та ОПП за напрямом підготовки 6.050101 «Комп’ютерні науки»;
* навчальний план та графік навчального процесу;
* наказ ректора Національного технічного університету «Дніпровська політехніка» №317 -с від 07.06.2021 року;
* завдання на дипломний проект на тему: «Розробка ігрового додатку для адаптації дій гравця з використанням бібліотек RayLib v3.5, OpenMP 2.0 на мові C++».

**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: 4096 МБ.
* VRAM (відеопам’ять): 128 МБ;
* Вільне місце на накопичувачі: 256 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 ігрових додатків.
* 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**

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).

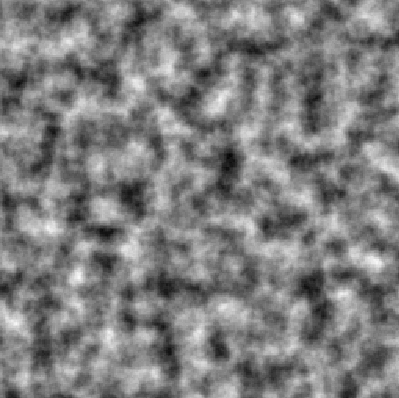


Рис. 2.2.1 Приклад згенерованої чорно-білої текстури карти висот

Принцип генерації шума Перліна:

1. Для кожного цілого числа вибирається псевдовипадкове дійсне з діапазону [0, 1]. «Це буде нахил (тангенс кута нахилу) прямої, що проходить через дану точку на числовій осі [2]» (рис 2.2.2). Параметр частоти впливає на те, скільки цих чисел буде на обраному проміжку, параметр зерна – на те, які саме псевдовипадкові значення будуть отримані.

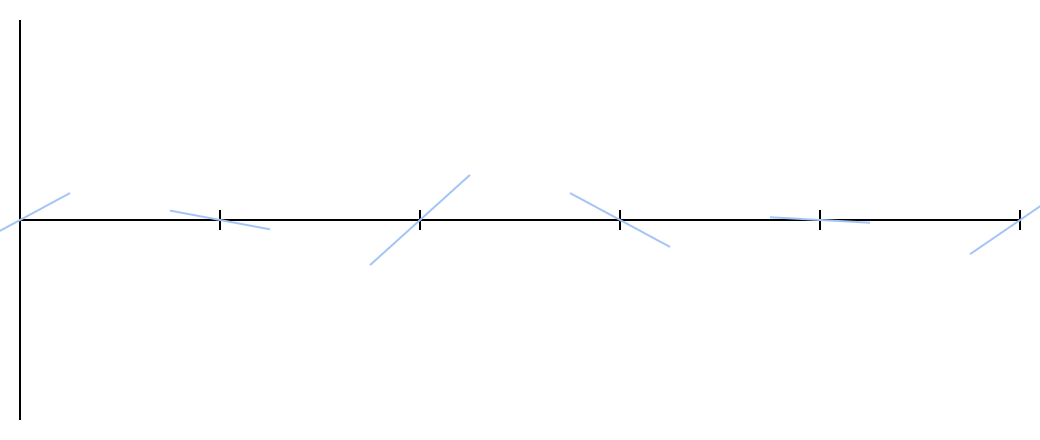
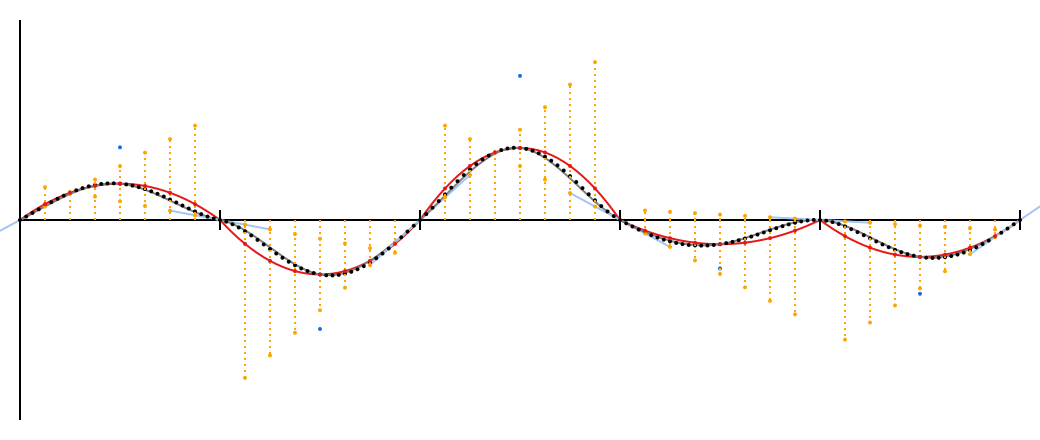


Рис. 2.2.2. Початкові точки генерації

1. Значення між кожною точкою підлягають лінійній інтерполяції (рис. 2.2.3.).

  
Рис. 2.2.3. Криві, отримані після лінійної інтерполяції

1. Отримані значення підлягають дії октав: створюється окремі значення з подвоєною частотою, діляться на 2 та додаються до оригінальних значень. Цей процес може повторюватися декілька раз, кожен з яких використовує частоту в 2 рази більше за попередній. У результаті значення стають більш «шорсткими» (рис 2.2.4.).

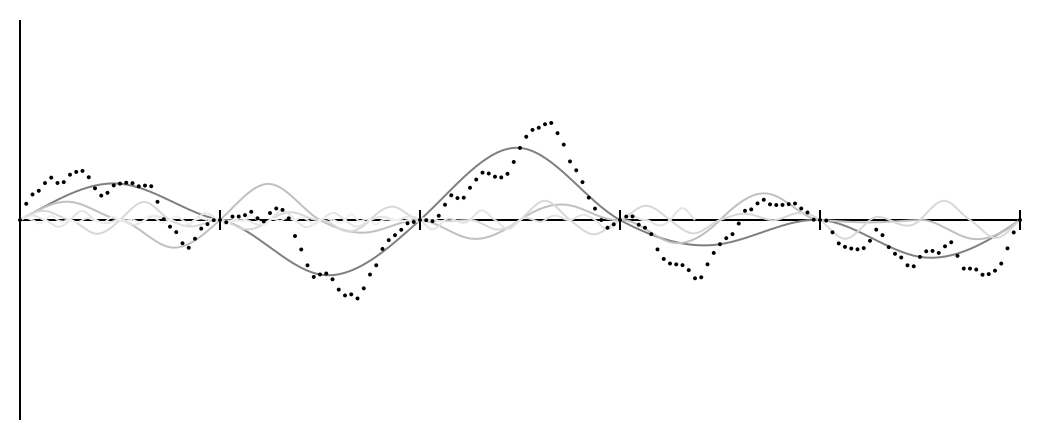


Рис. 2.2.4. Криві, отримані після додавання октав (результат позначений пунктиром)

Ці принципи генерації поширюються на більше число вимірів, ніж 1 (рис. 2.2.5.).

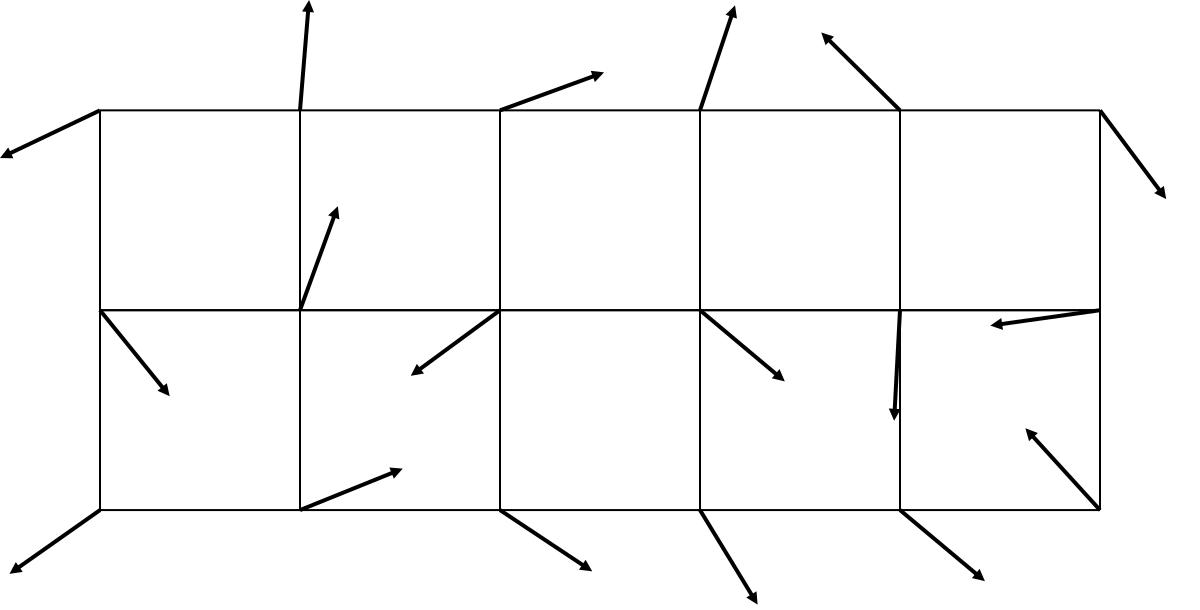


Рис. 2.2.5. Двовимірна генерація потребує створення векторів кутів нахилу

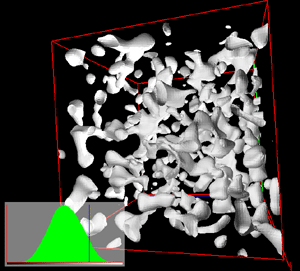


Рис. 2.2.6. Тривимірний варіант шуму Перліна

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 перцептрона, 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 перцептрон 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:

1. C++
2. RayLib 3.5
3. siv::PerlinNoise
4. 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 [3]».

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.

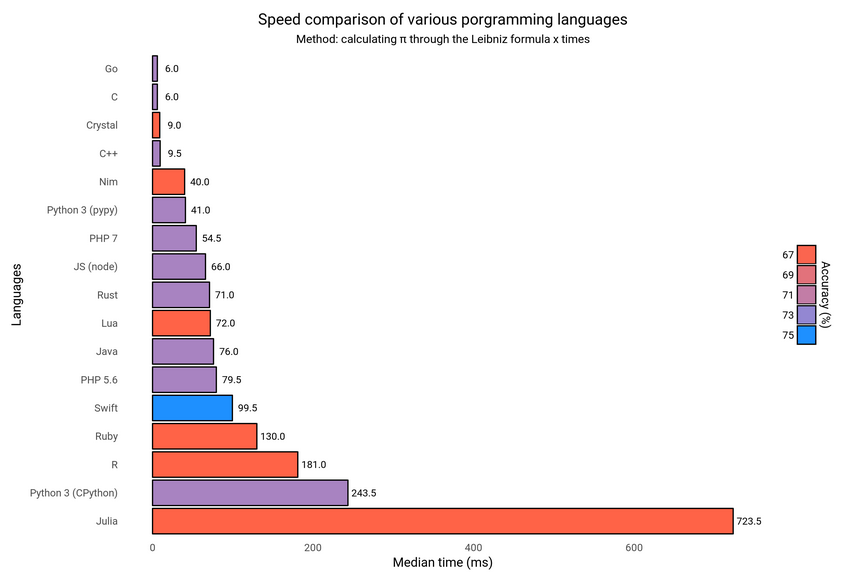


Рис. 2.3.1. Графік порівняння часу виконання формули Лейбніца на різних мовах програмування [4]

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.

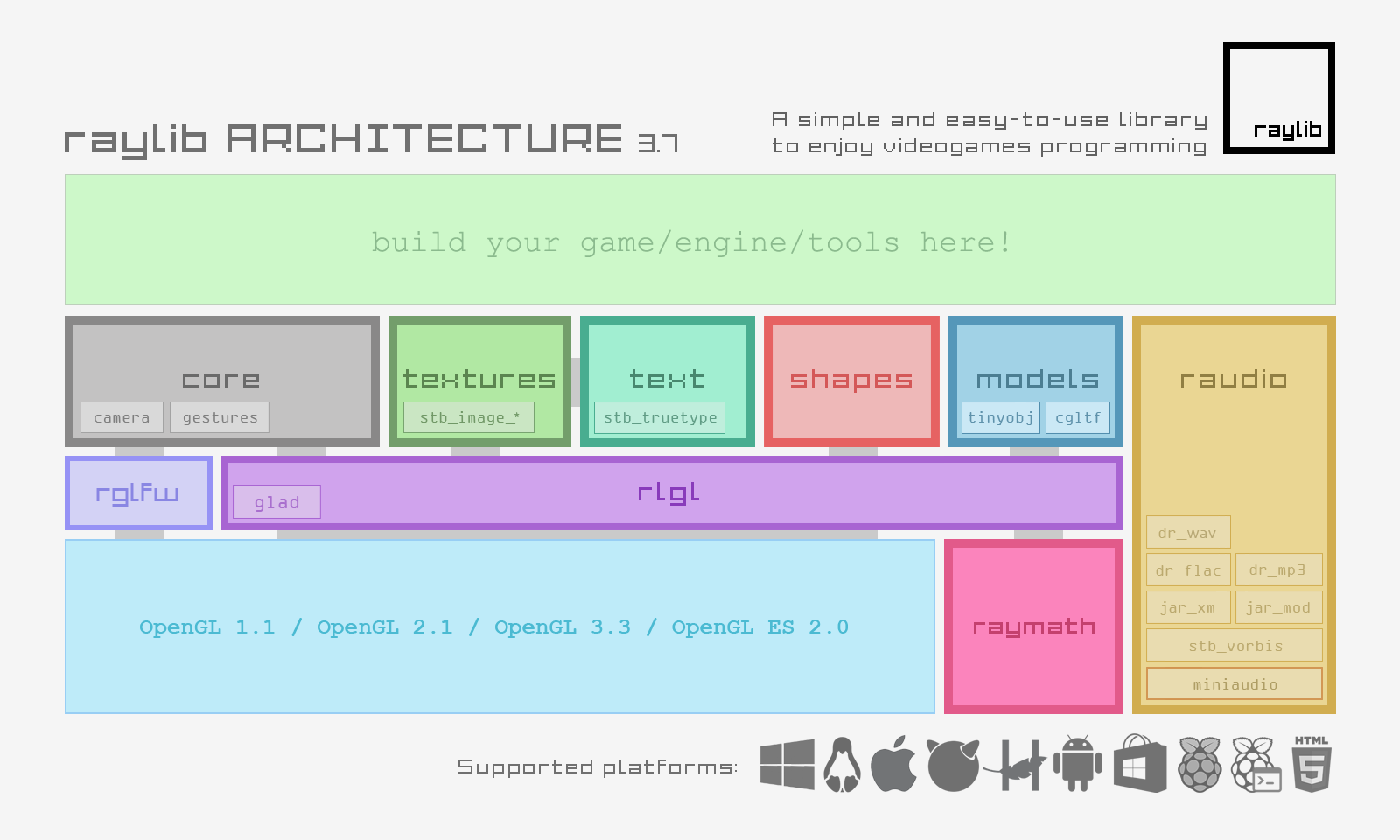


Рис 2.3.1. Архітектура бібліотеки RayLib

RayLib складається з семи основних модулів:

1. core, управління вікном, графічним контекстом OpenGL та введенням команд;
2. rlgl, обгортка-API над OpenGL;
3. shapes, функції візуалізація двомірних примітивів;
4. textures, завантаження та управління текстурами та зображеннями;
5. text, завантаження шрифтів та візуалізація тексту;
6. models, завантаження та візуалізація тривимірних моделей;
7. 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 [5]».

Крім основних модулів, бібліотека містить також допоміжні модулі:

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

«RayLib доступний для використання на великій кількості мов програмування, у тому числі: C, C++, C#, D, Go, Rust, Lua, Python, Java та інші [6]».

«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) [7]».

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.

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.

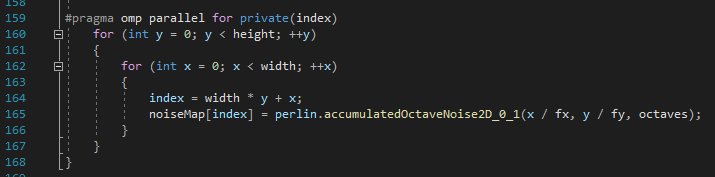


Рис 2.3.2. Приклад застосування OpenMP

«siv::PerlinNoise – бібліотека що складається з одного файлу заголовка, що реалізує шум Перліна на сучасному C++. Базується на коді Improved Noise Кена Перліна [8]».

Бібліотека підтримує наступні можливості:

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

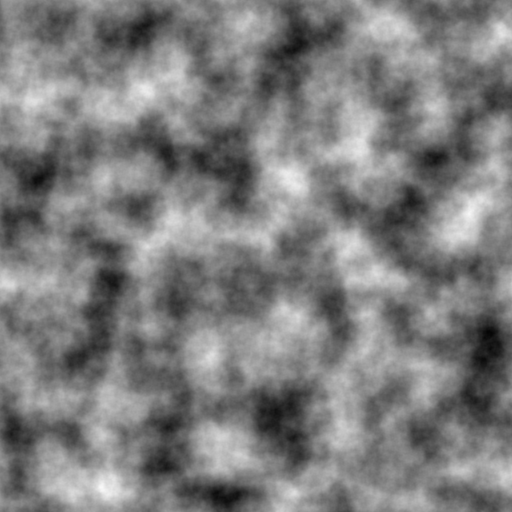


Рис 2.3.3. Приклад генерованого шуму з параметрами частоти(8.0), октав(8) та зерна(12345)

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. Програміст може не знати, як розташовуються елементи контейнера в пам'яті, однак точно відомо, 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() [9]».

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

**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.

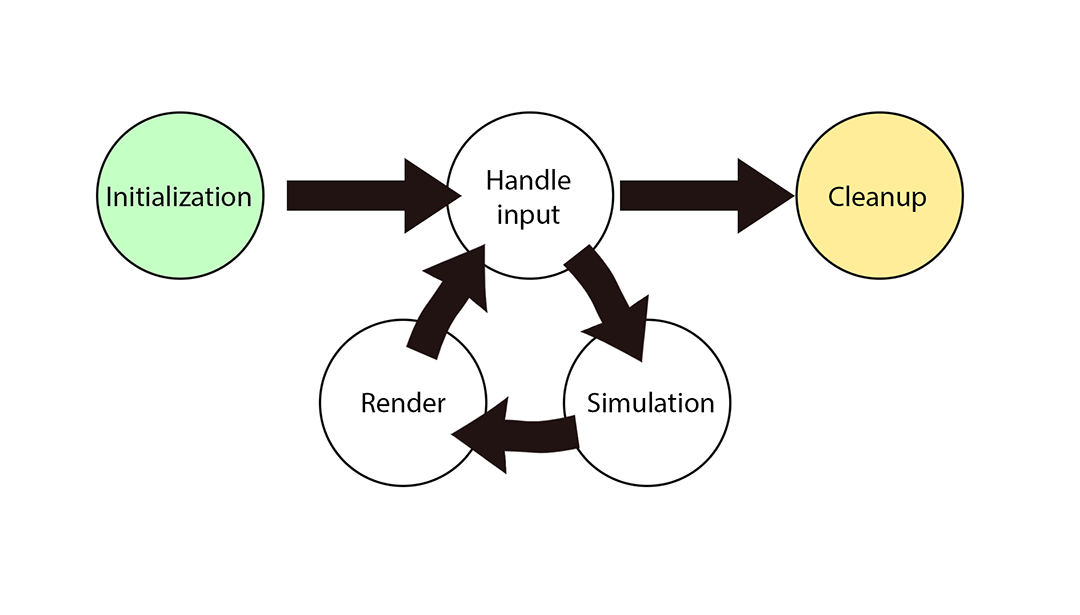


Рис.2.4.1. Цикл виконання

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.

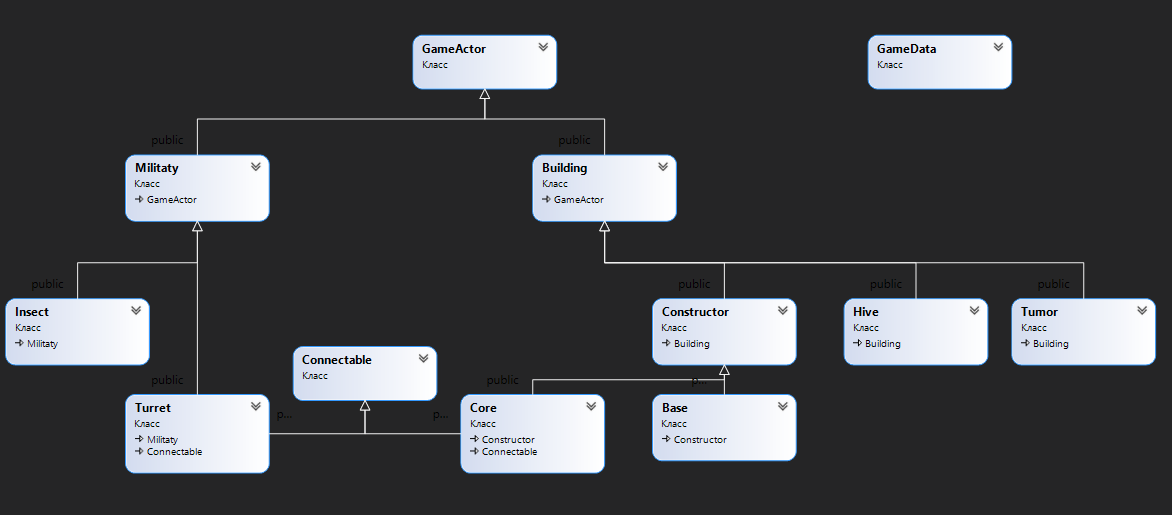


Рис. 2.4.2. Діаграма класів розробленої програми

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.

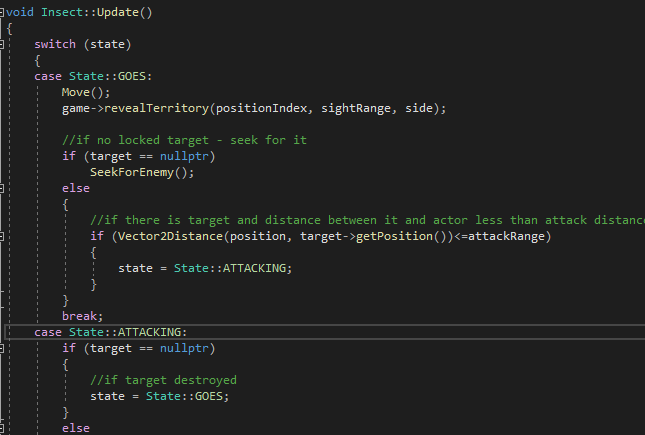


Рис. 2.4.3. Фрагмент коду поведінки юніта

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 [11]».

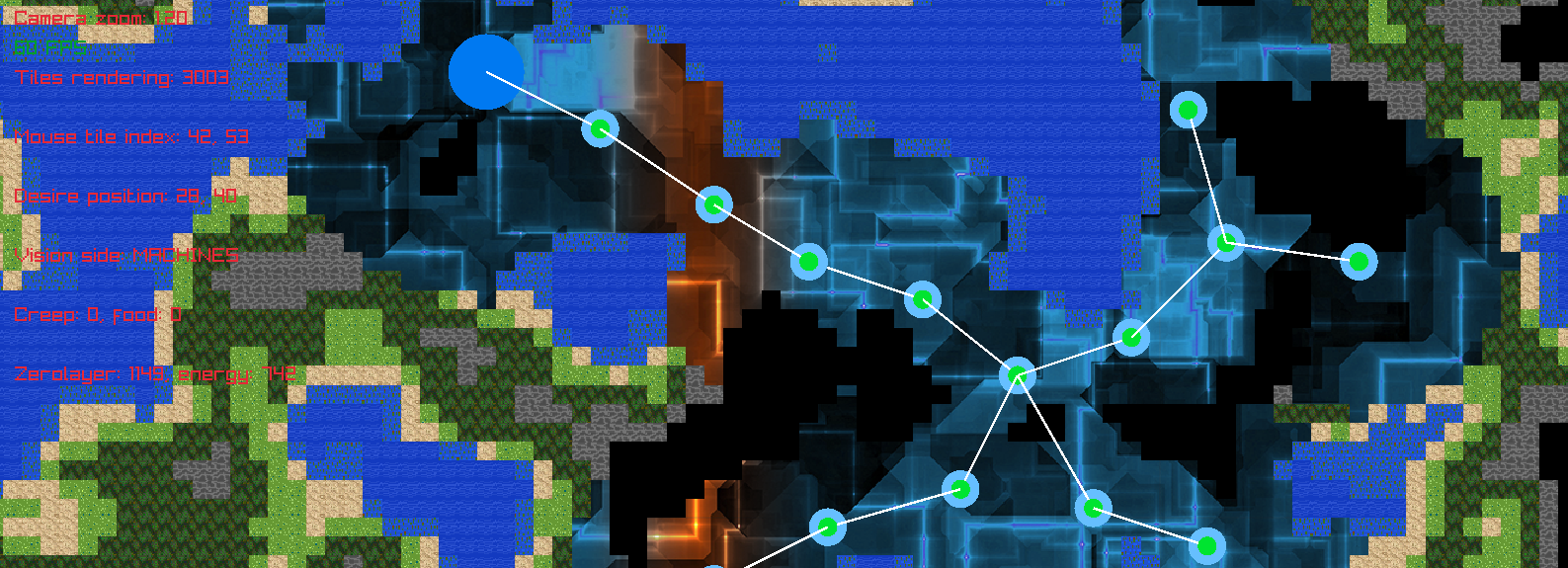


Рис. 2.4.5. База та Ядра гравця розповсюджує експансію Машин

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 теплової мапи значення 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 [10]».

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.

Генерація ландшафту виконується наступним чином:

1. Звільняється пам’ять, якщо вона була зайнята попередньою версією ландшафту, та виділяється нова пам’ять відповідно до заданого розміру мапи;
2. Генерується масив значень шуму Перліна, відповідно до заданих користувачем параметрів: кількості октав, частоти та зерна. Отримані значення лежать в проміжку від 0.0 до 1.0.;
3. Генерується чорно-біле зображення шуму: яскравість кожного пікселя дорівнює значенню шуму відповідної точки, помноженого на 255.

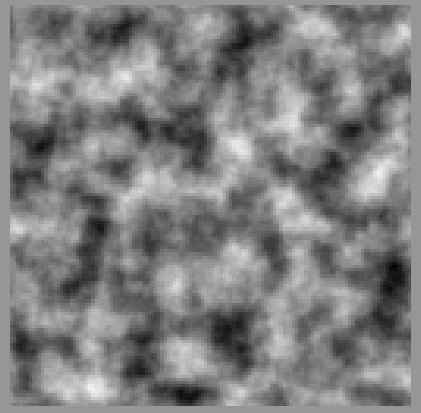


Рис. 2.4.6. чорно-біла мапа шуму Перліна

1. Генерується масив типу ландшафта: для кожної точки вираховується тип місцевості, враховуючи найперший, що відповідає значенню шуму у цій точці (рис. 2.4.6.).

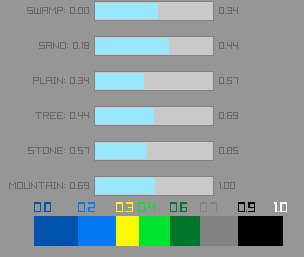


Рис. 2.4.7. Інтерфейс генератора ландшафту, повзунки що задають рівні меж типів місцевості

1. Генерується кольорове зображення місцевості: колір кожного пікселя визначається відповідним значенням масиву типу місцевості та заздалегідь заданій палітрі кольорів, де кожен колір відповідає своєму типу місцевості. (рис. 2.4.7.).

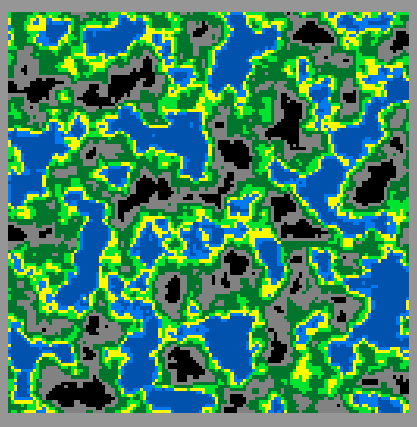


Рис. 2.4.8. Кольорова мапа ландшафту

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

Бібліотека RayLib зберігає попередній стан усіх клавіш. Коли користувач натискає будь-яку клавішу, цей факт зберігається у поточному стані кнопки та згодом переноситься у попередній стан. Це дозволяє програмісту перевіряти, чи була натиснута потрібна клавіша, та як саме вона була натиснута (уточнюючи попередній стан натискання), що дає можливість перевіряти лише те, що потрібно в даний момент для обчислень.

Положення миші також видається на вимогу програміста, що відрізняється від того, з чим працює модель подій (Event model) тим, що кожну обробку потрібно контролювати вручну, що є досить гнучким підходом.

**2.6. Опис розробленої системи**

**2.6.1. Використані технічні засоби**

При тестуванні системи була використана персональна ЕОМ-ноутбук з наступними характеристиками:

* Процесор Intel Celeron N3350;
* Монітор 11 дюймів;
* Відеоадаптер Intel HD Graphics;
* Оперативна пам’ять 4 GB RAM;
* Накопичувач HDD 500 GB;
* 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.

Для резервного зберігання та контролю версій використовувались веб-сервіс GitHub та система контролю версій Git.

**2.6.3. Виклик та завантаження програми**

Для запуску програми треба запустити виконуваний 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» зі сторони Машин на Комах та навпаки, натисніть «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; [12]
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 – 14 грн/год. [13]

**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.

**ВИСНОВКИ**

В даній кваліфікаційній роботі був розроблений ігровий додаток, призначений для демонстрації можливостей машинного навчання та розвитку стратегічного мислення гравця.

Розробка була проведена із застосуванням мови програмування C++, бібліотек RayLib (полегшення створення ігрових додатків), STL (стандартні шаблони контейнерів), OpenMP (розподілені обчислення) та siv::PerlinNoise (реалізація шуму Перліна). Обрана мова програмування, бібліотеки та оптимізація обчислень забезпечили ігровому додатку високу продуктивність роботи.

Під час виконання даного дипломного проекту були виконані наступні задачі:

* Розроблена ігрова механіка стратегії у реальному часі;
* Досліджені найближчі аналоги ігрових додатків та методи геймдизайну;
* Проведено збалансування параметрів ігрових юнітів;
* Поєднано ігрову механіку зі машинним навчанням.

Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту (1703,5 людино-годин), проведений підрахунок вартості роботи по створенню програми (269915,8 грн) та розраховано час на його створення (9,67 місяців).

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

* + - 1. Обзор техник реализации игрового ИИ. URL: <https://habr.com/ru/post/420219/>
      2. Perlin noise. URL: <https://eev.ee/blog/2016/05/29/perlin-noise/>
      3. Bjarne Stroustrup. The C++ Programming Language, 4th Edition. Addison–Wesley, 2013, ‎ 1376 с.
      4. Speed comparison of programming languages. URL: <https://github.com/niklas-heer/speed-comparison>
      5. raylib architecture. URL: <https://github.com/raysan5/raylib/wiki/raylib-architecture>
      6. raylib library bindings to other languages. URL: <https://github.com/raysan5/raylib/blob/master/BINDINGS.md>
      7. Параллельные заметки №1 – технология OpenMP. URL: <https://habr.com/ru/company/intel/blog/82486/>
      8. siv::PerlinNoise. URL: <https://github.com/Reputeless/PerlinNoise>
      9. Основные понятия стандартной библиотеки. С++ URL: <https://habr.com/ru/post/434986/>
      10. Understanding Goal-Based Vector Field Pathfinding. URL: <https://gamedevelopment.tutsplus.com/tutorials/understanding-goal-based-vector-field-pathfinding--gamedev-9007>
      11. Brood War API – The Comprehensive Guide: Creating buildings, and everything about the creep <https://makingcomputerdothings.com/brood-war-api-the-comprehensive-guide-creating-buildings-and-everything-about-the-creep/>
      12. Зарплаты программистов в Украине. ДОУ. URL: <https://jobs.dou.ua/salaries/#period=dec2020&city=Dnipro&title=Junior%20Software%20Engineer&language=C%2B%2B&spec=&exp1=0&exp2=10>.
      13. Расчет стоимости машино-часа ЭВМ. studopedia.ru. URL: <https://studopedia.ru/4_74479_raschet-stoimosti-mashinochasa-ekspluatatsii-evm.html>.

**ДОДАТОК А**

**КОД ПРОГРАМИ**

**TerrainGenerator.h**

#pragma once

#include "PerlinNoise.h"

#include "GameClasses.h"

#include "Enums.h"

#include "raylib.h"

#include "raygui.h"

class TerrainGenerator

{

private:

bool rendered;

int width, height, octaves;

int seed;

double frequency, fx, fy;

const char\*\* levelNames;

Color\* palette;

float\* levels;

double\* noiseMap;

TerrainType\* terrainMap = nullptr;

Texture2D colorPreview;

Texture2D grayPreview;

bool valueChanging[3];

public:

bool closed;

bool isGenerated();

TerrainGenerator(void);

void RegenerateTerrain(); // создать новую карту высот шумом Перлина

void RerenderTerrain(); // вызывается при смене значения ползунков (проверить насколько быстро это происходит, распараллелить?), вызывать только если кнопка мыши отжата

void DrawInterface();

~TerrainGenerator();

Terrain getMap();

};

**GameClasses.h**

#pragma once

#include "Enums.h"

//#include <memory>

class GameActor;

class Building;

class Constructor;

class Connectable;

class Militaty;

class Turret;

class Base;

class GameData

{

private:

//TODO: make this array two-dimensional

TerrainType\* mapTerrain = nullptr;

Image\* tileset;

Texture2D tilesetTex[8];

Texture2D terrainTexture;

Texture2D expansionInsectsTexture;

Animation expansionMachinesAnimation;

int currentFrame = 0;

bool gamePaused = false;

std::map<ActorType, std::map<State, Animation>> unitAnimations;

Color\* palette;

std::vector<GameActor\*> unitsList;

std::vector<Building\*> expansionUnitsList\_Insects;

std::vector<Building\*> expansionUnitsList\_Machines;

std::vector<Constructor\*> constructorList;

std::vector<Militaty\*> militaryUnitsList;

std::vector<Turret\*> turretUnitsList;

GameActor\* basePtr = nullptr;

GameActor\* hivePtr = nullptr;

int mapHeight; // num of tile rows

int mapWidth; // num of tile columns

Camera2D camera = { 0 };

Vector2 mousePosition;

Vector2 mouseWorldPosition;

TileIndex mouseIndex;

Vector2 screenSize;

Vector2 mapSize; //mapsize in pixels

Rectangle viewBorders;

int renderBorders[4]; //indices: top horiz, left vert, bottom horiz, right vert

ActorType wantToBuild = ActorType::ACTOR\_NULL;

bool wantToRemove = false;

//INTERFACE

bool showingCreepStates = false;

Side visionSide = Side::MACHINES;

bool gameOver = false;

//DAMAGE MAP

//RESOURCES

unsigned resourcesInsects = 0;

unsigned resourcesMachines = 1000;

//оставить тут методы "приказов", а вызывать их извне в общем цикле?

//FOG OF WAR

std::map<Side, int\*\*> mapsFogOfWar;

TileIndex insectsDesirePosition = {0,0};

public:

unsigned short timeCount; //for Update()

unsigned int lastID = 0;

const float pixelsPerTile = 16.f;

int getMaxWidth() { return mapWidth; }

int getMaxHeight() { return mapHeight; }

//attributes for GameActor constructor

std::map<ActorType, std::map<std::string, int>> genericAttributes;

std::map<ActorType, std::map<std::string, int>> buildingsAttributes;

std::map<ActorType, std::map<std::string, int>> militaryAttributes;

std::map<ActorType, std::map<std::string, int>> connectableAttributes;

std::map<ActorType, std::map<std::string, int>> constructorsAttributes;

std::map<ActorType, std::map<std::string, int>> turretsAttributes;

Animation getUnitAnimation(ActorType type, State state) { return unitAnimations[type][state]; }

//ECONOMICS

//returns amount of resources, that can be used

int trySpendResources(int amount, Side side);

//actual spend of resources

void spendResources(int amount, Side side);

//num of creep-covered tiles

unsigned int creepTilesCount = 0;

//num of energy layer tiles

unsigned int energisedTilesCount = 0;

unsigned char\*\* mapExpansionCreep = nullptr;

unsigned char\*\* mapExpansionEnergised = nullptr;

//return num of adjoined expansion tiles

int numOfExpansionTileAdjoin(int x, int y, Side side);

int numOfExpansionTileAdjoinFading(int x, int y, Side side);

bool isTileExpanded(TileIndex tile, Side side);

bool isTileInBounds(TileIndex tile);

//recalculate state of expansion tiles

void recalculateExpansion(Side side);

//if some unit dies, recalculate targets for opposite side

void recalculateMilitaryTargets(Side side);

//VECTOR PATHFINDING

//pre-calculated matrix with neighbors indices

NeighborsIndex\*\* neighborsIndices = nullptr;

//have inside mapsVector, mapsHeat, mapsTerrainMod, mapsDamage

std::map<ActorType, std::map<std::string, float\*\*>> mapsPathfinding;

std::map<ActorType, Vector2\*\*> vectorFields;

//mapVector - target mapVector to calculate, mapTerrainMod - speed modification by terrain type, mapDamage - modification by receiving/dealing damage

void calculateVectorPathfinding(TileIndex target, ActorType actorType);

//reveals fog of war in circle with radius (in tiles) around TileIndex position

void revealTerritory(TileIndex position, int radius, Side side);

TerrainType getTerrainType(int x, int y); //{return this->mapTerrain[x][y]};

bool closed = false;

std::vector<TileIndex>tilesInsideCircle(Vector2 center, unsigned int radius);

std::vector<TileIndex>tilesInsideCircleOrdered(TileIndex center, int radius);

//std::vector<TileIndex>tilesInPerimeterCircle(TileIndex center, unsigned int radius);

//return TileIndex of Vector2

TileIndex getTileIndex(Vector2);

//returns vector of neightbors indices

NeighborsIndex getNeighbors(int x, int y);

//returns vector of neighbors tiles

std::vector<TileIndex> getNeighborsAsVector(int x, int y);

//returns pointer to actor in coordinates

GameActor\* getActorInTile(int x, int y);

//returns vector of actors pointers in radius of circle

std::vector<GameActor\*> getActorsInRadius(Vector2 center, float radius);

GameActor\* getNearestSpecificActor(Vector2 position, std::vector<GameActor\*> actors, GameActor\* caller, ActorType type = ActorType::ACTOR\_NULL);

//returns true, if no obstacles on line of sight for this unit

bool isOnLineOfSight(TileIndex pos1, TileIndex pos2, ActorType unitType);

void addActor(ActorType type, Vector2 position, State state); //add actor on map, on full health or not - depends on "state" and debug mod on/off

void removeActor(unsigned int ID);

void Hit(GameActor\* target, int damage, ActorType hitBy); //return true, if actor destroyed by hit

void setTerrain(Terrain);

bool isMapLoaded();

void clearMap();

TileIndex getInsectsDesirePosition() { return insectsDesirePosition; };

//для вызова обновлений и отрисовки по всем актерам, вычисления экономических тайлов и т.п.

void GameDraw();

void GameUpdate();

GameData();

~GameData();

void saveToFile(std::string fileName);

void loadFromFile(std::string fileName);

};

class GameActor {

protected:

Animation sprite; //заменить на структуру анимации+максимального числа кадров? надо глянуть, как анимация реализована у других

int currentFrame; //current animation

Vector2 position;

TileIndex positionIndex;

State state;

int HP;

bool selectable; //удалить? все равно просматривать данные необходимо, а приказы отдавать не тут/сменить на selected, чтобы в Draw рисовать рамку, а в GUI хранить указатель на выбранного актера

int angle; //float?

public:

GameData\* game;

const ActorType type;

unsigned int ID; //private?

Side side;

std::string name;

//make consts?

int maxHP;

int size;

int armor;

int cost;

int inBattleCounter = 0;

bool inBattle = false; //if in battle - cant be repaired

//int buildCount; //just tune buildRate in cores and base

unsigned int sightRange; //при создании неподвижного актера единожды разведывать туман войны, для military(и турелей в мобильном режиме) обновлять каждый тик (или каждый переход на новую клетку)

virtual void Draw() = 0;

virtual void Update() = 0;

virtual void Destroy() = 0; //создать обломки/ошметки на карте, возможно даже не виртуальный, внутри использовать деструктор и/или удалять из вектора-хранилища данных игры

virtual ~GameActor()

{

}

//when spawning new unit, use GameData unit settings and ActorType for seting mapHP, size, etc. State defines currentHP and, maybe, something else

GameActor(GameData\* ptr, ActorType type, Vector2 pos, State state)

:game(ptr), type(type), position(pos), state(state)

{

size = ptr->genericAttributes[type]["size"];

maxHP = ptr->genericAttributes[type]["maxHP"];

cost = ptr->genericAttributes[type]["cost"];

sightRange = ptr->genericAttributes[type]["sightRange"];

armor = ptr->genericAttributes[type]["armor"];

positionIndex = ptr->getTileIndex(pos);

switch (state)

{

case State::ONLINE:

case State::OFFLINE:

case State::GOES:

HP = maxHP;

break;

case State::UNDER\_CONSTRUCTION:

HP = 0;

break;

default:

break;

}

switch (type)

{

case ActorType::LIGHT\_TURRET:

case ActorType::HEAVY\_TURRET:

case ActorType::AIRDEFENSE\_TURRET:

case ActorType::CORE:

case ActorType::BASE:

side = Side::MACHINES;

break;

case ActorType::LIGHT\_INSECT:

case ActorType::HEAVY\_INSECT:

case ActorType::FLYING\_INSECT:

case ActorType::HIVE:

case ActorType::TUMOR:

side = Side::INSECTS;

break;

}

this->ID = this->game->lastID;

this->game->lastID++;

ptr->revealTerritory(positionIndex, sightRange, side);

sprite = game->getUnitAnimation(type, state);

}

//draw bar with HP

void drawHP()

{

if (HP != maxHP)

{

DrawRectangle(position.x - size/2, position.y + size, size, 2, RED);

if(HP!=0)

DrawRectangle(position.x - size/2, position.y + size, size \* ((float)HP / (float)maxHP), 2, GREEN);

}

};

void isInBattleCheck()

{

inBattleCounter--;

if (inBattleCounter <= 0)

{

inBattleCounter = 0;

inBattle = false;

}

}

State getState() { return state; }

int getHP() { return this->HP; }

void setHP(int amount) { this->HP = amount; }

Vector2 getPosition() { return this->position; }

TileIndex getPositionIndex() { return this->positionIndex; }

//for building construction, repair and insects units and buildings regeneration

int RestoreHP(int amount)

{

int restored;

if ((this->HP + amount) > this->maxHP)

{

restored = this->maxHP - this->HP;

this->HP = this->maxHP;

}

else

{

restored = amount;

this->HP += amount;

}

return restored;

}

};

class Building: public GameActor {

private:

int expansionRange;

int expansionTime;

bool expanded; //true, if all tiles around this building is filled by expansion

std::vector<TileIndex> expansionIndices;

public:

Building(GameData\* ptr, ActorType type, Vector2 pos, State state);

void markAreaFade();

void markAreaExpand();

void Expand();

virtual ~Building() = 0;

};

class Tumor : public Building

{

public:

void Update();

void Draw();

void Destroy();

Tumor(GameData\* ptr, ActorType type, Vector2 pos, State state);

};

class Hive : public Building

{

public:

void Update();

void Draw();

void Destroy();

void SpawnUnits();

Hive(GameData\* ptr, ActorType type, Vector2 pos, State state);

};

class Constructor : public Building

{

//TODO: idea, common Draw() method for constructors

private:

int buildPower;

int buildRange;

GameActor\* target = nullptr;

int lastRayPosition = 0;

protected:

std::vector<Connectable\*>connectedUnits;

public:

Constructor(GameData\* ptr, ActorType type, Vector2 pos, State state);

~Constructor();

//returns true, if attaching unit is not a parent of this constructor, to prevent locking loops

bool RequestAttachment(Connectable\* unit);

void UnAttach(Connectable\* unit);

void BuildOrRepair();

void SeekForTarget();

void DrawBuildingRay();

};

class Connectable

{

private:

int connectRange;

GameData\* gameConn;

protected:

Constructor\* parent;

//return true, if connection succesful

bool TryConnect(Vector2 position, int ID);

public:

//recursive checking, is this unit have chain connection to Base

Connectable(GameData \*ptr, ActorType type);

~Connectable();

//use in destructor, or when parent destructed

virtual void Disconect();

};

class Core : public Constructor, public Connectable

{

public:

Core(GameData\* ptr, ActorType type, Vector2 pos, State state);

void Disconect();

void Update();

void Draw();

void Destroy();

};

class Base : public Constructor

{

public:

Base(GameData\* ptr, ActorType type, Vector2 pos, State state);

void Update();

void Draw();

void Destroy();

};

class Militaty : public GameActor

{

protected:

int seekRange;

int attackRange;

int speed;

int damage;

int cooldownDuration; //how much ticks need to recharge

int cooldownRemain; //amount of ticks, remains to reload

float rotationSpeed;

Vector2 velocityVector;

GameActor\* target;

ActorType targetPriority;

Militaty(GameData\* ptr, ActorType type, Vector2 pos, State state);

~Militaty();

virtual void Attack() = 0;

void Reload(); //IN TURRETS USE ONLY WHEN charge>=energyPerShot!

bool Targeting(); //turning actor in direction of enemy

void DrawReloadBar();

public:

void SeekForEnemy();

};

class Insect : public Militaty

{

void Move();

public:

Insect(GameData\* ptr, ActorType type, Vector2 pos, State state);

void Attack();

void Update();

void Draw();

void Destroy();

};

class Turret : public Militaty, public Connectable

{

protected:

Animation chasisSprite;

int chasisCurrentFrame;

int chasisAngle;

bool isMounted = true;

int maxCharge;

int charge; //amount of accumulated energy

int chargeRate; //how much energy takes per second

int energyPerShot;

std::vector<TileIndex> waypoints;

int modeProgressCounter = 0; //for changing mode

int attackProgressCounter = 0; //for counting attack

void Move();

void Recharge();

void DrawChargeBar();

void Attack();

public:

void ChangeMode();

//common for all of turrets (make only Attack() uniq):

void Update();

void Draw();

void Destroy();

Turret(GameData\* ptr, ActorType type, Vector2 pos, State state);

//~Turret();

};

**Enums.h**

#pragma once

#include "raylib.h"

#include <string>

#include <algorithm>

#include <vector>

#include <map>

#include <omp.h>

#include <math.h>

enum class Side

{

MACHINES,

INSECTS

};

enum class State

{

ONLINE,

OFFLINE,

UNDER\_CONSTRUCTION,

ATTACKING,

GOES,

IDLE, //remove?

CHANGING\_MODE, //for turrets mount/dismount

};

enum class ActorType

{

//for wantToBuild

ACTOR\_NULL,

//for Turret animation

TURRET\_CHASIS,

//Machines turrets

LIGHT\_TURRET,

HEAVY\_TURRET,

AIRDEFENSE\_TURRET,

//Insects units

LIGHT\_INSECT,

HEAVY\_INSECT,

FLYING\_INSECT,

//Machines buildings

CORE,

BASE,

//Insects buildings

HIVE,

TUMOR

};

enum class TerrainType

{

LAKE,

SWAMP,

SAND,

PLAIN,

TREE,

STONE,

MOUNTAIN,

ASH,

};

enum ExpandState

{

UNAVAILABLE,

AVAILABLE,

EXPANDED,

EXPANDED\_WITHOUT\_SOURCE,

};

enum class GameState

{

MAIN\_MENU,

TERRAIN\_GENERATOR,

GAMEPLAY,

OPTIONS //??

};

struct Terrain

{

int width;

int height;

TerrainType\* map;

};

struct TileIndex

{

int x;

int y;

operator Vector2() { return Vector2{ static\_cast<float>(x), static\_cast<float>(y) }; };

bool operator==(TileIndex rhs) { return (x == rhs.x) && (y == rhs.y); };

};

struct NeighborsIndex

{

TileIndex left = {-1,-1};

TileIndex up = { -1,-1 };

TileIndex right = { -1,-1 };

TileIndex down = { -1,-1 };

TileIndex upLeft = { -1,-1 };

TileIndex upRight = { -1,-1 };

TileIndex downLeft = { -1,-1 };

TileIndex downRight = { -1,-1 };

};

struct Animation

{

Texture2D\* frames;

int framesAmount;

int width = 0;

int height = 0;

};

inline int Vector2Angle(Vector2 v1, Vector2 v2)

{

int result = atan2f(v2.y - v1.y, v2.x - v1.x) \* (180.0f / PI);

if (result < 0) result += 360.0f;

return result;

}

inline float Vector2Distance(Vector2 v1, Vector2 v2)

{

float result = sqrtf((v1.x - v2.x) \* (v1.x - v2.x) + (v1.y - v2.y) \* (v1.y - v2.y));

return result;

}

inline float Vector2Length(Vector2 v)

{

float result = sqrtf((v.x \* v.x) + (v.y \* v.y));

return result;

}

inline float max(float a, float b)

{

if (a > b)

return a;

else

return b;

}

inline float min(float a, float b)

{

if (a < b)

return a;

else

return b;

}

**TerrainGenerator.cpp**

#include "TerrainGenerator.h"

#include <omp.h>

TerrainGenerator::TerrainGenerator()

{

closed = false;

levelNames = new const char\* [] { "LAKE", "SWAMP", "SAND", "PLAIN", "TREE", "STONE", "MOUNTAIN" };

levels = new float[] { 0., 0.35, 0.4, 0.45, 0.5, 0.6, 0.70, 1. };

palette = new Color[]{ DARKBLUE, BLUE, YELLOW, GREEN, DARKGREEN, GRAY, BLACK };

//TODO: make preset maps (empty map, for path finding etc.)

width = 128, height = 128, seed = 12345, octaves = 16;

frequency = 8.;

//TODO: replace memory allocation to regenerate/rerender

}

void TerrainGenerator::DrawInterface()

{

closed = GuiButton(Rectangle{ 20.f, 20.f, 100.f, 50.f }, "Back") || IsKeyPressed(KEY\_ESCAPE);

if (isGenerated())

{

DrawTextureEx(

colorPreview,

Vector2{ 50,100 },

0, (GetScreenWidth() \* 0.25f) / static\_cast<float>(colorPreview.width),

WHITE);

DrawTextureEx(

grayPreview,

Vector2{ (GetScreenWidth() \* 0.25f) + 100, 100 },

0, (GetScreenWidth() \* 0.25f) / static\_cast<float>(grayPreview.width), WHITE);

}

else

{

RegenerateTerrain();

RerenderTerrain();

}

//REPLACE IN CONSTRUCTOR PARAMETERS(??)

Vector2 guiZeroPoint = { GetScreenWidth()\*0.75, 0 };

//drawing levels sliders and retriving values from them

Rectangle position;

std::string buf;

for (int i = 1; i < 7; i++) // 7 - `cause 8 terrain levels, 8-1

{

buf = levelNames[i] + std::string(": %.2f");

position = { guiZeroPoint.x, guiZeroPoint.y + 35.f \*i, 120, 20 };

levels[i] = GuiSliderBar(

position,

TextFormat(buf.c\_str(), levels[i-1]),

TextFormat("%.2f", levels[i+1]),

levels[i],

levels[i - 1],

levels[i+1]

);

}

//level distribution visualisation

//TODO: solve problem with mountain visual

for (int i = 0; i < 8; i++)

{

position = { levels[i] \* 240.f + guiZeroPoint.x-60, guiZeroPoint.y + 250, (levels[i + 1] - levels[i]) \* 300, 30.f };

DrawText(TextFormat("%.1f", levels[i]), position.x, position.y - 15, 14, palette[i]);

DrawRectangleRec(position, palette[i]);

}

octaves = GuiSliderBar(

Rectangle{ guiZeroPoint.x, guiZeroPoint.y+ 410, 120, 20},

"Octaves: 1",

"16",

octaves,

1,

16

);

frequency = GuiSliderBar(

Rectangle{ guiZeroPoint.x, guiZeroPoint.y + 440, 120, 20 },

"Frequency: 1.0",

"16.0",

frequency,

1.f,

16.f

);

if (IsMouseButtonPressed(MOUSE\_LEFT\_BUTTON))

{

position = { guiZeroPoint.x, guiZeroPoint.y + 370, 60, 30 };

if (CheckCollisionPointRec(GetMousePosition(), position))

{

valueChanging[0] = true;

valueChanging[1] = false;

valueChanging[2] = false;

}

position = { guiZeroPoint.x, guiZeroPoint.y + 470, 60, 30 };

if (CheckCollisionPointRec(GetMousePosition(), position))

{

valueChanging[0] = false;

valueChanging[1] = true;

valueChanging[2] = false;

}

position = { guiZeroPoint.x + 120, guiZeroPoint.y + 470, 60, 30 };

if (CheckCollisionPointRec(GetMousePosition(), position))

{

valueChanging[0] = false;

valueChanging[1] = false;

valueChanging[2] = true;

}

}

//numerable params

position = { guiZeroPoint.x, guiZeroPoint.y + 370, 60, 30 };

GuiValueBox(position, "Generation seed:", &seed, 0, INT\_MAX, valueChanging[0]);

position = { guiZeroPoint.x, guiZeroPoint.y + 470, 60, 30 };

GuiValueBox(position, "Width:", &width, 1, 4048, valueChanging[1]);

position = { guiZeroPoint.x + 120, guiZeroPoint.y + 470, 60, 30 };

GuiValueBox(position, "Height:", &height, 1, 4048, valueChanging[2]);

//TODO: lock button, if width/height was changed and not regenerated, to prevent access violation

//Rerender button

if (GuiButton(Rectangle{ guiZeroPoint.x, guiZeroPoint.y + 300, 150.f, 50.f }, "Rerender"))

RerenderTerrain();

//Regenerate button

if (GuiButton(Rectangle{ guiZeroPoint.x, guiZeroPoint.y+ 510, 150.f, 50.f }, "Regenerate"))

{

RegenerateTerrain();

RerenderTerrain();

}

//TODO: add random seed button (with "refresh" icon)

//TODO: Add regenerate animation(??)

//TODO: ADD HISTOGRAM BELOW MAP PREVIEWS

}

void TerrainGenerator::RegenerateTerrain()

{

if (isGenerated())

delete[] noiseMap;

noiseMap = new double[width \* height];

int index;

const siv::PerlinNoise perlin((unsigned int)seed);

//TODO: regenerate according to chosen terrain size (with pointers)

fx = width / frequency;

fy = height / frequency;

#pragma omp parallel for private(index)

for (int y = 0; y < height; ++y)

{

for (int x = 0; x < width; ++x)

{

index = width \* y + x;

noiseMap[index] = perlin.accumulatedOctaveNoise2D\_0\_1(x / fx, y / fy, octaves);

}

}

}

void TerrainGenerator::RerenderTerrain()

{

if (isGenerated())

{

UnloadTexture(colorPreview);

UnloadTexture(grayPreview);

delete[] terrainMap;

}

terrainMap = new TerrainType[width \* height];

//updating color preview

Color\* colorPixels = new Color[width \* height];

unsigned char\* grayPixels = new unsigned char[width \* height];

#pragma omp parallel for

for (int i = 0; i < width \* height; i++)

{

for (int j = 0; j < 8; j++)

{

grayPixels[i] = (unsigned char)(noiseMap[i] \* 255.);

if (noiseMap[i] <= levels[j+1])

{

colorPixels[i] = palette[j];

terrainMap[i] = static\_cast<TerrainType>(j);

break;

}

}

}

Image colorImage = {

colorPixels,

width,

height,

1,

UNCOMPRESSED\_R8G8B8A8

};

Image grayImage = {

grayPixels,

width,

height,

1,

UNCOMPRESSED\_GRAYSCALE

};

colorPreview = LoadTextureFromImage(colorImage);

grayPreview = LoadTextureFromImage(grayImage);

UnloadImage(colorImage);

UnloadImage(grayImage);

}

TerrainGenerator::~TerrainGenerator()

{

delete[] levels;

delete[] noiseMap;

delete[] palette;

delete[] levelNames;

}

bool TerrainGenerator::isGenerated()

{

return terrainMap != nullptr;

}

Terrain TerrainGenerator::getMap()

{

return Terrain{width, height, terrainMap};

}

**Main.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define RAYGUI\_IMPLEMENTATION

#include "PerlinNoise.h"

#include <cassert>

#include <iostream>

#include <fstream>

#include <sstream>

#include <algorithm>

#include "raygui.h"

#include "raylib.h"

#include "TerrainGenerator.h"

int main(void)

{

// Initialization

//--------------------------------------------------------------------------------------

const int screenWidth = 1700;

const int screenHeight = 600;

InitWindow(screenWidth, screenHeight, "raylib [core] map generation");

SetTargetFPS(60);

SetExitKey(KEY\_HOME); //releasing ESC-key for in-gameplay menu

bool gameClosed = false;

GameState state = GameState::MAIN\_MENU;

GameData gameData;

TerrainGenerator terrain{};

// Main game loop

while (!gameClosed) // Detect window close button

{

BeginDrawing();

ClearBackground(Color{ 150,150,150,255 });

switch (state)

{

//TODO: same zeropoint-based visualisation

case GameState::MAIN\_MENU:

if (GuiButton(Rectangle{ screenWidth /2.f, 200, 100,50}, "Play") && gameData.isMapLoaded())

state = GameState::GAMEPLAY;

if (GuiButton(Rectangle{ screenWidth / 2.f, 300, 100,50 }, "Generate terrain"))

state = GameState::TERRAIN\_GENERATOR;

if (GuiButton(Rectangle{ screenWidth / 2.f, 400, 100,50 }, "Exit"))

gameClosed = true;

break;

case GameState::TERRAIN\_GENERATOR:

terrain.DrawInterface();

if (terrain.closed)

{

gameData.setTerrain(terrain.getMap());

state = GameState::MAIN\_MENU;

}

break;

case GameState::GAMEPLAY:

gameData.GameUpdate();

gameData.GameDraw();

if (gameData.closed)

{

state = GameState::MAIN\_MENU;

gameData.closed = false;

}

break;

case GameState::OPTIONS:

break;

default:

break;

}

EndDrawing();

}

CloseWindow(); // Close window and OpenGL context

return 0;

}

**GameData.cpp**

#include "GameClasses.h"

#define TILE\_DRAWING

GameData::GameData()

{

screenSize = Vector2{ static\_cast<float>(GetScreenWidth()), static\_cast<float>(GetScreenHeight()) };

camera.target = { screenSize.x / 2, screenSize.y / 2 };

camera.offset = { screenSize.x / 2, screenSize.y / 2 };

camera.rotation = 0.0f;

camera.zoom = 1.0f;

palette = new Color[]{ DARKBLUE, BLUE, YELLOW, GREEN, DARKGREEN, GRAY, BLACK };

//TILESET TEXTURES

const char\* filenames[] = {

"textures\\tileset\\LAKE.bmp",

"textures\\tileset\\SWAMP.bmp",

"textures\\tileset\\SAND.bmp",

"textures\\tileset\\PLAIN.bmp",

"textures\\tileset\\TREE.bmp",

"textures\\tileset\\STONE.bmp",

"textures\\tileset\\MOUNTAIN.bmp",

"textures\\tileset\\CREEP.png"

};

for (int i = 0; i < 8; i++)

{

tilesetTex[i] = LoadTexture(filenames[i]);

}

//EXPANSION TEXTURES

expansionInsectsTexture = LoadTexture("textures\\source\\creep\_1.png");

expansionMachinesAnimation = {

new Texture2D[15],

15

};

for (int i = 0; i < expansionMachinesAnimation.framesAmount; i++)

{

expansionMachinesAnimation.frames[i] = LoadTexture(TextFormat("textures\\tileset\\machines\_expansion\_frames\\%d.png", i));

}

expansionMachinesAnimation.width = expansionMachinesAnimation.frames[0].width;

expansionMachinesAnimation.height = expansionMachinesAnimation.frames[0].height;

//UNIT SPRITES

//HEAVY INSECT

unitAnimations[ActorType::HEAVY\_INSECT][State::GOES] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::HEAVY\_INSECT][State::GOES].frames[i] = LoadTexture(TextFormat("textures\\sprites\\heavyInsect\_goes\_%d.png", i));

}

//FLYING INSECT

unitAnimations[ActorType::FLYING\_INSECT][State::GOES] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::FLYING\_INSECT][State::GOES].frames[i] = LoadTexture(TextFormat("textures\\sprites\\flyingInsect\_goes\_%d.png", i));

}

//LIGHT INSECT

unitAnimations[ActorType::LIGHT\_INSECT][State::GOES] =

{

new Texture2D[5],

5

};

for (int i = 0; i < 5; i++)

{

unitAnimations[ActorType::LIGHT\_INSECT][State::GOES].frames[i] = LoadTexture(TextFormat("textures\\sprites\\lightInsect\_goes\_%d.png", i));

}

//TURRET CHASIS

//CHANGING\_MODE

unitAnimations[ActorType::TURRET\_CHASIS][State::CHANGING\_MODE] =

{

new Texture2D[4],

4

};

for (int i = 0; i < 4; i++)

{

unitAnimations[ActorType::TURRET\_CHASIS][State::CHANGING\_MODE].frames[i] = LoadTexture(TextFormat("textures\\sprites\\turretChasis\_changeMode\_%d.png", i));

}

//GOES

unitAnimations[ActorType::TURRET\_CHASIS][State::GOES] =

{

new Texture2D[3],

3

};

for (int i = 0; i < 3; i++)

{

unitAnimations[ActorType::TURRET\_CHASIS][State::GOES].frames[i] = LoadTexture(TextFormat("textures\\sprites\\turretChasis\_goes\_%d.png", i));

}

//HEAVY\_TURRET

unitAnimations[ActorType::HEAVY\_TURRET][State::ONLINE] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::HEAVY\_TURRET][State::ONLINE].frames[i] = LoadTexture(TextFormat("textures\\sprites\\heavyTurret\_online\_%d.png", i));

}

unitAnimations[ActorType::HEAVY\_TURRET][State::ATTACKING] =

{

new Texture2D[6],

6

};

for (int i = 0; i < 6; i++)

{

unitAnimations[ActorType::HEAVY\_TURRET][State::ATTACKING].frames[i] = LoadTexture(TextFormat("textures\\sprites\\heavyTurret\_attacking\_%d.png", i));

}

//LIGHT TURRET

unitAnimations[ActorType::LIGHT\_TURRET][State::ONLINE] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::LIGHT\_TURRET][State::ONLINE].frames[i] = LoadTexture(TextFormat("textures\\sprites\\lightTurret\_online\_%d.png", i));

}

unitAnimations[ActorType::LIGHT\_TURRET][State::ATTACKING] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::LIGHT\_TURRET][State::ATTACKING].frames[i] = LoadTexture(TextFormat("textures\\sprites\\lightTurret\_attacking\_%d.png", i));

}

//AIRDEFENSE TURRET

unitAnimations[ActorType::AIRDEFENSE\_TURRET][State::ONLINE] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::AIRDEFENSE\_TURRET][State::ONLINE].frames[i] = LoadTexture(TextFormat("textures\\sprites\\antiAirTurret\_online\_%d.png", i));

}

unitAnimations[ActorType::AIRDEFENSE\_TURRET][State::ATTACKING] =

{

new Texture2D[1],

1

};

for (int i = 0; i < 1; i++)

{

unitAnimations[ActorType::AIRDEFENSE\_TURRET][State::ATTACKING].frames[i] = LoadTexture(TextFormat("textures\\sprites\\antiAirTurret\_attacking\_%d.png", i));

}

//GENERIC ACTORS ATTRIBUTES

genericAttributes[ActorType::TUMOR] = std::map<std::string, int>{

{"maxHP", 100},

{"size", 16},

{"cost", 10},

{"sightRange", 10},

{"armor", 5},

};

genericAttributes[ActorType::CORE] = std::map<std::string, int>{

{"maxHP", 100},

{"size", 16},

{"cost", 50},

{"sightRange", 10},

{"armor", 4},

};

genericAttributes[ActorType::BASE] = std::map<std::string, int>{

{"maxHP", 300},

{"size", 32},

{"cost", 1000},

{"sightRange", 12},

{"armor", 10},

};

genericAttributes[ActorType::LIGHT\_TURRET] = std::map<std::string, int>{

{"maxHP", 100},

{"size", 12},

{"cost", 50},

{"sightRange", 12},

{"armor", 5},

};

genericAttributes[ActorType::HEAVY\_TURRET] = std::map<std::string, int>{

{"maxHP", 200},

{"size", 16},

{"cost", 50},

{"sightRange", 12},

{"armor", 5},

};

genericAttributes[ActorType::AIRDEFENSE\_TURRET] = std::map<std::string, int>{

{"maxHP", 150},

{"size", 12},

{"cost", 50},

{"sightRange", 12},

{"armor", 5},

};

genericAttributes[ActorType::LIGHT\_INSECT] = std::map<std::string, int>{

{"maxHP", 30},

{"size", 8},

{"cost", 10},

{"sightRange", 4},

{"armor", 1},

};

genericAttributes[ActorType::HEAVY\_INSECT] = std::map<std::string, int>{

{"maxHP", 100},

{"size", 10},

{"cost", 30},

{"sightRange", 3},

{"armor", 10},

};

genericAttributes[ActorType::FLYING\_INSECT] = std::map<std::string, int>{

{"maxHP", 70},

{"size", 8},

{"cost", 20},

{"sightRange", 5},

{"armor", 2},

};

//BUILDING ACTORS ATTRIBUTES

buildingsAttributes[ActorType::TUMOR] = std::map<std::string, int>{

{"expansionRange", 8},

{"expansionTime", 8 },

};

buildingsAttributes[ActorType::HIVE] = std::map<std::string, int>{

{"expansionRange", 12},

{"expansionTime", 8 },

};

buildingsAttributes[ActorType::CORE] = std::map<std::string, int>{

{"expansionRange", 8},

{"expansionTime", 4 },

};

buildingsAttributes[ActorType::BASE] = std::map<std::string, int>{

{"expansionRange", 12},

{"expansionTime", 4 },

};

//MILITARY ACTORS ATTRIBUTES

militaryAttributes[ActorType::LIGHT\_INSECT] = std::map<std::string, int>{

{"seekRange", 8 \* pixelsPerTile},

{"attackRange", 2 \* pixelsPerTile},

{"speed", 2},

{"damage", 5},

{"rotationSpeed", 8},

{"cooldownDuration", 30}, //ticks to reload

};

militaryAttributes[ActorType::HEAVY\_INSECT] = std::map<std::string, int>{

{"seekRange", 8 \* pixelsPerTile},

{"attackRange", 2 \* pixelsPerTile},

{"speed", 1},

{"damage", 30},

{"rotationSpeed", 4},

{"cooldownDuration", 60}, //ticks to reload

};

militaryAttributes[ActorType::FLYING\_INSECT] = std::map<std::string, int>{

{"seekRange", 8 \* pixelsPerTile},

{"attackRange", 2 \* pixelsPerTile},

{"speed", 2},

{"damage", 15},

{"rotationSpeed", 8},

{"cooldownDuration", 45}, //ticks to reload

};

militaryAttributes[ActorType::LIGHT\_TURRET] = std::map<std::string, int>{

{"seekRange", 10 \* pixelsPerTile},

{"attackRange", 6 \* pixelsPerTile},

{"speed", 1},

{"damage", 20},

{"rotationSpeed", 6},

{"cooldownDuration", 15}, //ticks to reload

};

militaryAttributes[ActorType::HEAVY\_TURRET] = std::map<std::string, int>{

{"seekRange", 12 \* pixelsPerTile},

{"attackRange", 10 \* pixelsPerTile},

{"speed", 1},

{"damage", 70},

{"rotationSpeed", 2},

{"cooldownDuration", 60}, //ticks to reload

};

militaryAttributes[ActorType::AIRDEFENSE\_TURRET] = std::map<std::string, int>{

{"seekRange", 16 \* pixelsPerTile},

{"attackRange", 12 \* pixelsPerTile},

{"speed", 1},

{"damage", 40},

{"rotationSpeed", 4},

{"cooldownDuration", 30}, //ticks to reload

};

//CONNECTABLE ACTORS ATTRIBUTES

connectableAttributes[ActorType::CORE] = std::map<std::string, int>

{

{"connectRange", 10 \* pixelsPerTile},

};

connectableAttributes[ActorType::LIGHT\_TURRET] = std::map<std::string, int>

{

{"connectRange", 6 \* pixelsPerTile},

};

connectableAttributes[ActorType::HEAVY\_TURRET] = std::map<std::string, int>

{

{"connectRange", 6 \* pixelsPerTile},

};

connectableAttributes[ActorType::AIRDEFENSE\_TURRET] = std::map<std::string, int>

{

{"connectRange", 6 \* pixelsPerTile},

};

//CONSTRUCTORS ATTRIBUTES

constructorsAttributes[ActorType::CORE] = std::map<std::string, int>

{

{"buildPower", 10},

{"buildRange", 8 \* pixelsPerTile},

};

constructorsAttributes[ActorType::BASE] = std::map<std::string, int>

{

{"buildPower", 20},

{"buildRange", 8 \* pixelsPerTile},

};

//TURRETS ATTRIBUTES

turretsAttributes[ActorType::LIGHT\_TURRET] = std::map<std::string, int>

{

{"maxCharge", 70},

{"chargeRate", 2},

{"energyPerShot", 3},

};

turretsAttributes[ActorType::HEAVY\_TURRET] = std::map<std::string, int>

{

{"maxCharge", 100},

{"chargeRate", 2},

{"energyPerShot", 20},

};

turretsAttributes[ActorType::AIRDEFENSE\_TURRET] = std::map<std::string, int>

{

{"maxCharge", 50},

{"chargeRate", 2},

{"energyPerShot", 7},

};

}

bool GameData::isMapLoaded()

{

return this->mapTerrain != nullptr;

}

void GameData::clearMap()

{

std::vector<unsigned int> IDs;

for (GameActor \*actor : this->unitsList)

{

if(actor!=nullptr)

IDs.push\_back(actor->ID);

}

for (unsigned int ID : IDs)

{

removeActor(ID);

}

lastID = 0;

resourcesInsects = 0;

resourcesMachines = 1000;

basePtr = nullptr;

//creepTilesCount = 0;

//energisedTilesCount = 0;

}

TerrainType GameData::getTerrainType(int x, int y)

{

return mapTerrain[y \* mapWidth + x];

}

void GameData::setTerrain(Terrain terr)

{

//TODO: make check for existence of path from Base to Hive (if not - create it)

//TODO: MAKE NEW FUNCTION TO REDRAW PREVIEW TEXTURE, CALL IT ONLY FEW TIMES IN UPDATE

if (isMapLoaded())

{

UnloadTexture(terrainTexture);

//expansion

for (int x = 0; x < mapWidth; x++)

delete[] mapExpansionCreep[x];

delete[] mapExpansionCreep;

//pathfinding

ActorType types[] = { ActorType::LIGHT\_INSECT, ActorType::HEAVY\_INSECT, ActorType::FLYING\_INSECT };

std::vector<std::string> matrices = { "mapsHeat", "mapsTerrainMod", "mapsDamage" };

for (ActorType type : types)

{

//support matrices

for (std::string matrixName : matrices)

{

for (int x = 0; x < mapWidth; x++)

delete[] mapsPathfinding[type][matrixName][x];

delete[] mapsPathfinding[type][matrixName];

}

//vector matrix

for (int x = 0; x < mapWidth; x++)

delete[] vectorFields[type][x];

delete[] vectorFields[type];

}

//neighbors matrix

for (int x = 0; x < mapWidth; x++)

delete[] neighborsIndices[x];

delete[] neighborsIndices;

//fog of war

Side sides[] = {Side::INSECTS, Side::MACHINES};

for (Side side : sides)

{

for (int x = 0; x < mapWidth; x++)

delete[] mapsFogOfWar[side][x];

delete[] mapsFogOfWar[side];

}

}

this->mapHeight = terr.height;

this->mapWidth = terr.width;

this->mapTerrain = terr.map;

mapSize.x = mapWidth \* pixelsPerTile;

mapSize.y = mapHeight \* pixelsPerTile;

//expansion maps memory allocation

mapExpansionCreep = new unsigned char\* [mapWidth];

for (int x = 0; x < mapWidth; x++)

mapExpansionCreep[x] = new unsigned char[mapHeight];

mapExpansionEnergised = new unsigned char\* [mapWidth];

for (int x = 0; x < mapWidth; x++)

mapExpansionEnergised[x] = new unsigned char[mapHeight];

//expansion pre-calculation

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

{

switch (getTerrainType(x, y))

{

//if cant expand in this tile

case TerrainType::LAKE:

case TerrainType::SWAMP:

case TerrainType::MOUNTAIN:

mapExpansionCreep[x][y] = ExpandState::UNAVAILABLE;

mapExpansionEnergised[x][y] = ExpandState::UNAVAILABLE;

break;

//if expansion can be provide

case TerrainType::SAND:

case TerrainType::PLAIN:

case TerrainType::TREE:

case TerrainType::STONE:

case TerrainType::ASH:

mapExpansionCreep[x][y] = ExpandState::AVAILABLE;

mapExpansionEnergised[x][y] = ExpandState::AVAILABLE;

break;

//just in case

default:

mapExpansionCreep[x][y] = ExpandState::UNAVAILABLE;

mapExpansionEnergised[x][y] = ExpandState::UNAVAILABLE;

break;

}

}

}

//tiles neighbors allocation

neighborsIndices = new NeighborsIndex \* [mapWidth];

for (int x = 0; x < mapWidth; x++)

neighborsIndices[x] = new NeighborsIndex[mapHeight];

//neighbors pre - calculation

for (int x = 0; x < mapWidth; x++)

for (int y = 0; y < mapHeight; y++)

neighborsIndices[x][y] = getNeighbors(x, y);

//vector pathfinding matrix allocation

ActorType types[] = { ActorType::LIGHT\_INSECT, ActorType::HEAVY\_INSECT, ActorType::FLYING\_INSECT };

std::vector<std::string> matrices = { "mapsHeat", "mapsTerrainMod", "mapsDamage" };

for (ActorType type : types)

{

//memory allocation, support matrices

for (std::string matrixName : matrices)

{

mapsPathfinding[type][matrixName] = new float\* [mapWidth];

for (int x = 0; x < mapWidth; x++)

mapsPathfinding[type][matrixName][x] = new float[mapHeight];

}

//vector matrices

vectorFields[type] = new Vector2\* [mapWidth];

for (int x = 0; x < mapWidth; x++)

vectorFields[type][x] = new Vector2[mapHeight];

//terrain speed modifier, individual for each Insect's unit type

float\*\* mapTerrainMod = mapsPathfinding[type]["mapsTerrainMod"];

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

{

if (type == ActorType::LIGHT\_INSECT)

{

switch (getTerrainType(x, y))

{

//cant walk in this tile

case TerrainType::LAKE:

case TerrainType::MOUNTAIN:

mapTerrainMod[x][y] = -1.f;

break;

//can, but much slower

case TerrainType::SWAMP:

mapTerrainMod[x][y] = 0.5f;

break;

case TerrainType::SAND:

mapTerrainMod[x][y] = 0.8f;

break;

case TerrainType::PLAIN:

case TerrainType::TREE:

case TerrainType::STONE:

case TerrainType::ASH:

mapTerrainMod[x][y] = 1.f;

break;

//just in case

default:

mapTerrainMod[x][y] = -1.f;

break;

}

}

if (type == ActorType::HEAVY\_INSECT)

{

switch (getTerrainType(x, y))

{

//cant walk in this tile

case TerrainType::MOUNTAIN:

mapTerrainMod[x][y] = -1.f;

break;

//can, but much slower

case TerrainType::LAKE:

case TerrainType::SWAMP:

mapTerrainMod[x][y] = 0.5f;

break;

case TerrainType::SAND:

case TerrainType::TREE:

mapTerrainMod[x][y] = 0.7f;

break;

case TerrainType::PLAIN:

case TerrainType::STONE:

case TerrainType::ASH:

mapTerrainMod[x][y] = 1.f;

break;

//just in case

default:

mapTerrainMod[x][y] = -1.f;

break;

}

}

if (type == ActorType::FLYING\_INSECT)

{

switch (getTerrainType(x, y))

{

//can fly through any tile:

case TerrainType::LAKE:

case TerrainType::MOUNTAIN:

case TerrainType::SWAMP:

case TerrainType::SAND:

case TerrainType::PLAIN:

case TerrainType::TREE:

case TerrainType::STONE:

case TerrainType::ASH:

mapTerrainMod[x][y] = 1.f;

break;

//just in case

default:

mapTerrainMod[x][y] = -1.f;

break;

}

}

}

}

}

//fog of war

Side sides[] = { Side::INSECTS, Side::MACHINES };

for (Side side : sides)

{

mapsFogOfWar[side] = new int\*[mapWidth];

for (int x = 0; x < mapWidth; x++)

mapsFogOfWar[side][x] = new int[mapHeight];

int\*\* ptr = mapsFogOfWar[side];

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

ptr[x][y] = -1;

}

}

#ifndef TILE\_DRAWING

unsigned short\* colorPixels = new unsigned short[mapSize.x \* mapSize.y];

Image buf = {

colorPixels,

mapSize.x,

mapSize.y,

1,

UNCOMPRESSED\_R5G6B5

};

int index;

//TODO: check blending principle for generating texture

#pragma omp parallel for private(index)

for (int i = 0; i < mapHeight; i++)

{

for (int j = 0; j < mapWidth; j++)

{

index = mapWidth \* i + j;

ImageDraw(

&buf,

tileset[static\_cast<int>(mapTerrain[index])],

Rectangle{ 0,0, pixelsPerTile, pixelsPerTile },

Rectangle{ j \* pixelsPerTile, i \* pixelsPerTile, pixelsPerTile, pixelsPerTile },

WHITE);

}

}

#else

Color\* colorPixels = new Color[mapHeight \* mapWidth];

Image buf = {

colorPixels,

mapWidth,

mapHeight,

1,

UNCOMPRESSED\_R8G8B8A8

};

#pragma omp parallel for

for (int i = 0; i < mapHeight \* mapWidth; i++)

{

colorPixels[i] = palette[static\_cast<int>(mapTerrain[i])];

}

#endif

terrainTexture = LoadTextureFromImage(buf);

UnloadImage(buf);

calculateVectorPathfinding(

TileIndex{ 0,0 },

ActorType::LIGHT\_INSECT

);

}

std::vector<TileIndex> GameData::tilesInsideCircle(Vector2 center, unsigned int radius)

{

std::vector<TileIndex> result;

TileIndex upLeftCorner = { std::clamp<int>(center.x - radius, 0, center.x), std::clamp<int>(center.y - radius, 0, center.y) };

TileIndex downRightCorner = { std::clamp<int>(center.x + radius, center.x, this->mapWidth-1), std::clamp<int>(center.y + radius, center.y, this->mapHeight-1) };

//reducing up limit `cause of max index of maps (not the actual indices)

for (int y = upLeftCorner.y; y <= downRightCorner.y; y++)

{

for (int x = upLeftCorner.x; x <= downRightCorner.x; x++)

{

if (CheckCollisionPointCircle(Vector2{static\_cast<float>(x), static\_cast<float>(y) }, center, radius))

result.push\_back(TileIndex{x,y});

//TODO: optimize this crap (and make parallel computaton?)

}

}

return result;

}

std::vector<TileIndex> GameData::tilesInsideCircleOrdered(TileIndex center, int radius)

{

std::vector<TileIndex> result;

for (int r = 1; r <= radius; r++)

{

for (int x = center.x + 1;x<center.x + r;x++)

{

for (int y = center.y; y > center.y - r; y--) //REVERSE int y = center.y - r; y < center.y + r; y++

{

if (CheckCollisionPointCircle(Vector2{ static\_cast<float>(x), static\_cast<float>(y) }, center, r))

{

//adding this and opposite tiles

result.push\_back(TileIndex{ x,y }); // up-right

result.push\_back(TileIndex{ center.x + (center.x - x), y }); //up-left

result.push\_back(TileIndex{ x, center.y + (center.y - y) }); //down-right

result.push\_back(TileIndex{ center.x + (center.x - x), center.y + (center.y - y) }); //down-left

}

}

}

result.push\_back(TileIndex{ center.x, center.y - r }); //up

result.push\_back(TileIndex{ center.x, center.y + r }); //down

result.push\_back(TileIndex{ center.x + r, center.y }); //right

result.push\_back(TileIndex{ center.x - r, center.y });//left

}

std::vector<TileIndex>::iterator iter;

for (iter = result.begin(); iter != result.end(); )

{

if ((\*iter).x < 0 ||

(\*iter).x>this->mapWidth - 1 ||

(\*iter).y < 0 ||

(\*iter).y>this->mapHeight - 1)

iter = result.erase(iter);

else

iter++;

}

return result;

}

int GameData::numOfExpansionTileAdjoin(int x, int y, Side side)

{

int result = 0;

if (side == Side::INSECTS)

{

if (x - 1 >= 0) //left

if (mapExpansionCreep[x - 1][y] == ExpandState::EXPANDED || mapExpansionCreep[x - 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y - 1 >= 0) //up

if (mapExpansionCreep[x][y - 1] == ExpandState::EXPANDED || mapExpansionCreep[x][y - 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (x + 1 < this->mapWidth) //right

if (mapExpansionCreep[x + 1][y] == ExpandState::EXPANDED || mapExpansionCreep[x + 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y + 1 < this->mapHeight) //down

if (mapExpansionCreep[x][y + 1] == ExpandState::EXPANDED || mapExpansionCreep[x][y + 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

}

if(side==Side::MACHINES)

{

if (x - 1 >= 0) //left

if (mapExpansionEnergised[x - 1][y] == ExpandState::EXPANDED || mapExpansionEnergised[x - 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y - 1 >= 0) //up

if (mapExpansionEnergised[x][y - 1] == ExpandState::EXPANDED || mapExpansionEnergised[x][y - 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (x + 1 < this->mapWidth) //right

if (mapExpansionEnergised[x + 1][y] == ExpandState::EXPANDED || mapExpansionEnergised[x + 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y + 1 < this->mapHeight) //down

if (mapExpansionEnergised[x][y + 1] == ExpandState::EXPANDED || mapExpansionEnergised[x][y + 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

}

return result;

}

int GameData::numOfExpansionTileAdjoinFading(int x, int y, Side side)

{

int result = 0;

if (side == Side::INSECTS)

{

if (x - 1 >= 0) //left

if (mapExpansionCreep[x - 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y - 1 >= 0) //up

if (mapExpansionCreep[x][y - 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (x + 1 < this->mapWidth) //right

if (mapExpansionCreep[x + 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y + 1 < this->mapHeight) //down

if (mapExpansionCreep[x][y + 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

}

if (side == Side::MACHINES)

{

if (x - 1 >= 0) //left

if (mapExpansionEnergised[x - 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y - 1 >= 0) //up

if (mapExpansionEnergised[x][y - 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (x + 1 < this->mapWidth) //right

if (mapExpansionEnergised[x + 1][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

if (y + 1 < this->mapHeight) //down

if (mapExpansionEnergised[x][y + 1] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

result++;

}

return result;

}

bool GameData::isTileExpanded(TileIndex tile, Side side)

{

if (side == Side::INSECTS)

{

if (mapExpansionCreep[tile.x][tile.y] == ExpandState::EXPANDED || mapExpansionCreep[tile.x][tile.y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

return true;

}

if(side == Side::MACHINES)

{

if (mapExpansionEnergised[tile.x][tile.y] == ExpandState::EXPANDED || mapExpansionEnergised[tile.x][tile.y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

return true;

}

return false;

}

bool GameData::isTileInBounds(TileIndex tile)

{

if (

tile.x < 0 ||

tile.x>this->mapWidth - 1 ||

tile.y < 0 ||

tile.y>this->mapHeight - 1

)

return false;

else

return true;

}

TileIndex GameData::getTileIndex(Vector2 position)

{

return TileIndex{

static\_cast<int>(position.x / pixelsPerTile),

static\_cast<int>(position.y / pixelsPerTile)

};

}

NeighborsIndex GameData::getNeighbors(int x, int y)

{

NeighborsIndex result;

if (x - 1 >= 0) //left

result.left = { x - 1, y };

if (y - 1 >= 0) //up

result.up = { x, y - 1 };

if (x + 1 < this->mapWidth) //right

result.right = { x + 1, y };

if (y + 1 < this->mapHeight) //down

result.down = { x, y + 1 };

//corners

if (y - 1 >= 0 && x - 1 >= 0) //up-left

result.upLeft = { x - 1, y - 1 };

if (y - 1 >= 0 && x + 1 < this->mapWidth) //up-right

result.upRight = { x + 1, y - 1 };

if (y + 1 < this->mapHeight && x - 1 >= 0) //down-left

result.downLeft = { x - 1, y + 1 };

if (y + 1 < this->mapHeight && x + 1 < this->mapWidth) //down-right

result.downRight = { x + 1, y + 1 };

return result;

}

std::vector<TileIndex> GameData::getNeighborsAsVector(int x, int y)

{

std::vector<TileIndex> result;

result.resize(9);

NeighborsIndex neighbors = neighborsIndices[x][y];

//this tile

result[0] = TileIndex{ x,y };

//left

if (neighbors.left.x != -1)

result[1] = neighbors.left;

else

result[1] = { -1,-1 };

//up

if (neighbors.up.x != -1)

result[2] = neighbors.up;

else

result[2] = { -1,-1 };

//right

if (neighbors.right.x != -1)

result[3] = neighbors.right;

else

result[3] = { -1,-1 };

//down

if (neighbors.down.x != -1)

result[4] = neighbors.down;

else

result[4] = { -1,-1 };

//upLeft

if (neighbors.upLeft.x != -1)

result[5] = neighbors.upLeft;

else

result[5] = { -1,-1 };

//upRight

if (neighbors.upRight.x != -1)

result[6] = neighbors.upRight;

else

result[6] = { -1,-1 };

//downLeft

if (neighbors.downLeft.x != -1)

result[7] = neighbors.downLeft;

else

result[7] = { -1,-1 };

//downRight

if (neighbors.downRight.x != -1)

result[8] = neighbors.downRight;

else

result[8] = { -1,-1 };

return result;

}

bool GameData::isOnLineOfSight(TileIndex pos1, TileIndex pos2, ActorType unitType)

{

bool result = true;

float\*\* mapMod = nullptr;

switch (unitType)

{

case ActorType::LIGHT\_TURRET:

case ActorType::HEAVY\_TURRET:

//can shoot only when no mountains on sight

mapMod = mapsPathfinding[ActorType::FLYING\_INSECT]["mapsTerrainMod"];

break;

case ActorType::AIRDEFENSE\_TURRET:

//can shoot everywhere

mapMod = mapsPathfinding[ActorType::FLYING\_INSECT]["mapsTerrainMod"];

break;

case ActorType::LIGHT\_INSECT:

case ActorType::HEAVY\_INSECT:

case ActorType::FLYING\_INSECT:

//checking line of sight according to type of insect

mapMod = mapsPathfinding[unitType]["mapsTerrainMod"];

break;

default:

break;

}

int x0 = pos1.x;

int y0 = pos1.y;

int x1 = pos2.x;

int y1 = pos2.y;

int dx = abs(x1 - x0);

int dy = abs(y1 - y0);

int x = x0;

int y = y0;

int n = 1 + dx + dy;

int x\_inc = (x1 > x0) ? 1 : -1;

int y\_inc = (y1 > y0) ? 1 : -1;

int error = dx - dy;

dx \*= 2;

dy \*= 2;

for (; n > 0; --n)

{

//checking tile here

if (mapMod[x][y] == -1.f)

return false;

if (error > 0)

{

x += x\_inc;

error -= dy;

}

else

{

y += y\_inc;

error += dx;

}

}

//returns true, if all checks fine

return result;

}

void GameData::recalculateExpansion(Side side)

{

if (side == Side::INSECTS)

{

for (Building\* actor : this->expansionUnitsList\_Insects)

{

if (actor->getState() == State::ONLINE)

actor->markAreaExpand();

}

}

if (side == Side::MACHINES)

{

for (Building\* actor : this->expansionUnitsList\_Machines)

{

if(actor->getState()==State::ONLINE)

actor->markAreaExpand();

}

}

}

void GameData::recalculateMilitaryTargets(Side side)

{

for (Militaty\* unit : militaryUnitsList)

{

if (unit->side == side)

unit->SeekForEnemy();

}

}

void GameData::calculateVectorPathfinding(TileIndex target, ActorType actorType)

{

float\*\* mapHeat = mapsPathfinding[actorType]["mapsHeat"];

float\*\* mapTerrainMod = mapsPathfinding[actorType]["mapsTerrainMod"];

Vector2\*\* mapVector = vectorFields[actorType];

int\*\* mapFogOfWar = mapsFogOfWar[Side::INSECTS];

//heat and vector map nullification

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

{

mapHeat[x][y] = 0.f;

mapVector[x][y] = { 0.f, 0.f };

}

}

//HEATMAP CALCULATION

mapHeat[target.x][target.y] = 1.f; //initial target point

//std::vector<TileIndex> toCheck = { target };

int i\_checking = 0, i\_max = mapHeight \* mapHeight, i\_last\_added = 0; //to replace vector

TileIndex\* toCheck = new TileIndex[i\_max];

toCheck[0] = target;

NeighborsIndex checking;

//std::vector<TileIndex> checking;

float previous = 1.f;

for(int i = 0; i<i\_max; i++)

{

if (i <= i\_last\_added) //for "repair" crush on Degug configuration

{

previous = mapHeat[toCheck[i].x][toCheck[i].y]; //previous - to calculate neighbors tiles

checking = neighborsIndices[toCheck[i].x][toCheck[i].y];

//checking = getNeighborsAsVector(toCheck[i].x, toCheck[i].y);

}

//left

if (checking.left.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.left.x][checking.left.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.left.x][checking.left.y] == -1)

{

i\_last\_added++;

mapHeat[checking.left.x][checking.left.y] = 1.f;

toCheck[i\_last\_added] = (checking.left);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.left.x][checking.left.y] != -1)

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.left.x][checking.left.y] = previous + (1 / mapTerrainMod[checking.left.x][checking.left.y]);

//add this checking.left for further checking

toCheck[i\_last\_added] = (checking.left);

}

}

}

}

//up

if (checking.up.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.up.x][checking.up.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.up.x][checking.up.y] == -1)

{

i\_last\_added++;

mapHeat[checking.up.x][checking.up.y] = 1.f;

toCheck[i\_last\_added] = (checking.up);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.up.x][checking.up.y] != -1)

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.up.x][checking.up.y] = previous + (1 / mapTerrainMod[checking.up.x][checking.up.y]);

//add this checking.up for further checking

toCheck[i\_last\_added] = (checking.up);

}

}

}

}

//right

if (checking.right.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.right.x][checking.right.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.right.x][checking.right.y] == -1)

{

i\_last\_added++;

mapHeat[checking.right.x][checking.right.y] = 1.f;

toCheck[i\_last\_added] = (checking.right);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.right.x][checking.right.y] != -1)

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.right.x][checking.right.y] = previous + (1 / mapTerrainMod[checking.right.x][checking.right.y]);

//add this checking.right for further checking

toCheck[i\_last\_added] = (checking.right);

}

}

}

}

//down

if (checking.down.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.down.x][checking.down.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.down.x][checking.down.y] == -1)

{

i\_last\_added++;

mapHeat[checking.down.x][checking.down.y] = 1.f;

toCheck[i\_last\_added] = (checking.down);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.down.x][checking.down.y] != -1)

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.down.x][checking.down.y] = previous + (1 / mapTerrainMod[checking.down.x][checking.down.y]);

//add this checking.down for further checking

toCheck[i\_last\_added] = (checking.down);

}

}

}

}

//upLeft

if (checking.upLeft.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.upLeft.x][checking.upLeft.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.upLeft.x][checking.upLeft.y] == -1)

{

i\_last\_added++;

mapHeat[checking.upLeft.x][checking.upLeft.y] = 1.f;

toCheck[i\_last\_added] = (checking.upLeft);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.upLeft.x][checking.upLeft.y] != -1 && (mapTerrainMod[checking.up.x][checking.up.y] != -1.f || mapTerrainMod[checking.left.x][checking.left.y] != -1.f))

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.upLeft.x][checking.upLeft.y] = previous + (1 / mapTerrainMod[checking.upLeft.x][checking.upLeft.y]);

//add this checking.upLeft for further checking

toCheck[i\_last\_added] = (checking.upLeft);

}

}

}

}

//upRight

if (checking.upRight.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.upRight.x][checking.upRight.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.upRight.x][checking.upRight.y] == -1)

{

i\_last\_added++;

mapHeat[checking.upRight.x][checking.upRight.y] = 1.f;

toCheck[i\_last\_added] = (checking.upRight);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.upRight.x][checking.upRight.y] != -1 && (mapTerrainMod[checking.up.x][checking.up.y] != -1.f || mapTerrainMod[checking.right.x][checking.right.y] != -1.f))

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.upRight.x][checking.upRight.y] = previous + (1 / mapTerrainMod[checking.upRight.x][checking.upRight.y]);

//add this checking.upRight for further checking

toCheck[i\_last\_added] = (checking.upRight);

}

}

}

}

//downLeft

if (checking.downLeft.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.downLeft.x][checking.downLeft.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.downLeft.x][checking.downLeft.y] == -1)

{

i\_last\_added++;

mapHeat[checking.downLeft.x][checking.downLeft.y] = 1.f;

toCheck[i\_last\_added] = (checking.downLeft);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.downLeft.x][checking.downLeft.y] != -1 && (mapTerrainMod[checking.down.x][checking.down.y] != -1.f || mapTerrainMod[checking.left.x][checking.left.y] != -1.f))

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.downLeft.x][checking.downLeft.y] = previous + (1 / mapTerrainMod[checking.downLeft.x][checking.downLeft.y]);

//add this checking.downLeft for further checking

toCheck[i\_last\_added] = (checking.downLeft);

}

}

}

}

//downRight

if (checking.downRight.x != -1) //if index valid

{

//if value not set:

if (mapHeat[checking.downRight.x][checking.downRight.y] == 0.f)

{

//if there's no fog of war:

if (mapFogOfWar[checking.downRight.x][checking.downRight.y] == -1)

{

i\_last\_added++;

mapHeat[checking.downRight.x][checking.downRight.y] = 1.f;

toCheck[i\_last\_added] = (checking.downRight);

}

else

{

//if there's no obstacle:

if (mapTerrainMod[checking.downRight.x][checking.downRight.y] != -1 && (mapTerrainMod[checking.down.x][checking.down.y] != -1.f || mapTerrainMod[checking.right.x][checking.right.y] != -1.f))

{

i\_last\_added++;

//set value, according to terrain speed modification and damage map

mapHeat[checking.downRight.x][checking.downRight.y] = previous + (1 / mapTerrainMod[checking.downRight.x][checking.downRight.y]);

//add this checking.downRight for further checking

toCheck[i\_last\_added] = (checking.downRight);

}

}

}

}

}

delete[] toCheck;

//VECTOR MAP CALCULATION

float x\_buf;

float y\_buf;

float minimumHeat;

Vector2 buf;

NeighborsIndex neighbors;

TileIndex upLeft, rightDown, indexMinimum;

//#pragma omp parallel for private(upLeft, rightDown, indexMinimum, minimumHeat, buf)

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

{

neighbors = neighborsIndices[x][y];

//TODO: remake for using gradient?

minimumHeat = mapHeat[x][y];

//left

if (neighbors.left.x != -1) //check for map borders

{

if (mapTerrainMod[x - 1][y] != -1.f) //check for obstruction

if (mapHeat[x - 1][y] < minimumHeat)

{

minimumHeat = mapHeat[x - 1][y];

mapVector[x][y] = { -1, 0 };

}

}

//right

if (neighbors.right.x != -1) //check for map borders

{

if (mapTerrainMod[x + 1][y] != -1.f) //check for obstruction

if (mapHeat[x + 1][y] < minimumHeat)

{

minimumHeat = mapHeat[x + 1][y];

mapVector[x][y] = { 1.f, 0.f };

}

}

//up

if (neighbors.up.x != -1) //check for map borders

{

if (mapTerrainMod[x][y - 1] != -1.f) //check for obstruction

if (mapHeat[x][y - 1] < minimumHeat)

{

minimumHeat = mapHeat[x][y - 1];

mapVector[x][y] = { 0.f, -1.f };

}

}

//down

if (neighbors.down.x != -1) //check for map borders

{

if (mapTerrainMod[x][y + 1] != -1.f) //check for obstruction

if (mapHeat[x][y + 1] < minimumHeat)

{

minimumHeat = mapHeat[x][y + 1];

mapVector[x][y] = { 0.f, 1.f };

}

}

//upLeft

if (neighbors.upLeft.x != -1) //check for map borders

{

if (mapTerrainMod[x - 1][y - 1] != -1.f && (mapTerrainMod[x][y - 1] != -1.f || mapTerrainMod[x - 1][y] != -1.f)) //check for obstruction

if (mapHeat[x - 1][y - 1] < minimumHeat)

{

minimumHeat = mapHeat[x - 1][y - 1];

mapVector[x][y] = { -1.f, -1.f };

}

}

//upRight

if (neighbors.upRight.x != -1) //check for map borders

{

if (mapTerrainMod[x + 1][y - 1] != -1.f && (mapTerrainMod[x][y - 1] != -1.f || mapTerrainMod[x + 1][y] != -1.f)) //check for obstruction

if (mapHeat[x + 1][y - 1] < minimumHeat)

{

minimumHeat = mapHeat[x + 1][y - 1];

mapVector[x][y] = { 1.f, -1.f };

}

}

//downLeft

if (neighbors.downLeft.x != -1) //check for map borders

{

if (mapTerrainMod[x - 1][y + 1] != -1.f && (mapTerrainMod[x][y + 1] != -1.f || mapTerrainMod[x - 1][y] != -1.f)) //check for obstruction

if (mapHeat[x - 1][y + 1] < minimumHeat)

{

minimumHeat = mapHeat[x - 1][y + 1];

mapVector[x][y] = { -1.f, 1.f };

}

}

//downRight

if (neighbors.downRight.x != -1) //check for map borders

{

if (mapTerrainMod[x + 1][y + 1] != -1.f && (mapTerrainMod[x][y + 1] != -1.f || mapTerrainMod[x + 1][y] != -1.f)) //check for obstruction

if (mapHeat[x + 1][y + 1] < minimumHeat)

{

minimumHeat = mapHeat[x + 1][y + 1];

mapVector[x][y] = { 1.f, 1.f };

}

}

}

}

}

void GameData::addActor(ActorType type, Vector2 position, State state)

{

Building\* buf\_building;

Militaty\* buf\_military;

Turret\* buf\_turret;

Constructor\* buf\_constructor;

switch (wantToBuild)

{

case ActorType::LIGHT\_TURRET:

case ActorType::HEAVY\_TURRET:

case ActorType::AIRDEFENSE\_TURRET:

buf\_turret = new Turret(

this,

wantToBuild,

position,

State::UNDER\_CONSTRUCTION);

unitsList.push\_back(buf\_turret);

militaryUnitsList.push\_back(buf\_turret);

turretUnitsList.push\_back(buf\_turret);

break;

case ActorType::LIGHT\_INSECT:

case ActorType::HEAVY\_INSECT:

case ActorType::FLYING\_INSECT:

buf\_military = new Insect(

this,

wantToBuild,

position,

State::GOES);

unitsList.push\_back(buf\_military);

militaryUnitsList.push\_back(buf\_military);

break;

case ActorType::CORE:

buf\_constructor = new Core(

this,

ActorType::CORE,

position,

State::UNDER\_CONSTRUCTION);

unitsList.push\_back(buf\_constructor);

expansionUnitsList\_Machines.push\_back(buf\_constructor);

constructorList.push\_back(buf\_constructor);

break;

case ActorType::BASE:

buf\_constructor = new Base(

this,

ActorType::BASE,

position,

State::ONLINE);

unitsList.push\_back(buf\_constructor);

expansionUnitsList\_Machines.push\_back(buf\_constructor);

basePtr = buf\_constructor;

constructorList.push\_back(buf\_constructor);

insectsDesirePosition = basePtr->getPositionIndex(); //once base placed, game begins

break;

case ActorType::HIVE:

break;

case ActorType::TUMOR:

buf\_building = new Tumor(

this,

ActorType::TUMOR,

position,

State::UNDER\_CONSTRUCTION);

unitsList.push\_back(buf\_building);

expansionUnitsList\_Insects.push\_back(buf\_building);

break;

default:

break;

}

//in any case nulify wantToBuild

//wantToBuild = ActorType::ACTOR\_NULL;

}

GameActor\* GameData::getActorInTile(int x, int y)

{

GameActor\* result = nullptr;

Vector2 pos;

for (GameActor\* actor : unitsList)

{

if (actor != nullptr)

{

pos = actor->getPosition();

Vector2 test = { pos.x / pixelsPerTile, pos.y / pixelsPerTile };

if (static\_cast<int>(pos.x / pixelsPerTile) == x && static\_cast<int>(pos.y / pixelsPerTile) == y)

{

result = actor;

}

}

}

return result;

}

std::vector<GameActor\*> GameData::getActorsInRadius(Vector2 center, float radius)

{

std::vector<GameActor\*> result;

for (GameActor\* actor: unitsList)

{

if (actor != nullptr)

{

if (CheckCollisionPointCircle(actor->getPosition(), center, radius))

{

result.push\_back(actor);

}

}

}

return result;

}

GameActor\* GameData::getNearestSpecificActor(Vector2 position, std::vector<GameActor\*> actors, GameActor\* caller, ActorType type)

{

GameActor\* result = nullptr;

double minDistance;

double buf;

//TODO: if type==ActorType::ACTOR\_NULL, do search"by default"

for (GameActor\* actor : actors)

{

if ((actor->type == type || type == ActorType::ACTOR\_NULL) && actor!=caller)

{

if (result == nullptr) //if no actor found yet

{

result = actor;

//eucledian distance

minDistance = sqrt(pow(position.x - actor->getPosition().x, 2) + pow(position.y - actor->getPosition().y, 2)); //+ actor->size;

}

else

{

buf = sqrt(pow(position.x - actor->getPosition().x, 2) + pow(position.y - actor->getPosition().y, 2));// + actor->size;

if (minDistance > buf)

{

minDistance = buf;

result = actor;

}

}

}

}

return result;

}

void GameData::removeActor(unsigned int ID)

{

std::vector<GameActor\*>::iterator iter;

GameActor\* buf = nullptr;

for (iter = unitsList.begin(); iter != unitsList.end(); iter++)

{

if ((\*iter) != nullptr)

{

if ((\*iter)->ID == ID)

{

buf = \*iter;

//iter = unitsList.erase(iter);

(\*iter) = nullptr;

break;

}

}

}

//seek in insects expansion units

std::vector<Building\*>::iterator expandIter;

for (expandIter = expansionUnitsList\_Insects.begin(); expandIter != expansionUnitsList\_Insects.end(); )

{

if ((\*expandIter)->ID == ID)

expandIter = expansionUnitsList\_Insects.erase(expandIter);

else

expandIter++;

}

//seek in machines expansion units

for (expandIter = expansionUnitsList\_Machines.begin(); expandIter != expansionUnitsList\_Machines.end(); )

{

if ((\*expandIter)->ID == ID)

expandIter = expansionUnitsList\_Machines.erase(expandIter);

else

expandIter++;

}

std::vector<Militaty\*>::iterator militaryIter;

for (militaryIter = militaryUnitsList.begin(); militaryIter != militaryUnitsList.end(); )

{

if ((\*militaryIter)->ID == ID)

militaryIter = militaryUnitsList.erase(militaryIter);

else

militaryIter++;

}

std::vector<Turret\*>::iterator turretIter;

for (turretIter = turretUnitsList.begin(); turretIter != turretUnitsList.end(); )

{

if ((\*turretIter)->ID == ID)

turretIter = turretUnitsList.erase(turretIter);

else

turretIter++;

}

std::vector<Constructor\*>::iterator constructorIter;

for (constructorIter = constructorList.begin(); constructorIter != constructorList.end(); )

{

if ((\*constructorIter)->ID == ID)

constructorIter = constructorList.erase(constructorIter);

else

constructorIter++;

}

if(buf!=nullptr)

delete buf;

}

void GameData::Hit(GameActor\* target, int damage, ActorType hitBy)

{

bool result = false;

damage -= target->armor;

if (damage <= 0) damage = 1; //every hit deals at least 1 damage

target->setHP(target->getHP() - damage);

target->inBattle = true;

target->inBattleCounter = 60;

//TODO: consider victory

if (target->getHP() <= 0)

{

Side opposite;

if (target->side == Side::INSECTS)

opposite = Side::MACHINES;

else

opposite = Side::INSECTS;

if (target->side == Side::MACHINES)

{

for (Constructor\* constructor : constructorList)

{

constructor->SeekForTarget();

}

}

removeActor(target->ID);

recalculateMilitaryTargets(opposite);

result = true;

}

//TODO: update damage maps here

}

void GameData::revealTerritory(TileIndex position, int radius, Side side)

{

//revealed tiles numered by distance between unit and tile

int\*\* matrix = mapsFogOfWar[side];

int buf;

//check tile, is it need to reveal for again

if (static\_cast<int>(matrix[position.x][position.y]) < radius) //WTF?!

{

matrix[position.x][position.y] = radius;

//TODO: reverse radius to descending, remake radius using

for (int r = 1; r <= radius; r++)

{

buf = radius - r;

for (int x = position.x + 1; x < position.x + r; x++)

{

for (int y = position.y; y > position.y - r; y--) //REVERSE int y = position.y - r; y < position.y + r; y++

{

if (CheckCollisionPointCircle(Vector2{ static\_cast<float>(x), static\_cast<float>(y) }, position, r))

{

//adding this and opposite tiles

if (isTileInBounds({ x, y }))

if(matrix[x][y]<buf)

matrix[x][y] = buf; // up-right

if (isTileInBounds({ position.x + (position.x - x), y }))

if (matrix[position.x + (position.x - x)][y] < buf)

matrix[position.x + (position.x - x)][y] = buf; //up-left

if (isTileInBounds({ x, position.y + (position.y - y) }))

if (matrix[x][position.y + (position.y - y)] < buf)

matrix[x][position.y + (position.y - y)] = buf; //down-right

if (isTileInBounds({ position.x + (position.x - x), position.y + (position.y - y) }))

if (matrix[position.x + (position.x - x)][position.y + (position.y - y)] < buf)

matrix[position.x + (position.x - x)][position.y + (position.y - y)] = buf; //down-left

}

}

}

if (isTileInBounds({ position.x, position.y - r }))

if (matrix[position.x][position.y - r] < buf)

matrix[position.x][position.y - r] = buf; //up

if (isTileInBounds({ position.x, position.y + r }))

if (matrix[position.x][position.y + r] < buf)

matrix[position.x][position.y + r] = buf; //down

if (isTileInBounds({ position.x + r, position.y }))

if (matrix[position.x + r][position.y] < buf)

matrix[position.x + r][position.y] = buf; //right

if (isTileInBounds({ position.x - r, position.y }))

if (matrix[position.x - r][position.y] < buf)

matrix[position.x - r][position.y] = buf;//left

}

//TODO: optimization - make vector field recalculation only when revealed tile count above zero (increase count, when matrix value changed FROM -1)

if (side == Side::INSECTS)

{

calculateVectorPathfinding(

insectsDesirePosition,

ActorType::LIGHT\_INSECT

);

calculateVectorPathfinding(

insectsDesirePosition,

ActorType::HEAVY\_INSECT

);

calculateVectorPathfinding(

insectsDesirePosition,

ActorType::FLYING\_INSECT

);

}

}

}

int GameData::trySpendResources(int amount, Side side)

{

int spended;

if (side == Side::INSECTS)

{

if (static\_cast<int>(resourcesInsects) - amount < 0)

{

spended = resourcesInsects;

}

else

{

spended = amount;

}

}

if(side == Side::MACHINES)

{

if (static\_cast<int>(resourcesMachines) - amount < 0)

{

spended = resourcesMachines;

}

else

{

spended = amount;

}

}

return spended;

}

void GameData::spendResources(int amount, Side side)

{

if (side == Side::INSECTS)

{

resourcesInsects -= amount;

}

if (side == Side::MACHINES)

{

resourcesMachines -= amount;

}

}

void GameData::GameUpdate()

{

//TIME COUNTER

timeCount++;

if (timeCount > 60)

timeCount = 0;

if (IsKeyPressed(KEY\_ESCAPE)) closed = true;

mousePosition = GetMousePosition();

mouseWorldPosition = GetScreenToWorld2D(mousePosition, camera);

mouseIndex = { std::clamp<int>(mouseWorldPosition.x/pixelsPerTile, 0, this->mapWidth-1), std::clamp<int>(mouseWorldPosition.y / pixelsPerTile, 0, this->mapHeight-1) };

// Camera shifting

if ((mousePosition.x < screenSize.x \* 0.15f || IsKeyDown(KEY\_A)) && camera.target.x > 100) camera.target.x -= 10 / camera.zoom;

if ((mousePosition.x > screenSize.x \* 0.85f || IsKeyDown(KEY\_D)) && camera.target.x < mapSize.x) camera.target.x += 10 / camera.zoom;

if ((mousePosition.y < screenSize.y \* 0.15f || IsKeyDown(KEY\_W)) && camera.target.y > 100) camera.target.y -= 10 / camera.zoom;

if ((mousePosition.y > screenSize.y \* 0.85f || IsKeyDown(KEY\_S)) && camera.target.y < mapSize.y) camera.target.y += 10 / camera.zoom;

// Camera zoom controls

camera.zoom += ((float)GetMouseWheelMove() \* 0.2f);

if (camera.zoom > 10.0f) camera.zoom = 10.0f;

else if (camera.zoom < 0.1f) camera.zoom = 0.1f;

// view borders calculation

Vector2 buf1 = GetScreenToWorld2D(Vector2{ 0,0 }, camera);

Vector2 buf2 = GetScreenToWorld2D(Vector2{ screenSize.x,screenSize.y }, camera);

viewBorders = {

buf1.x,

buf1.y,

buf2.x - buf1.x,

buf2.y - buf1.y

};

renderBorders[0] = (buf1.y / pixelsPerTile) - 1;

renderBorders[1] = (buf1.x / pixelsPerTile) - 1;

renderBorders[2] = (buf2.y / pixelsPerTile) + 1;

renderBorders[3] = (buf2.x / pixelsPerTile) + 1;

if (renderBorders[0] < 0) renderBorders[0] = 0;

if (renderBorders[1] < 0) renderBorders[1] = 0;

if (renderBorders[2] > mapHeight) renderBorders[2] = mapHeight;

if (renderBorders[3] > mapWidth) renderBorders[3] = mapWidth;

//TURRET BUILDING

if (IsKeyPressed(KEY\_ONE))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::LIGHT\_TURRET;

if (IsKeyPressed(KEY\_TWO))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::HEAVY\_TURRET;

if (IsKeyPressed(KEY\_THREE))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::AIRDEFENSE\_TURRET;

//MANUAL INSECTS SPAWN

if (IsKeyPressed(KEY\_FOUR))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::LIGHT\_INSECT;

if (IsKeyPressed(KEY\_FIVE))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::HEAVY\_INSECT;

if (IsKeyPressed(KEY\_SIX))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::FLYING\_INSECT;

//SPAWN TUMOR

if (IsKeyPressed(KEY\_E))

if(wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::TUMOR;

else

wantToBuild = ActorType::ACTOR\_NULL;

//BUILD CORE

if (IsKeyPressed(KEY\_Q))

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::CORE;

else

wantToBuild = ActorType::ACTOR\_NULL;

//BUILD BASE

if (IsKeyPressed(KEY\_B) && basePtr == nullptr)

if (wantToBuild == ActorType::ACTOR\_NULL)

wantToBuild = ActorType::BASE;

else

wantToBuild = ActorType::ACTOR\_NULL;

if (IsKeyPressed(KEY\_C))

clearMap();

//clear all fog of war by current vision side

if (IsKeyPressed(KEY\_V))

{

int\*\* fogOfWar = mapsFogOfWar[visionSide];

for (int x = 0; x < mapWidth; x++)

for (int y = 0; y < mapHeight; y++)

fogOfWar[x][y] = 100;

calculateVectorPathfinding(insectsDesirePosition, ActorType::LIGHT\_INSECT);

calculateVectorPathfinding(insectsDesirePosition, ActorType::HEAVY\_INSECT);

calculateVectorPathfinding(insectsDesirePosition, ActorType::FLYING\_INSECT);

}

if (IsKeyPressed(KEY\_R))

if (wantToRemove)

wantToRemove = false;

else

{

wantToBuild = ActorType::ACTOR\_NULL;

wantToRemove = true;

}

if (IsKeyPressed(KEY\_T))

{

insectsDesirePosition = mouseIndex;

calculateVectorPathfinding(

insectsDesirePosition,

ActorType::LIGHT\_INSECT

);

calculateVectorPathfinding(

insectsDesirePosition,

ActorType::HEAVY\_INSECT

);

calculateVectorPathfinding(

insectsDesirePosition,

ActorType::FLYING\_INSECT

);

}

if (IsKeyPressed(KEY\_F1))

if (visionSide == Side::INSECTS)

visionSide = Side::MACHINES;

else

visionSide = Side::INSECTS;

if (IsKeyPressed(KEY\_F2))

if (showingCreepStates)

showingCreepStates = false;

else

showingCreepStates = true;

if (IsKeyPressed(KEY\_SPACE))

if (gamePaused)

gamePaused = false;

else

gamePaused = true;

if (IsMouseButtonPressed(MOUSE\_RIGHT\_BUTTON) && wantToBuild != ActorType::ACTOR\_NULL)

{

//right-click to clear

wantToBuild = ActorType::ACTOR\_NULL;

}

//ACTORS BUILDING

if (IsMouseButtonPressed(MOUSE\_LEFT\_BUTTON) && wantToBuild != ActorType::ACTOR\_NULL)

{

Vector2 position = { mouseIndex.x \* pixelsPerTile + pixelsPerTile / 2, mouseIndex.y \* pixelsPerTile + pixelsPerTile / 2 };

addActor(wantToBuild, position, State::ONLINE);

}

if (!gamePaused && basePtr!=nullptr) //game starts only when base placed

{

//UNIT UPDATING

std::vector<GameActor\*>::iterator iter;

GameActor\* buf = nullptr;

for (iter = unitsList.begin(); iter != unitsList.end(); )

{

//removing pointers for killed units from vector

if ((\*iter) == nullptr)

{

iter = unitsList.erase(iter);

}

else

{

(\*iter)->Update();

iter++;

}

}

if (IsMouseButtonPressed(MOUSE\_LEFT\_BUTTON) && wantToRemove)

{

GameActor\* buf = getActorInTile(mouseIndex.x, mouseIndex.y);

if (buf != nullptr)

removeActor(buf->ID);

wantToRemove = false;

}

//expansion fading

if (timeCount == 0)

{

std::vector<TileIndex>buf;

//INSECTS

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

{

if (mapExpansionCreep[x][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

{

if (numOfExpansionTileAdjoinFading(x, y, Side::INSECTS) < 4)

buf.push\_back(TileIndex{ x,y });

}

}

}

creepTilesCount -= buf.size();

for (TileIndex tile : buf)

mapExpansionCreep[tile.x][tile.y] = ExpandState::AVAILABLE;

//MACHINES

buf.clear();

for (int x = 0; x < mapWidth; x++)

{

for (int y = 0; y < mapHeight; y++)

{

if (mapExpansionEnergised[x][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

{

if (numOfExpansionTileAdjoinFading(x, y, Side::MACHINES) < 4)

buf.push\_back(TileIndex{ x,y });

}

}

}

energisedTilesCount -= buf.size();

for (TileIndex tile : buf)

mapExpansionEnergised[tile.x][tile.y] = ExpandState::AVAILABLE;

}

//RESOURCES GATHERING

if (timeCount == 0)

{

resourcesInsects += creepTilesCount / 100;

resourcesMachines += energisedTilesCount / 100;

if (resourcesMachines > 1000) resourcesMachines = 1000; //maximum of Energy

}

//TILE ANIMATION

if (timeCount % 15 == 0)

{

currentFrame++;

if (currentFrame >= expansionMachinesAnimation.framesAmount)

currentFrame = 0;

}

}

}

void GameData::GameDraw()

{

BeginMode2D(camera);

#ifndef TILE\_DRAWING

DrawTextureEx(

terrainTexture,

Vector2{ 0.f, 0.f },

0.f, pixelsPerTile, WHITE);

#else

if (camera.zoom < 1.f /\*&& !IsKeyDown(KEY\_T)\*/) //low-detailed map

{

DrawTextureEx(

terrainTexture,

Vector2{ 0.f, 0.f },

0.f, pixelsPerTile, WHITE);

}

else

{ //full-detailed map

//black background for fog of war

DrawRectangle(0, 0, mapSize.x, mapSize.y, BLACK);

Vector2\*\* vectorField = vectorFields[ActorType::LIGHT\_INSECT];

float\*\* terrainMod = mapsPathfinding[ActorType::LIGHT\_INSECT]["mapsTerrainMod"];

int\*\* fogOfWar = mapsFogOfWar[visionSide];

int index;

for (int x = renderBorders[1]; x < renderBorders[3]; x++) //columns

{

for (int y = renderBorders[0]; y < renderBorders[2]; y++) //rows

{

if (fogOfWar[x][y] != -1)

{

index = mapWidth \* y + x;

//if there is creep:

if (mapExpansionCreep[x][y] == ExpandState::EXPANDED || mapExpansionCreep[x][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

{

float x\_exp = static\_cast<float>(x % (expansionInsectsTexture.width / static\_cast<int>(pixelsPerTile)));

float y\_exp = static\_cast<float>(y % (expansionInsectsTexture.height / static\_cast<int>(pixelsPerTile)));

Rectangle buf = {

x\_exp \* pixelsPerTile,

y\_exp \* pixelsPerTile,

pixelsPerTile,

pixelsPerTile };

DrawTextureRec(

expansionInsectsTexture,

buf,

Vector2{ x \* pixelsPerTile, y \* pixelsPerTile, },

WHITE);

}

else

{

//if there is energized tile:

if (mapExpansionEnergised[x][y] == ExpandState::EXPANDED || mapExpansionEnergised[x][y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

{

float x\_exp = static\_cast<float>(x % (expansionMachinesAnimation.width / static\_cast<int>(pixelsPerTile)));

float y\_exp = static\_cast<float>(y % (expansionMachinesAnimation.height / static\_cast<int>(pixelsPerTile)));

Rectangle buf = {

x\_exp \* pixelsPerTile,

y\_exp \* pixelsPerTile,

pixelsPerTile,

pixelsPerTile };

DrawTextureRec(

expansionMachinesAnimation.frames[currentFrame],

buf,

Vector2{ x \* pixelsPerTile, y \* pixelsPerTile, },

WHITE);

}

else

{

DrawTexture(

tilesetTex[static\_cast<int>(mapTerrain[index])],

x \* pixelsPerTile,

y \* pixelsPerTile,

WHITE);

}

}

}

//TODO: remove

if (showingCreepStates)

DrawText(FormatText("%d", mapExpansionCreep[x][y]), x \* pixelsPerTile + pixelsPerTile / 3, y \* pixelsPerTile + pixelsPerTile / 3, 14, RED);

//heatmap, light insects

if(IsKeyDown(KEY\_F4))

DrawText(FormatText("%.0f", mapsPathfinding[ActorType::LIGHT\_INSECT]["mapsHeat"][x][y]), x \* pixelsPerTile + pixelsPerTile / 3, y \* pixelsPerTile + pixelsPerTile / 3, 10, SKYBLUE);

//vector field, light insects

if (IsKeyDown(KEY\_F5) && terrainMod[x][y]!=-1.f)

{

DrawLineEx(

Vector2{ x \* pixelsPerTile + pixelsPerTile / 2,

y \* pixelsPerTile + pixelsPerTile / 2 },

Vector2{ x \* pixelsPerTile + pixelsPerTile / 2 + vectorField[x][y].x \*6,

y \* pixelsPerTile + pixelsPerTile / 2 + vectorField[x][y].y \* 6,

},

3.f,

GREEN);

DrawCircle(x \* pixelsPerTile + pixelsPerTile / 2, y \* pixelsPerTile + pixelsPerTile / 2, 1, RED);

}

//fog of war, INSECTS side

if(IsKeyDown(KEY\_F6))

DrawText(FormatText("%d", static\_cast<int>(fogOfWar[x][y])), x \* pixelsPerTile + pixelsPerTile / 3, y \* pixelsPerTile + pixelsPerTile / 3, 10, SKYBLUE);

}

}

}

#endif

if (this->wantToBuild != ActorType::ACTOR\_NULL)

{

//draw hologram of chosen actor in any case

Texture2D bufTexture;

int size;

Color tint;

unsigned int radius;

int\*\* fogOfWar = mapsFogOfWar[visionSide];

switch (wantToBuild)

{

case ActorType::LIGHT\_TURRET:

case ActorType::HEAVY\_TURRET:

case ActorType::AIRDEFENSE\_TURRET:

DrawCircleLines(

mouseIndex.x \* pixelsPerTile + pixelsPerTile / 2,

mouseIndex.y \* pixelsPerTile + pixelsPerTile / 2,

militaryAttributes[wantToBuild]["attackRange"],

RED

);

DrawCircleLines(

mouseIndex.x\* pixelsPerTile + pixelsPerTile / 2,

mouseIndex.y\* pixelsPerTile + pixelsPerTile / 2,

militaryAttributes[wantToBuild]["seekRange"],

ORANGE

);

DrawCircleLines(

mouseIndex.x\* pixelsPerTile + pixelsPerTile / 2,

mouseIndex.y\* pixelsPerTile + pixelsPerTile / 2,

connectableAttributes[wantToBuild]["connectRange"],

DARKBLUE

);

bufTexture = getUnitAnimation(ActorType::TURRET\_CHASIS, State::CHANGING\_MODE).frames[3];

size = genericAttributes[wantToBuild]["size"];

if (getTerrainType(mouseIndex.x, mouseIndex.y) != TerrainType::MOUNTAIN &&

getTerrainType(mouseIndex.x, mouseIndex.y) != TerrainType::LAKE

)

tint = GREEN;

else

tint = RED;

DrawTexturePro(

bufTexture, //texture

Rectangle{ 0,0, (float)bufTexture.width, (float)bufTexture.height }, //source

Rectangle{

mouseIndex.x \* pixelsPerTile + pixelsPerTile / 2,

mouseIndex.y \* pixelsPerTile + pixelsPerTile / 2,

(float)size \* 2.f, (float)size \* 2.f }, //dest

Vector2{ (float)(size), (float)(size) }, //origin

(float)0, //rotation

tint

);

break;

case ActorType::LIGHT\_INSECT:

case ActorType::HEAVY\_INSECT:

case ActorType::FLYING\_INSECT:

break;

case ActorType::CORE:

DrawCircleLines(

mouseIndex.x \* pixelsPerTile + pixelsPerTile / 2,

mouseIndex.y \* pixelsPerTile + pixelsPerTile / 2,

connectableAttributes[wantToBuild]["connectRange"],

YELLOW

);

case ActorType::BASE:

DrawCircleLines(

mouseIndex.x \* pixelsPerTile + pixelsPerTile / 2,

mouseIndex.y \* pixelsPerTile + pixelsPerTile / 2,

constructorsAttributes[wantToBuild]["buildRange"],

VIOLET

);

radius = buildingsAttributes[wantToBuild]["expansionRange"];

for (TileIndex tile : tilesInsideCircle(Vector2{ static\_cast<float>(mouseIndex.x), static\_cast<float>(mouseIndex.y) }, radius))

{

if(mapExpansionEnergised[tile.x][tile.y]==ExpandState::AVAILABLE && fogOfWar[tile.x][tile.y]!=-1)

DrawRectangle(tile.x \* pixelsPerTile, tile.y \* pixelsPerTile, pixelsPerTile, pixelsPerTile, Fade(SKYBLUE, 0.5f));

}

case ActorType::HIVE:

case ActorType::TUMOR:

//draw blue circle of expansion

break;

}

}

if (wantToRemove)

{

DrawRectangle(mouseIndex.x \* pixelsPerTile, mouseIndex.y \* pixelsPerTile, pixelsPerTile, pixelsPerTile, Fade(RED, 0.3f));

}

TileIndex buf\_position;

int\*\* mapFogOfWar = mapsFogOfWar[visionSide];

for (GameActor\* actor : this->unitsList)

{

//TODO: consider fog of war

if (actor != nullptr)

{

buf\_position = actor->getPositionIndex();

if (buf\_position.x >= renderBorders[1] &&

buf\_position.x < renderBorders[3] &&

buf\_position.y >= renderBorders[0] &&

buf\_position.y < renderBorders[2] &&

mapFogOfWar[buf\_position.x][buf\_position.y] != -1)

{

actor->Draw();

}

}

}

EndMode2D();

DrawText(FormatText("Camera zoom: %.2f", camera.zoom), 20, 20, 20, RED);

DrawText(FormatText("Tiles rendering: %d", (renderBorders[2] - renderBorders[0])\* (renderBorders[3] - renderBorders[1])), 20, 80, 20, RED);

DrawText(FormatText("Mouse tile index: %d, %d", mouseIndex.x, mouseIndex.y), 20, 140, 20, RED);

DrawText(FormatText("Desire position: %d, %d", insectsDesirePosition.x, insectsDesirePosition.y), 20, 200, 20, RED);

if(visionSide==Side::INSECTS)

DrawText("Vision side: INSECTS", 20, 260, 20, RED);

else

DrawText("Vision side: MACHINES", 20, 260, 20, RED);

DrawText(TextFormat("Creep: %d, food: %d", creepTilesCount, resourcesInsects), 20, 320, 20, RED);

DrawText(TextFormat("Zerolayer: %d, energy: %d", energisedTilesCount, resourcesMachines), 20, 380, 20, RED);

DrawFPS(20, 50);

}

GameData::~GameData()

{

//Unloading images and textures

if (isMapLoaded())

{

delete[] mapTerrain;

UnloadTexture(terrainTexture);

for (int x = 0; x < mapWidth; x++)

delete[] mapExpansionCreep[x];

delete[] mapExpansionCreep;

ActorType types[] = { ActorType::LIGHT\_INSECT, ActorType::HEAVY\_INSECT, ActorType::FLYING\_INSECT };

std::vector<std::string> matrices = { "mapsHeat", "mapsTerrainMod", "mapsDamage" };

for (ActorType type : types)

{

for (std::string matrixName : matrices)

{

for (int x = 0; x < mapWidth; x++)

delete[] mapsPathfinding[type][matrixName][x];

delete[] mapsPathfinding[type][matrixName];

}

//vector matrix

for (int x = 0; x < mapWidth; x++)

delete[] vectorFields[type][x];

delete[] vectorFields[type];

}

//neightbors matrix

for (int x = 0; x < mapWidth; x++)

delete[] neighborsIndices[x];

delete[] neighborsIndices;

//fog of war

Side sides[] = { Side::INSECTS, Side::MACHINES };

for (Side side : sides)

{

for (int x = 0; x < mapWidth; x++)

delete[] mapsFogOfWar[side][x];

delete[] mapsFogOfWar[side];

}

}

UnloadTexture(expansionInsectsTexture);

for (int i = 0; i < 7; i++)

{

UnloadTexture(tilesetTex[i]);

}

delete[] palette;

}

**Turret.cpp**

#include "GameClasses.h"

Turret::Turret(GameData\* ptr, ActorType type, Vector2 pos, State state)

:Militaty(ptr, type, pos, state),

Connectable(ptr, type)

{

maxCharge = ptr->turretsAttributes[type]["maxCharge"];

chargeRate = ptr->turretsAttributes[type]["chargeRate"];

energyPerShot = ptr->turretsAttributes[type]["energyPerShot"];

isMounted = true;

charge = 0;

chasisAngle = 0;

chasisSprite = game->getUnitAnimation(ActorType::TURRET\_CHASIS, State::CHANGING\_MODE);

chasisCurrentFrame = chasisSprite.framesAmount - 1;

}

void Turret::Recharge()

{

if (charge < maxCharge)

{

int spended = game->trySpendResources(std::min(maxCharge - charge, chargeRate), side);

charge += spended;

game->spendResources(spended, side);

}

}

void Turret::Attack()

{

//TODO::make different attacks for all turrets

if(type==ActorType::HEAVY\_TURRET)

game->Hit(target, damage, type);

if (type == ActorType::LIGHT\_TURRET) //attacks units in front of turret (triangle-AoE)

{

Vector2 targetPosition = { position.x + attackRange \* cosf(angle \* PI / 180.f),position.y + attackRange \* sinf(angle \* PI / 180.f) };//= target->getPosition();

float s = 1 / sqrtf(3.f); //tan^-1 0.36 instead?

Vector2 pLeft = {

targetPosition.x + s \* (position.y - targetPosition.y),

targetPosition.y + s \* (targetPosition.x - position.x)

};

Vector2 pRight = {

targetPosition.x + s \* (targetPosition.y - position.y),

targetPosition.y + s \* (position.x - targetPosition.x)

};

std::vector <GameActor\*> buf\_targets = game->getActorsInRadius(position, Vector2Distance(position, target->getPosition()));

//TODO: seek for enemy that can be hited

for (GameActor\* actor : buf\_targets)

{

if (actor->type != ActorType::FLYING\_INSECT &&

actor->side != side &&

CheckCollisionPointTriangle(actor->getPosition(), position, pLeft, pRight))

{

game->Hit(actor, damage, type);

}

}

}

if (type == ActorType::AIRDEFENSE\_TURRET) //attacks units via AoE-attack (circle around target)

{

std::vector <GameActor\*> buf\_targets = game->getActorsInRadius(target->getPosition(), 16.f);

for (GameActor\* actor : buf\_targets)

{

if (actor->type == ActorType::FLYING\_INSECT)

{

game->Hit(actor, damage, type);

}

}

}

charge -= energyPerShot;

cooldownRemain = cooldownDuration;

}

void Turret::Update()

{

switch (state)

{

case State::ONLINE: //stationary mode, seeking for targets

//if no locked target - seek for it

if (target == nullptr)

SeekForEnemy();

else

{

//TODO: make check for angle (targeting return bool?)

//if there is target and distance between it and actor less than attack distance AND eniugh energy

if (Vector2Distance(position, target->getPosition()) <= attackRange && charge>= energyPerShot && Targeting())

{

state = State::ATTACKING;

sprite = game->getUnitAnimation(type, state);

currentFrame = 0;

}

}

break;

case State::UNDER\_CONSTRUCTION:

if (HP == maxHP)

{

state = State::ONLINE;

sprite = game->getUnitAnimation(type, state);

}

break;

case State::ATTACKING: //stationary mode, firing on targets

if (target == nullptr)

{

//if target destroyed

state = State::ONLINE;

}

else

{

if (Vector2Distance(position, target->getPosition()) <= attackRange && Targeting())

{

if (cooldownRemain == 0)

{

if (type == ActorType::HEAVY\_TURRET || type==ActorType::AIRDEFENSE\_TURRET)

{

//for animation

attackProgressCounter++;

currentFrame = attackProgressCounter / (60 / sprite.framesAmount);

if (currentFrame >= sprite.framesAmount)

{

currentFrame--;

}

if (attackProgressCounter == 60)

{

Attack(); //hit target and reset cooldown

attackProgressCounter = 0;

currentFrame = 0;

}

}

else

{

Attack();

currentFrame = 0;

}

}

}

}

break;

case State::GOES: //mobile mode, can't fire

if (waypoints.size() != 0)

{

//can move only when charged

if (charge != 0)

{

Move();

chasisAngle = angle;

game->revealTerritory(positionIndex, sightRange, side);

if (positionIndex == waypoints[0])

waypoints.erase(waypoints.begin());

//consume energy

if (game->timeCount % 10 == 0)

charge--;

//chasis animation

if (game->timeCount % 15 == 0)

chasisCurrentFrame++;

if (chasisCurrentFrame >= chasisSprite.framesAmount)

chasisCurrentFrame = 0;

}

}

break;

case State::CHANGING\_MODE:

modeProgressCounter--;

//set animation frame

if (isMounted) //if it was mounted

{

chasisCurrentFrame = chasisSprite.framesAmount - modeProgressCounter / (60 / chasisSprite.framesAmount);

}

else

{

chasisCurrentFrame = modeProgressCounter / (60 / chasisSprite.framesAmount);

}

if (modeProgressCounter == 0)

{

//if mode finally changed - check, which one mode it will be

if (isMounted) //if it was mounted

{

state = State::GOES;

chasisSprite = game->getUnitAnimation(ActorType::TURRET\_CHASIS, state);

Disconect();

}

else

state = State::ONLINE;

}

break;

}

Reload();

if (parent != nullptr) //if connected

{

//try to recharge, if construction complete and its not in mobile mode

if (state != State::UNDER\_CONSTRUCTION && state != State::GOES)

Recharge();

}

else

TryConnect(position, ID); //if not

isInBattleCheck();

}

void Turret::Move()

{

//!!!

//TODO: add consuming energy from battery, moving by waypoints

float\*\* terrainMod = game->mapsPathfinding[type]["mapsTerrainMod"];

Vector2 previousPosition = position;

//TODO: move method to new Insect class?

positionIndex = game->getTileIndex(position);

//if there is enemy - move toward him

Vector2 targetLocation = waypoints[0];

Vector2 buf = { static\_cast<float>(targetLocation.x - position.x), static\_cast<float>(targetLocation.y - position.y) };

velocityVector = { buf.x / Vector2Length(buf), buf.y / Vector2Length(buf) };

float speedMod = 1.f;//terrainMod[positionIndex.x][positionIndex.y];

velocityVector.x \*= speed;

velocityVector.y \*= speed;

//collision avoidance

Vector2 ahead = { position.x + velocityVector.x \* 4.f, position.y + velocityVector.y \* 4.f };

Vector2 steering = { 0.f, 0.f };

//UNIT STEERING AHEAD

GameActor\* nearestObstacle = game->getNearestSpecificActor(ahead, game->getActorsInRadius(ahead, size), this, type);

if (nearestObstacle != nullptr)

{

float R = sqrt(pow(ahead.x - nearestObstacle->getPosition().x, 2) + pow(ahead.y - nearestObstacle->getPosition().y, 2));

if (R != 0.f)

{

steering.x += (ahead.x - nearestObstacle->getPosition().x) / R;

steering.y += (ahead.y - nearestObstacle->getPosition().y) / R;

}

}

//NEARBY UNITS COLLISION CHECK

for (GameActor\* colActor : game->getActorsInRadius(position, size \* 2))

{

if (colActor != this)

{

if (CheckCollisionCircles(position, size, colActor->getPosition(), colActor->size))

{

float R = sqrt(pow(position.x - colActor->getPosition().x, 2) + pow(position.y - colActor->getPosition().y, 2)); //distance between circles

if (R != 0.f)

{

steering.x += (position.x - colActor->getPosition().x) / R;

steering.y += (position.y - colActor->getPosition().y) / R;

}

}

}

}

//WALLS CHECKING AHEAD

TileIndex tileAhead = game->getTileIndex(ahead);

std::vector<TileIndex> nearTiles = game->getNeighborsAsVector(tileAhead.x, tileAhead.y);

Rectangle tileBuf;

for (TileIndex checkingTile : nearTiles)

{

if (checkingTile.x != -1)

{

tileBuf = Rectangle{ checkingTile.x \* game->pixelsPerTile, checkingTile.y \* game->pixelsPerTile, game->pixelsPerTile, game->pixelsPerTile };

if (terrainMod[checkingTile.x][checkingTile.y] == -1.f && CheckCollisionCircleRec(ahead, size, tileBuf))

{

//if positionIndex near is collides with actor

float R = sqrt(pow(ahead.x - (tileBuf.x + (tileBuf.width / 2.f)), 2) + pow(ahead.y - (tileBuf.y + (tileBuf.height / 2.f)), 2));

if (R != 0.f)

{

steering.x += (ahead.x - (tileBuf.x + (tileBuf.width / 2.f))) / R;

steering.y += (ahead.y - (tileBuf.y + (tileBuf.height / 2.f))) / R;

}

}

}

}

//std::clamp <float>(steering.x, -1.f, 1.f);

//std::clamp <float>(steering.y, -1.f, 1.f);

velocityVector.x += steering.x;

velocityVector.y += steering.y;

velocityVector.x = std::clamp <float>(velocityVector.x, -speed, speed);

velocityVector.y = std::clamp <float>(velocityVector.y, -speed, speed);

if (speedMod > 0)

{

velocityVector.x \*= speedMod;

velocityVector.y \*= speedMod;

}

position.x += velocityVector.x;

position.y += velocityVector.y;

//WALLS COLLISION CHECK

nearTiles = game->getNeighborsAsVector(positionIndex.x, positionIndex.y);

for (TileIndex checkingTile : nearTiles)

{

if (checkingTile.x != -1)

{

tileBuf = Rectangle{ checkingTile.x \* game->pixelsPerTile, checkingTile.y \* game->pixelsPerTile, game->pixelsPerTile, game->pixelsPerTile };

if (terrainMod[checkingTile.x][checkingTile.y] == -1.f && CheckCollisionCircleRec(position, size, tileBuf))

{

//if positionIndex near is collides with actor

float R = sqrt(pow(position.x - (tileBuf.x + (tileBuf.width / 2.f)), 2) + pow(position.y - (tileBuf.y + (tileBuf.height / 2.f)), 2));

if (R != 0.f)

{

position.x += (position.x - (tileBuf.x + (tileBuf.width / 2.f))) / R;

position.y += (position.y - (tileBuf.y + (tileBuf.height / 2.f))) / R;

}

}

}

}

//MAP BOUNDS CHECK

if (position.x < 0 + size)

position.x = 0 + size;

if (position.y < 0 + size)

position.y = 0 + size;

if (position.x > game->getMaxWidth() \* game->pixelsPerTile - size)

position.x = game->getMaxWidth() \* game->pixelsPerTile - size;

if (position.y > game->getMaxHeight() \* game->pixelsPerTile - size)

position.y = game->getMaxHeight() \* game->pixelsPerTile - size;

angle = Vector2Angle(previousPosition, position);

}

void Turret::ChangeMode()

{

modeProgressCounter = 240; //4 sec.

state = State::CHANGING\_MODE;

chasisSprite = game->getUnitAnimation(ActorType::TURRET\_CHASIS, state);

}

void Turret::DrawChargeBar()

{

DrawRectangle(position.x - size / 2, position.y + size + 2, size, 2, Color{165, 100, 25, 255});

if (charge != 0)

DrawRectangle(position.x - size / 2, position.y + size+2, size \* ((float)charge / (float)maxCharge), 2, Color{ 255, 200, 15, 255 });

}

void Turret::Draw()

{

//TODO: separately draw gun and chassis

//chassis

DrawTexturePro(

chasisSprite.frames[chasisCurrentFrame], //texture

Rectangle{ 0,0, (float)chasisSprite.frames[chasisCurrentFrame].width, (float)chasisSprite.frames[chasisCurrentFrame].height }, //source

Rectangle{ position.x, position.y, (float)size \* 2.f, (float)size \* 2.f }, //dest

Vector2{ (float)(size), (float)(size) }, //origin

(float)chasisAngle, //rotation

WHITE);

//if culminating attack animation

if (state == State::ATTACKING && attackProgressCounter>50 && type == ActorType::HEAVY\_TURRET)

DrawLineEx(position, target->getPosition(), 1, RED); //laser beam

if (type == ActorType::LIGHT\_TURRET)

{

Vector2 targetPosition = { position.x + attackRange \* cosf(angle \* PI / 180.f),position.y + attackRange \* sinf(angle \* PI / 180.f) };//= target->getPosition();

float s = 1 / sqrtf(3.f); //tan^-1 0.36 instead?

Vector2 pLeft = {

targetPosition.x + s \* (position.y - targetPosition.y),

targetPosition.y + s \* (targetPosition.x - position.x)

};

Vector2 pRight = {

targetPosition.x + s \* (targetPosition.y - position.y),

targetPosition.y + s \* (position.x - targetPosition.x)

};

if(state==State::ATTACKING)

DrawTriangle(position, pLeft, pRight, Fade(Color{240, 50,50,255}, 0.25));

else

if(state!=State::UNDER\_CONSTRUCTION)

DrawTriangle(position, pLeft, pRight, Fade(YELLOW, 0.25));

}

if (state == State::ATTACKING && attackProgressCounter > 55 && type == ActorType::AIRDEFENSE\_TURRET)

{

Vector2 targetPos = target->getPosition();

DrawCircleGradient(targetPos.x, targetPos.y, 16.f, Fade(RED, 0.5), Fade(ORANGE,0.5));

}

//gun

if (state != State::UNDER\_CONSTRUCTION)

{

DrawTexturePro(

sprite.frames[currentFrame], //texture

Rectangle{ 0,0, (float)sprite.frames[currentFrame].width, (float)sprite.frames[currentFrame].height }, //source

Rectangle{ position.x, position.y, (float)size \* 2.f, (float)size \* 2.f }, //dest

Vector2{ (float)(size), (float)(size) }, //origin

(float)angle, //rotation

WHITE);

}

drawHP();

DrawChargeBar();

}

void Turret::Destroy()

{

}

**Military.cpp**

#include "GameClasses.h"

Militaty::Militaty(GameData\* ptr, ActorType type, Vector2 pos, State state)

:GameActor(ptr, type, pos, state)

{

seekRange = ptr->militaryAttributes[type]["seekRange"];

attackRange = ptr->militaryAttributes[type]["attackRange"];

speed = ptr->militaryAttributes[type]["speed"];

damage = ptr->militaryAttributes[type]["damage"];

cooldownDuration = ptr->militaryAttributes[type]["cooldownDuration"];

rotationSpeed = static\_cast<float>(ptr->militaryAttributes[type]["rotationSpeed"])/100.f;

cooldownRemain = cooldownDuration;

}

void Militaty::SeekForEnemy()

{

target = nullptr;

//what actors will be looking for

Side enemy;

if (side == Side::INSECTS)

enemy = Side::MACHINES;

else

enemy = Side::INSECTS;

std::vector<GameActor\*> buf;

for (GameActor\* actor : game->getActorsInRadius(position, seekRange)) // TODO: seek for nearest?

{

if (actor->side == enemy && game->isOnLineOfSight(positionIndex, actor->getPositionIndex(), type))

{

//additional check for different types of units:

if ((type == ActorType::LIGHT\_TURRET && actor->type != ActorType::FLYING\_INSECT) ||

(type == ActorType::HEAVY\_TURRET && actor->type != ActorType::FLYING\_INSECT) ||

(type == ActorType::AIRDEFENSE\_TURRET && actor->type == ActorType::FLYING\_INSECT) ||

side == Side::INSECTS) //insects can attack all machines buildings

{

buf.push\_back(actor);

}

}

}

if (buf.size() != 0)

target = game->getNearestSpecificActor(position, buf, this);

}

void Militaty::Reload()

{

if (cooldownRemain > 0)

cooldownRemain--;

}

bool Militaty::Targeting()

{

bool result = true;

Vector2 targetPosition = target->getPosition();

float destinationRotation = atan2f(position.y - targetPosition.y, position.x - targetPosition.x) + PI;

float angleTan = angle \* PI / 180.f;

if (destinationRotation != angleTan)

{

if (fmaxf(destinationRotation - angleTan, angleTan - destinationRotation) < rotationSpeed)

{

angleTan = destinationRotation;

}

else

{

if (destinationRotation > angleTan)

{

if (angleTan < destinationRotation - PI)

angleTan -= rotationSpeed;

else

angleTan += rotationSpeed;

}

else

{

if (angleTan > destinationRotation + PI)

angleTan += rotationSpeed;

else

angleTan -= rotationSpeed;

}

if (angleTan > PI \* 2.0f) angleTan = 0;

if (angleTan < 0) angleTan = PI \* 2.0f;

}

angle = angleTan \* 180.0 / PI;

result = destinationRotation == angleTan;

}

return result;

}

Militaty::~Militaty()

{

}

**Insect.cpp**

#include "GameClasses.h"

Insect::Insect(GameData\* ptr, ActorType type, Vector2 pos, State state)

:Militaty(ptr, type, pos, state)

{

}

void Insect::Update()

{

switch (state)

{

case State::GOES:

Move();

game->revealTerritory(positionIndex, sightRange, side);

//if no locked target - seek for it

if (target == nullptr)

SeekForEnemy();

else

{

//if there is target and distance between it and actor less than attack distance

if (Vector2Distance(position, target->getPosition())<=attackRange)

{

state = State::ATTACKING;

}

}

break;

case State::ATTACKING:

if (target == nullptr)

{

//if target destroyed

state = State::GOES;

}

else

{

//if not - hit target and reset cooldown

Targeting();

if (Vector2Distance(position, target->getPosition()) <= attackRange)

{

if (cooldownRemain == 0)

Attack();

}

else

state = State::GOES;

}

break;

}

Reload();

if (game->timeCount % 15 == 0)

currentFrame++;

if (currentFrame >= sprite.framesAmount)

currentFrame = 0;

}

void Insect::Move()

{

float\*\* terrainMod = game->mapsPathfinding[type]["mapsTerrainMod"];

Vector2 previousPosition = position;

//TODO: move method to new Insect class?

positionIndex = game->getTileIndex(position);

if (target == nullptr)

{

//if no enemy in sight - moving on vector field

velocityVector = game->vectorFields[type][positionIndex.x][positionIndex.y];

}

else

{

//if there is enemy - move toward him

Vector2 targetLocation = target->getPosition();

Vector2 buf = { static\_cast<float>(targetLocation.x - position.x), static\_cast<float>(targetLocation.y - position.y) };

velocityVector = { buf.x / Vector2Length(buf), buf.y / Vector2Length(buf) };

}

float speedMod = terrainMod[positionIndex.x][positionIndex.y];

//creep speed mod

if (game->isTileExpanded(positionIndex, side) && type != ActorType::FLYING\_INSECT)

{

speedMod \*= 1.25f;

}

velocityVector.x \*= speed;

velocityVector.y \*= speed;

//collision avoidance

Vector2 ahead = { position.x + velocityVector.x \* 4.f, position.y + velocityVector.y \* 4.f };

Vector2 steering = { 0.f, 0.f };

//UNIT STEERING AHEAD

GameActor\* nearestObstacle = game->getNearestSpecificActor(ahead, game->getActorsInRadius(ahead, size), this, type);

if (nearestObstacle != nullptr)

{

float R = sqrt(pow(ahead.x - nearestObstacle->getPosition().x, 2) + pow(ahead.y - nearestObstacle->getPosition().y, 2));

if (R != 0.f)

{

steering.x += (ahead.x - nearestObstacle->getPosition().x) / R;

steering.y += (ahead.y - nearestObstacle->getPosition().y) / R;

}

}

//NEARBY UNITS COLLISION CHECK

for (GameActor\* colActor : game->getActorsInRadius(position, size \* 2))

{

if (colActor != this)

{

if (CheckCollisionCircles(position, size, colActor->getPosition(), colActor->size))

{

float R = sqrt(pow(position.x - colActor->getPosition().x, 2) + pow(position.y - colActor->getPosition().y, 2)); //distance between circles

if (R != 0.f)

{

steering.x += (position.x - colActor->getPosition().x) / R;

steering.y += (position.y - colActor->getPosition().y) / R;

}

}

}

}

//WALLS CHECKING AHEAD

TileIndex tileAhead = game->getTileIndex(ahead);

std::vector<TileIndex> nearTiles = game->getNeighborsAsVector(tileAhead.x, tileAhead.y);

Rectangle tileBuf;

for (TileIndex checkingTile : nearTiles)

{

if (checkingTile.x != -1)

{

tileBuf = Rectangle{ checkingTile.x \* game->pixelsPerTile, checkingTile.y \* game->pixelsPerTile, game->pixelsPerTile, game->pixelsPerTile };

if (terrainMod[checkingTile.x][checkingTile.y] == -1.f && CheckCollisionCircleRec(ahead, size, tileBuf))

{

//if positionIndex near is collides with actor

float R = sqrt(pow(ahead.x - (tileBuf.x + (tileBuf.width / 2.f)), 2) + pow(ahead.y - (tileBuf.y + (tileBuf.height / 2.f)), 2));

if (R != 0.f)

{

steering.x += (ahead.x - (tileBuf.x + (tileBuf.width / 2.f))) / R;

steering.y += (ahead.y - (tileBuf.y + (tileBuf.height / 2.f))) / R;

}

}

}

}

velocityVector.x += steering.x;

velocityVector.y += steering.y;

velocityVector.x = std::clamp <float>(velocityVector.x, -speed, speed);

velocityVector.y = std::clamp <float>(velocityVector.y, -speed, speed);

if (speedMod > 0)

{

velocityVector.x \*= speedMod;

velocityVector.y \*= speedMod;

}

position.x += velocityVector.x;

position.y += velocityVector.y;

//WALLS COLLISION CHECK

nearTiles = game->getNeighborsAsVector(positionIndex.x, positionIndex.y);

for (TileIndex checkingTile : nearTiles)

{

if (checkingTile.x != -1)

{

tileBuf = Rectangle{ checkingTile.x \* game->pixelsPerTile, checkingTile.y \* game->pixelsPerTile, game->pixelsPerTile, game->pixelsPerTile };

if (terrainMod[checkingTile.x][checkingTile.y] == -1.f && CheckCollisionCircleRec(position, size, tileBuf))

{

//if positionIndex near is collides with actor

float R = sqrt(pow(position.x - (tileBuf.x + (tileBuf.width / 2.f)), 2) + pow(position.y - (tileBuf.y + (tileBuf.height / 2.f)), 2));

if (R != 0.f)

{

position.x += (position.x - (tileBuf.x + (tileBuf.width / 2.f))) / R;

position.y += (position.y - (tileBuf.y + (tileBuf.height / 2.f))) / R;

}

}

}

}

//MAP BOUNDS CHECK

if (position.x < 0 + size)

position.x = 0 + size;

if (position.y < 0 + size)

position.y = 0 + size;

if (position.x > game->getMaxWidth() \* game->pixelsPerTile - size)

position.x = game->getMaxWidth() \* game->pixelsPerTile - size;

if (position.y > game->getMaxHeight() \* game->pixelsPerTile - size)

position.y = game->getMaxHeight() \* game->pixelsPerTile - size;

angle = Vector2Angle(previousPosition, position);

}

void Insect::Attack()

{

game->Hit(target, damage, type);

cooldownRemain = cooldownDuration;

}

void Insect::Draw()

{

DrawTexturePro(

sprite.frames[currentFrame], //texture

Rectangle{ 0,0, (float)sprite.frames[currentFrame].width, (float)sprite.frames[currentFrame].height }, //source

Rectangle{ position.x, position.y, (float)size\*2.f, (float)size\*2.f }, //dest

Vector2{ (float)(size), (float)(size) }, //origin

(float)angle, //rotation

WHITE);

DrawCircleLines(position.x, position.y, size, ORANGE);

DrawCircle(position.x + velocityVector.x, position.y + velocityVector.y, 1, BLUE);

drawHP();

}

void Insect::Destroy()

{

}

**Tumor.cpp**

#include "GameClasses.h"

Tumor::Tumor(GameData\* ptr, ActorType type, Vector2 pos, State state)

:Building(ptr, type, pos, state)

{

}

void Tumor::Update()

{

switch (state)

{

case State::ONLINE:

//regeneration every second, if builded

if (this->game->timeCount == 0)

this->RestoreHP(1);

Expand();

break;

case State::UNDER\_CONSTRUCTION:

//building

if (this->game->timeCount % 5 == 0)

RestoreHP(1);

if (HP == maxHP)

{

//if it ready - mark area around to expand and create initiate expansion tile

state = State::ONLINE;

markAreaExpand();

}

break;

default:

break;

}

}

void Tumor::Draw()

{

DrawCircle(position.x, position.y, size, RED);

drawHP();

}

void Tumor::Destroy()

{

}

**Core.cpp**

#include "GameClasses.h"

Core::Core(GameData\* ptr, ActorType type, Vector2 pos, State state)

:Constructor(ptr, type, pos, state),

Connectable(ptr, type)

{

}

void Core::Update()

{

switch (state)

{

case State::ONLINE:

if (parent != nullptr)

{

//if connected - routine work

Expand();

SeekForTarget();

if (game->timeCount == 0)

{

BuildOrRepair();

}

}

else

{

//try to connect, if cant - change state to OFFLINE and release expanded area free

if (TryConnect(position, ID) != true)

{

state = State::OFFLINE;

markAreaFade();

}

}

break;

case State::OFFLINE:

//constant trying to connect, if it possible - remark expansion area

if (TryConnect(position, ID) == true)

{

state = State::ONLINE;

markAreaExpand();

}

break;

case State::UNDER\_CONSTRUCTION:

if (HP == maxHP)

{

state = State::ONLINE;

markAreaExpand();

}

break;

default:

break;

}

isInBattleCheck();

}

//dark magic

void Core::Disconect()

{

if (parent != nullptr)

parent->UnAttach(this);

parent = nullptr;

std::vector<Connectable\*> buf = connectedUnits;

for (Connectable\* unit : buf)

{

unit->Disconect();

}

state = State::OFFLINE;

markAreaFade();

}

void Core::Draw()

{

DrawCircle(position.x, position.y, size, SKYBLUE);

//draw connection line

if (parent != nullptr)

{

DrawLineEx(position, parent->getPosition(), 2, WHITE);

}

if(state==State::ONLINE)

DrawCircle(position.x, position.y, size / 2, GREEN);

if (state == State::OFFLINE)

DrawCircle(position.x, position.y, size / 2, ORANGE);

if (state == State::UNDER\_CONSTRUCTION)

DrawCircle(position.x, position.y, size / 2, RED);

drawHP();

DrawBuildingRay();

}

void Core::Destroy()

{

}

**Constructor.cpp**

#include "GameClasses.h"

Constructor::Constructor(GameData\* ptr, ActorType type, Vector2 pos, State state)

:Building(ptr, type, pos, state)

{

buildPower = ptr->constructorsAttributes[type]["buildPower"];

buildRange = ptr->constructorsAttributes[type]["buildRange"];

}

void Constructor::BuildOrRepair()

{

//if have target

if (target != nullptr)

{

//repair, if target fully repaired - nullify pointer

if (!target->inBattle/\* && target->cost != 0\*/)

{

//determine, how much resources can be spend

int canSpend = game->trySpendResources(buildPower, side);

if (canSpend != 0)

{

//technical singularity dark magic

int restored = target->RestoreHP(target->maxHP / (target->cost / canSpend));

if (restored != 0)

game->spendResources(min((target->cost \* restored) / target->maxHP, canSpend), side);

//if target hp restored, nullify pointer

if (target->getHP() == target->maxHP)

target = nullptr;

}

}

else

{

target = nullptr;

SeekForTarget();

}

}

}

bool Constructor::RequestAttachment(Connectable\* unit)

{

bool result = true;

for (Connectable\* connected : connectedUnits)

{

if (connected == unit)

{

result = false;

break;

}

}

if (result == true)

connectedUnits.push\_back(unit);

return result;

}

void Constructor::SeekForTarget()

{

if (target == nullptr)

{

for (GameActor\* unit : game->getActorsInRadius(position, buildRange))

{

if (!unit->inBattle) //check, if this unit attacked

{

if (unit->getHP() != unit->maxHP && unit->side == this->side)

{

target = unit;

break;

}

}

}

}

}

void Constructor::UnAttach(Connectable\* unit)

{

//seek for unit to remove from vector

std::vector<Connectable\*>::iterator iter;

for (iter = connectedUnits.begin(); iter != connectedUnits.end(); iter++)

{

if ((\*iter) == unit)

{

connectedUnits.erase(iter);

break;

}

}

}

void Constructor::DrawBuildingRay()

{

if (target != nullptr)

{

Vector2 offset;

Vector2 targetPosition = target->getPosition();

if (game->timeCount % 15 == 0)

lastRayPosition++;

if (lastRayPosition > 7)

lastRayPosition = 0;

switch (lastRayPosition)

{

case 0:

offset = { -1.f,-1.f };

break;

case 1:

offset = { 0.f,-1.f };

break;

case 2:

offset = { 0.f,-1.f };

break;

case 3:

offset = { -1.f,0.f };

break;

case 4:

offset = { 1.f,0.f };

break;

case 5:

offset = { -1.f,1.f };

break;

case 6:

offset = { 0.f,1.f };

break;

case 7:

offset = { 1.f,1.f };

break;

default:

offset = { 0.f,0.f };

break;

}

targetPosition.x += (size / 2) \* offset.x;

targetPosition.y += (size / 2) \* offset.y;

DrawLineEx(position, targetPosition, 2, VIOLET);

}

}

Constructor::~Constructor()

{

std::vector<Connectable\*> buf = connectedUnits;

for (Connectable\* unit : buf)

{

unit->Disconect();

}

}

**Connectable.cpp**

#include "GameClasses.h"

Connectable::Connectable(GameData\* ptr, ActorType type)

{

gameConn = ptr;

connectRange = gameConn->connectableAttributes[type]["connectRange"];

}

bool Connectable::TryConnect(Vector2 position, int ID)

{

bool result = false;

std::vector<GameActor\*> buf = gameConn->getActorsInRadius(position, connectRange);

for (GameActor\* actor : buf)

{

//seek for first Core or Base, that operational and NOT the same unit which connecting, and connect to it

if ((actor->type == ActorType::CORE || actor->type == ActorType::BASE)

&& actor->getState() == State::ONLINE

&& actor->ID !=ID)

{

//request to add this unit to parent`s vector of connected units, and check if this unit parent for connected or not

if (static\_cast<Constructor\*>(actor)->RequestAttachment(this))

{

parent = (Constructor\*)actor;

break;

}

}

}

if (parent != nullptr)

result = true;

return result;

}

void Connectable::Disconect()

{

if (parent != nullptr)

parent->UnAttach(this);

parent = nullptr;

}

Connectable::~Connectable()

{

Disconect();

}

**Building.cpp**

#include "GameClasses.h"

Building::Building(GameData\* ptr, ActorType type, Vector2 pos, State state)

:GameActor(ptr, type, pos, state)

{

expansionRange = ptr->buildingsAttributes[type]["expansionRange"];

expansionTime = ptr->buildingsAttributes[type]["expansionTime"];

//positionIndex = TileIndex{static\_cast<int>(pos.x / ptr->pixelsPerTile), static\_cast<int>(pos.y / ptr->pixelsPerTile) };

expansionIndices = this->game->tilesInsideCircleOrdered(this->positionIndex, expansionRange);

//only if it created operational

if(state==State::ONLINE)

markAreaExpand();

}

void Building::Expand()

{

if (!this->expanded)

{

if (this->game->timeCount % expansionTime == 0)

{

//TODO: further optimization - remove expanded tile from vector, on Destroy call recalculate it again to set mapExpansionCreep

for (TileIndex tile : expansionIndices)

{

if (this->side == Side::INSECTS)

{

if (game->mapExpansionCreep[tile.x][tile.y] == ExpandState::AVAILABLE)

{

if (game->numOfExpansionTileAdjoin(tile.x, tile.y, this->side) > 0)

{

game->mapExpansionCreep[tile.x][tile.y] = ExpandState::EXPANDED;

game->creepTilesCount++;

return;

}

}

}

if (this->side == Side::MACHINES)

{

if (game->mapExpansionEnergised[tile.x][tile.y] == ExpandState::AVAILABLE)

{

if (game->numOfExpansionTileAdjoin(tile.x, tile.y, this->side) > 0)

{

game->mapExpansionEnergised[tile.x][tile.y] = ExpandState::EXPANDED;

game->energisedTilesCount++;

return;

}

}

}

}

this->expanded = true;

}

}

}

void Building::markAreaExpand()

{

this->expanded = false;

//create creep or energised on position to initiate expansion

if (side == Side::INSECTS)

{

if(game->mapExpansionCreep[positionIndex.x][positionIndex.y]==ExpandState::AVAILABLE)

game->creepTilesCount++;

game->mapExpansionCreep[positionIndex.x][positionIndex.y] = ExpandState::EXPANDED;

}

if (side == Side::MACHINES)

{

if (game->mapExpansionEnergised[positionIndex.x][positionIndex.y] == ExpandState::AVAILABLE)

game->energisedTilesCount++;

game->mapExpansionEnergised[positionIndex.x][positionIndex.y] = ExpandState::EXPANDED;

}

for (TileIndex tile : expansionIndices)

{

if (this->side == Side::INSECTS)

{

if (game->mapExpansionCreep[tile.x][tile.y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

{

game->mapExpansionCreep[tile.x][tile.y] = ExpandState::EXPANDED;

}

}

if (this->side == Side::MACHINES)

{

if (game->mapExpansionEnergised[tile.x][tile.y] == ExpandState::EXPANDED\_WITHOUT\_SOURCE)

{

game->mapExpansionEnergised[tile.x][tile.y] = ExpandState::EXPANDED;

}

}

}

}

void Building::markAreaFade()

{

//setting expansion tiles around building to fade

for (TileIndex tile : expansionIndices)

{

if (this->side == Side::INSECTS)

{

if (game->mapExpansionCreep[tile.x][tile.y] == ExpandState::EXPANDED)

{

game->mapExpansionCreep[tile.x][tile.y] = ExpandState::EXPANDED\_WITHOUT\_SOURCE;

}

}

if (this->side == Side::MACHINES)

{

if (game->mapExpansionEnergised[tile.x][tile.y] == ExpandState::EXPANDED)

{

game->mapExpansionEnergised[tile.x][tile.y] = ExpandState::EXPANDED\_WITHOUT\_SOURCE;

}

}

}

if (this->side == Side::INSECTS)

if (game->mapExpansionCreep[positionIndex.x][positionIndex.y] == ExpandState::EXPANDED)

game->mapExpansionCreep[positionIndex.x][positionIndex.y] = ExpandState::EXPANDED\_WITHOUT\_SOURCE;

if (this->side == Side::MACHINES)

if (game->mapExpansionEnergised[positionIndex.x][positionIndex.y] == ExpandState::EXPANDED)

game->mapExpansionEnergised[positionIndex.x][positionIndex.y] = ExpandState::EXPANDED\_WITHOUT\_SOURCE;

//TODO: possible optimizaton - recalculate expansion in specific range (with getActorsInRadius method or kinda like that)

//recalculate expansion tiles to overlap faded tiles by existing buildings

this->game->recalculateExpansion(this->side);

}

Building::~Building()

{

markAreaFade();

}

**Base.cpp**

#include "GameClasses.h"

Base::Base(GameData\* ptr, ActorType type, Vector2 pos, State state)

:Constructor(ptr, type, pos, state)

{

}

void Base::Update()

{

Expand();

SeekForTarget();

isInBattleCheck();

if (game->timeCount == 0)

{

BuildOrRepair();

}

}

void Base::Draw()

{

DrawCircle(position.x, position.y, size, BLUE);

drawHP();

DrawBuildingRay();

}

void Base::Destroy()

{

}

**ДОДАТОК Б**

**Відгук керівника економічного розділу**

**ДОДАТОК В**

**ПереЛІК документІВ на оптичному НосІЇ**

|  |  |
| --- | --- |
| **Ім’я файлу** | **Опис** |
| Пояснювальні документи | |
| Диплом\_Бондар.docx | Пояснювальна записка до дипломного проекту. Документ Word. |
| Диплом\_Бондар.pdf | Пояснювальна записка до дипломного проекту в форматі PDF |
| Програма | |
| Program.zip | Архів. Містить коди програми і відкомпільовану програму |
| Презентація | |
| Презентація\_Бондар.pptx | Презентація дипломного проекту |

**ВІДГУК**

**на дипломний проект бакалавра**

**на тему:**

**"Розробка ігрового додатку для адаптації дій гравця з використанням бібліотек RayLib v3.5, OpenMP 2.0 на мові C++"**

**студента групи 122-17-3 Бондаря Андрія Павловича**

Розроблений в дипломному проекті ігровий додаток призначений для демонстрації можливостей машинного навчання та розвитку стратегічного мислення гравця.

Актуальність розробленої системи полягає в створенні програмного продукту, що використовує машинне навчання.

Для вирішення поставлених задач при розробці системи були здійснені наступні дії:

1. Розробка ігрової механіки.

2. Балансування ігрових юнітів.

3. Розробка простого та зрозумілого інтерфейсу гри.

4. Поєднання ігрової механіки та машинного навчання.

Додаток розроблений на мові C++ в середовищі Visual Studio 2019 з використанням бібліотеки RayLib.

Практична значимість створення даного програмного продукту полягає в можливості створювати для гравця нові ігрові ситуації та покращувати його стратегічні навички.

Працездатність представленої програми підтверджена під час розробки та тестування додатку.

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

Тема дипломного проекту безпосередньо пов'язана з об'єктом діяльності бакалавра за напрямом підготовки 050101 Комп’ютерні науки.

Оформлення пояснювальної записки до дипломного проекту виконано відповідно до стандартів на програмну документацію. Недоліком розглянутої роботи є великий вихідний код розробленого додатку.

Дипломний проект виконаний самостійно та заслуговує оцінки «відмінно», а студент Бондар Андрій Павлович присвоєння йому кваліфікації бакалавра за спеціальністю «122 Комп’ютерні науки».

**Керівник дипломного проекту**

**доцент каф. ПЗКС, к.ф-м.н. М.Г. Бердник**

**РЕЦЕНЗІЯ**

**на дипломний проект бакалавра**

**на тему:**

**"Розробка ігрового додатку для адаптації дій гравця з використанням бібліотек RayLib v3.5, OpenMP 2.0 на мові C++"**

**студента групи 122-17-3 Бондаря Андрія Павловича**

Метою дипломного проекту є розроблення ігрового додатка з адаптивним ШІ для демонстрації можливостей методів машинного навчання.

Актуальність розробленого додатку полягає в створенні програмного забезпечення, що застосовує машинне навчання для розвитку стратегічного мислення гравця.

У пояснювальній записці розглянуто необхідність створення і сфера застосування розробленої, виконано постановку завдання, опис вхідних і вихідних даних, розроблено логічну структуру програми, розроблено інформаційне забезпечення системи, наведені загальні відомості про додаток та його функціональне призначення, зазначені використовувані технічні засоби, визначені джерела, використані при розробці.

Запропонований в даному дипломному проекті ігровий додаток розроблений на мові C++ в середовищі програмування Visual Studio 2019 з використанням бібліотеки RayLib.

Вважаю завдання і зміст дипломного проекту відповідним для перевірки ступеня підготовленості Бондаря А.П. за напрямом 050101 Комп’ютерні науки.

Список літератури, наведений в роботі, налічує 13 джерел, що свідчить про достатнє вміння автора працювати з літературою та іншими джерелами інформації.

Якість оформлення дипломного проекту можна визнати доброю, він супроводжується достатньою кількістю малюнків (23). В роботі присутні заключні висновки. Оформлення програмної документації в цілому відповідає ЄСПД.

Працездатність цього додатку підтверджується тестуваннями.

Рівень теоретичної та практичної підготовки автора, логіка і стиль викладу матеріалу в цілому відповідає кваліфікаційним вимогам.

Недоліком розглянутої роботи є невикористання фреймворків для полегшення розробки ігрового додатку.

Дипломний проект Бондаря Андрія Павловича заслуговує оцінки «відмінно» і присудження його автору кваліфікації бакалавра за спеціальністю «122 Комп’ютерні науки».

**Рецензент дипломного проекту**