Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

КУРСОВА РОБОТА
ПОЯСНЮВАЛЬНА ЗАПИСКА
з дисципліни “Об’єктно-орієнтоване програмування”
Ігровий застосунок ”Boulder Dash”

Керівник , ст. викл. Ляпота В. М.
Студент гр. ПЗПІ-23-3 Білоус А. А.

Комісія:

Проф. _____ Бондарєв В. М.
Ст. викл. _____ Черепанова Ю. Ю.
Ст. викл. _____ Ляпота В. М.

Харків – 2025

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра *програмної інженерії*

Рівень вищої освіти *перший (бакалаврський)*

Дисципліна *Об'єктно-орієнтоване програмування*

Спеціальність 1*21 Інженерія програмного забезпечення*

Освітня програма *Програмна інженерія*

Курс *1* Група *ПЗПІ-23-3* Семестр *2*

ЗАВДАННЯ

на текстовий проєкт студента

Білоуса Антона Андрійовича

(Прізвище, Ім'я, По батькові)

1 Тема проєкту:

Ігровий застосунок "Boulder Dash"

2 Термін здачі студентом закінченого проекту: ***"08" - червня - 2024 р.***

3 Вихідні дані до проекту:

Методичні вказівки до виконання курсової роботи

Зміст розрахунково-пояснювальної записки:

Вступ, опис вимог, проектування програми, інструкція користувача, висновки

КАЛЕНДАРНИЙ ПЛАН

№	*Назва етапу*	*Термін виконання*
1	Видача теми, узгодження і затвердження теми	13.02.2024 - 15.03.2024 р.
2	Формулювання вимог до програми	15.02.2024 — 20.02.2024 р.
3	Розробка головної частини гри	20.02.2024 — 4.03.2024 р.
4	Розробка парсера аргументів командної строки	4.03.2024 — 14.03.2024 р.
5	Розробка основної логіки гри	14.03.2024 — 3.04.2024 р.
6	Розробка відображення гри у терміналі та у вікні	3.04.2024 — 23.04.2024 р.
7	Створення пояснювальної записки та документації до застосунку	23.04.2024 — 10.05.2024 р.
8	Захист	03.06.2024 — 08.06.2024 р.

Студент _____

Керівник _____ _____

(Прізвище, Ім’я, По батькові)

« 21 » лютого 2024 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи на тему ”Ігровий застосунок «Boulder Dash»”.

Метою роботи є розробка клону гри «Boulder Dash», яка буде мати базовий геймплей схожий на оригінальний, а також мінімальний редактор рівнів який дозволить створювати нові карти для гри.

Результатом є програма, котра може отимати список назв рівнів та опціональні параметри режиму малювання (графічний, псевдографічинй та консольний), програму яку треба запустити (гра або редактор рівнів), а також швидкість ігрового процесу в мілісекундах.

В процесі розробки використано текстовий редактор Neovim, графічну бібліотеку Rust-SDL2[2] та біблеотеку абстакції терміналу console, мова програмування Rust[3] 2021 року видання.

ЗМІСТ

ВСТУП

Залежність від дофаміну є великою проблемаю сучасності, тому темою цієї курсової роботи є створення мінімального клону Boulder Dash який допоможе людству не померти від нудьги, а автору відточити свої навички програмування.

Boulder Dash — це 2D відеогра-лабіринт-головоломка, випущена в 1984 році компанією First Star Software для 8-бітних комп'ютерів Atari. Її створили канадські розробники Пітер Ліепа та Кріс Грей. Гравець керує персонажем, який збирає скарби, уникаючи падаючого на нього каміння та інших небезпек.

Для реалізації цього проекту була обрана мова програмування Rust. Вона є безсумнівно найкращою мовою сьогодення, яка є неймовірно швидкою та ефективною по використанню пам'яті, і не витрачає час виконня і ресурси процесора збирачом сміття. Rust може працювати з критично важливими до продуктивності сервісами, запускатися на вбудованих пристроях та легко інтегруватися з іншими мовами. Розширена математична система типів та модель власності цієї дивовижної мови гарантує безпеку пам'яті та надзвичайно просте використання мультипоточності, що дозволяє усунути майже всі помилки ще на етапі компіляції.

Ця курсова робота була розроблена для Linux сумісних систем через зручність та наявність нормальних інструментів для програмування і користування застосунком.

1 ОПИС ВИМОГ

1.1 Основний задум гри

Повинен бути створений мінімальний клон аркадної гри Boulder Dash. Застосунок отримує список рівнів які треба пройти і дає змогу це зробити. Рівні це лабінт-печера, в ньому потрібно знайти спосіб зібрати всі кристалики оминаюче падаюче каміння, штовхаючи його вбік та оминати блокуючих ситуацій. Гравець з’являється на першому рівні який є печерою, він має пройти лабіринт що б попасти на наступний, або, якщо наступного рівня немає, закінчити гру перемогою.

1.2 Ігрові об’єкти

В оригінальній версії гри досить багато об’єктів. Через те що у цій роботі робиться лише клон, то мають бути реалізовані тільки ключові елементи. Застосунок має обов’язково містити в собі такі об’єкти:

1.2.1 Гравець

Гравець — об’єкт, яким користувач керує з клавіатури. Може переміщуватись у всіх чотирьох напрямках, ламати пісок та збирати алмази, а також штовхати каміння. Після руху гравець залишає порожнечу позаду себе.

Гравець має мати свій особистий клас який відповідає за поведінку, властивості та відображення цього об’єкту. Об’єкт має мати шляхи для взаємодії з усіма частинами застосунку. Реалізація класу повинна бути в окремому файлі.

1.2.2 Кристал

Кристал — статичний нерухомий об’єкт, який збільшує лічильник очок при знищені, та дає змогу завершити гру перемогою коли лічильник доходить

до максимально значення, тобто було зібранне все каміння.

Кристал має мати свій особистий клас який відповідає за поведінку, властивості та відображення цього об’єкту. Об’єкт має мати шляхи для взаємодії з усіма частинами застосунку. Реалізація класу повинна бути в окремому файлі.

1.2.3 Каміння

Каміння — об’єкт який падає вниз якщо під ним порожнеча або по діагоналі якщо він находиться на краю, тобто клітина збоку та клітину знизу цього боку є порожніми. Коли під цим об’єктом знаходиться Гравець, та Гравець не зламав об’єкт який стримував каміння від падіння (Гравець може встигнути пробіжати під камінням, якщо воно не падало), то гра закінчується програшом. Гравець може рухати камень в бік якщо за цим каменем порожнеча.

Каміння має мати свій особистий клас який відповідає за поведінку, властивості та відображення цього об’єкту. Об’єкт має мати шляхи для взаємодії з усіма частинами застосунку. Реалізація класу повинна бути в окремому файлі.

1.2.4 Стіна

Стіна — статичний нерухомий та незламний об’єкт котрий обмежує рух об’єктів таких як гравець та камень. Цей об’єкт використовується для створення каркасу ріня і самого лабіринту цієї гри, тому вона має робити рамку навколо рівня.

Стіна має мати свій особистий клас який відповідає за поведінку, властивості та відображення цього об’єкту. Об’єкт має мати шляхи для взаємодії з усіма частинами застосунку. Реалізація класу повинна бути в окремому файлі.

1.2.5 Пісок

Пісок — статичний нерухомий об’єкт, який може бути зламаний гравцем. Цей об’єкт слугує для утримання каміння від падіння з ціллю формування більш цікавив та динамічних рівняв, в котрих існуть пастки.

Пісок має мати свій особистий клас який відповідає за поведінку, властивості та відображення цього об’єкту. Об’єкт має мати шляхи для взаємодії з усіма частинами застосунку. Реалізація класу повинна бути в окремому файлі.

1.3 Режими відображення

Гра має мати два режими відображення: консольний та графічний. Обидва режими мають мати спільну кодову базу, точніше мають бути незалежними від внутрішньої логіки гри і використовувати інтерфейси для взаємодії з нею, створюючи невеличке API для відображення логічних об’єктів на екран.

1.3.1 Консольний режим відображення

Консольний режим має мати такий же самий вигляд як і текствоий вигляд рівня у файлі. Цей файл має бути завантажений у пам’ять застосунку, та кожен символ має бути перетворений на відповідний об’єкт у матриці рівня.

Консльний режим має відображати весь інтерфейс гри символами в консолі, а також оброблювати натискання клавіатури та корректно реагувати на сполуки клавіш такі як Ctrl+C для виходу з гри.

1.3.2 Графічний режим відображення

Графічний режим має мати свою теку з текстурами для кожного ігрового об’єкту. За ім’ям кожного об’єкту цей режим завантажує в пам’ять застосунку відповідний файл в теці, піля чого при відображенні у графічному

вікні загружена інформація перетворюється в текстуру.

Графічний режим має створити нове вікно де буде відображатись рівень та поточний статус застосунку. Вікно має автоматично змінювати свій розмір та коректно реагувати на події закриття, спроби зміни розміру вікна та натискання клавіш та їх сполук на клавіатурі.

1.4 Редактор рівнів

Застосунок має містити в собі редактор рінів. Цей режим застосунку дозволяє створювати нові рівні гри та мати змогу їх зберегти для проходження в режимі гри пізніше. Редактор має мати інтуітивно зрозумілий інтерфейс та бути зручним для використання. Розмір рівня який редагується має мати змогу динамічно змінюватись.

Редактор має коректно реагувати на невідомі об’єкти, коректно зберігати рівні у файл та підтримувати всі режими відрисовки. Для цього мають бути реалізовані відповідні інтерфейси та робота з файлами.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вимоги

В ході реалізації проекту крім вимог наведених у першому пункті було також реалізовано наступні пункти:

- незалежний від інших компонентів застосунку парсер командної строки;
- третій режим відображення який знаходиться між консольним та графічним, — псевдографічний;
- запам’товування налаштувань гри між різними рівнями;
- можливість перепройти програний рівень;
- режим паузи у котрому каміння падає тільки поки гравець рухається;
- регулювання швидкості гри яке дозволяє прискорити падіння каміння та рух гравця.

2.2 Проектування

При проектування застосунку було вирішено використовувати принципи SOLID. На протязі розробки ці принципи були кілька разів порушені через нові вимоги які раптово з’являлись, а також через відсутність повноцінного об’єктно орієнтованого програмування в Rust та його специфічну парадигму яку прийшлося пристосовувати до ООП. Після рефакторингу та змін в структурі застосунку, принципи SOLID були дотримні. Поліморфізм та наявність інтерфейсів зробили кодову базу набагато зрозуміліше та менше, даючи можливість дуже просто імплементувати нові речі, такі як об’єкти, режими взаємодії (консольний, псевдографічний та графічний) та режими застосунку(гра та редактор рівів).

2.3 Етап запуску застосунку

Застосунок запускається командою консолі, в якій він отримає аргументи для запуску, після чого ці аргументи передаються до парсерсу. Якщо якогось агрументу не має, то використовується параметр за замовчуванням. Коли було дано помилкове значення, то застосунок не запуститься, але напише який агрумент має некоректне значення. Отримати список всіх аргументів та їх параметрів можна за допомогою аргрумента ”−h або ”−−help”.

Реалізація парсеру аргументів повністю незалежна від інших компонетів гри, що відповідає принципам SOLID. Стуктура яку збирає цей парсер використовується по застосунку тільки для того, щоб компоненти могли ініціалізуватися дивлячись на конфігурацію задану користувачем. Вона передається у функцію run, яка сворює нові об’єкти режиму взаємодії та режиму гри. Після цього фукнція run передає контроль у створені об’єкти які можуть повернути помилку, в цьому випадку функція run поверне цю помилку у функцію main, де ця помилка буде виведена в консольний канал для помилок.

2.4 Інтерфейси застосунку

Цей застосунок має в собі п’ять інтерфейсів, які є самою головною частиною програми та значно полегшують розробку нового функціоналу. Можна виокремити дві групи: інтерфейси взаємодії та інтерфейси ігрових об’єктів.

Інтерфейси взаємодії дозволяють різним режимам взаємодії відображати різні режими застосунку, їх перелік:

– ”Drawable”— інтерфейс для режимів застосунку;

– ”Interaction”— інтерфейс для режимів взаємодії.

Кожний ігровий об’єкт імплементує три інтерфейси:

1) ”Labels”— інтерфейс для позначення об’єкту;

2) ”Properties”— інтерфейс для отримання логічних властивостей об’єкту;
3) ”Behaviour”— інтерфейс для реалізації логіки об’єктів.

2.4.1 Інтерфейс малювання Drawable

Цей інтерфейс має в собі функції для отримання графічних даних режима застосунка режимом взаємодії, а саме:

- get_cursor — повертає позицію курсову. Використовується режимом редактору рівнів;
- get_width — повертає найбільшу ширишу рівня враховуючи ширину тексту статусу. Використовується в графічному режимі відображення;
- get_height — повертає висоту відображаємого вмісту режиму застосунка;
- get_status — повертає строку стану застосунку. Використовується для отримання актуальної інформації;
- get_damaged — повертає набір точок які були змінені і котрі треба перемалювати. Використовується консольним та графічним режимами взаємодії;
- get_objects — повертає матрицю з усіма ігровими об’єктами режима застосунку. Використовується всіма режимами взаємодії;
- get_object — отримує позицію та повертає об’єкт котрий лежить у тій позиції. Використовується у сполуці з функцією get_damaged.

2.4.2 Інтерфейс взаємодії Interaction

Цей інтерфейс має в собі функції для взаємодії режимів застосунку з зовнішнім світом, а саме:

- get_input — функція імлементація якої повинна зчитати ввід з клавіатури та повернути уніфіцированый тип застосунку для вводу Input;

– draw — функція імплементація якої повинна отримати об’єкт імплементуючий інтферфейс малювання Drawable, та відобразити його на екрані.

2.4.3 Інтерфейс позначення Labels

Цей інтерфейс має в собі функції для ідентифікації об’єкту, а саме:

– char — повертає символ який використовується при збереженні об’єкта у файл, а також у консольному режиму взаємодії;
– emoji — повертає емоджи яке використовується у псевдографічному режимі взаємодії;
– name — повертає ім’я об’єкту яке використовується для отримання назви файлу з картинкою даного об’єкту для його відображення у графічному режиму взаємодії.

2.4.4 Інтерфейс властивостей Properties

Цей інтерфейс має в собі функції для отримання логічних властивостей об’єкту, а саме:

– placeholder — об’єкти які просто займають місце у матриці рівня;
– can_be_moved — об’єкти які можуть бути подвинуті гравцем;
– player — об’єкт котрий являється гравцем;
– can_be_broken — об’єкт котрий може бути зламаний гравцем.

2.4.5 Інтерфейс поведінки Behaviour

Цей інтерфейс має в собі функції які імплементують ігрову логіку об’єкту, а саме:

– init — функція яка викликається при створені об’єкту і відправляє запроси до рівня гри;
– on_broken — функція яка викликається при знищенні об’єкту і відправляє запроси до рівня гри;

– tick — функція яка викликається коли оновлюється рівень, і відправляє запроси до рівня гри.

2.5 Об’єкти взаємодії

Ці об’єкти імплементують інтерфейс взаємодії Interaction та відображають об’єкти які імплементають інтерфейс відображення Drawabale. Це дозволяє ефективне використання поліморфізму для розширення застосунку. Інтерфейс взаємодії Interaction дозволяє режимам застосунку запросити дані з клавіатури, а інтерфейс взаємодії Drawable дає змогу відобразити зміни на екран.

2.5.1 Графічний режим (Gui)

Об’єкт взаємодії Gui створює нове вікно за допомогою Rust-SDL2[2] та обролює його події перетворюючи ввід з клавіатури в уніфіцированый тип вводу Input який використовуєтья по самому застосунку. Gui загружає всі файли спрайтів у бінарне дерево, за допомогою которого можна швидко знайти потрібну картинку об’єтку та перетворити її у текстуру.

2.5.2 Псевдографічний режим (Tui)

Об’єкт взаємодії Tui не свторює вікна, він отримує ввід з клавіатури та рисує графічні елементи у консолі за допомогою емоджи. Через те, що емоджи мають відмінну ширину від клітини консолі, реалізація відстеження пошкоджень не була імплементована, вона тільки додасть складності і зробить цей режим менш продуктивним. Щоб зробити відстеження пошкодженнь був створений консольний режим Cli.

2.5.3 Консольний режим (Cli)

Об’єкт взаємодії Cli має дуже зхожу кодову базу до Tui, він навіть використовує функцію для отримання вводу з клавіатури від нього, але

функція відображення графіки відрізняється. У всіх режимах ця функція отримує об’єкт який імплементує інтерфейс для рисовки, він має в собі функцію для отримання змінених клітин. При відображенні інтерфейсу Cli не очищає екран, натомість він малює символ з нових клітин поверх старих, які повертає ця фунція. Ефективніть цього режиму дозволить дуже швидко малювати неймовірно великі рівні.

2.6 Режими застосунку

Режими застосунку дозволяють мати кілька програм в одній. Вони імплементують інтерфейс для графічного відображення Drawable. Цей інтерфейс дозволяє об’єктам взаємодії отримати необхідну їм інформацію від режиму застосунку для того, щоб його відобразити.

2.6.1 Режим гри

Коли запускається режим гри, створюється новий об’єкт гри який створює об’єкти рівней, і для нього запускаться функція run. Після об’єкт гри починає запускати рівні по черзі. Він викликає функції об’єкту взаємодії для обробки подій клавіатури та рисування рівня з статусом. Рівні повертають стан на якому гра була закінчена, якщо була повернена поразка, гра закінчується з повідомленням поразки, в іншому випадку гра продовжується поки не буде пройдений останній рівень і виведе повідомлення перемоги.

2.6.2 Режим редактору рівнів

Цей режим отримує назву файла, та відкриває його. Зміст файлу записується в внутрішні данні редактору, після чого можна можна побачити рівень на екрані, та редагувати його. Цей режим дуже схожий на гру, тільки він не так тісно взаємодіє з об’єктами. Після виходу з цього режиму рівень зберігається, а програма завершує своє виконання.

2.7 Ігрові об’єкти

У кожного об’єкту є власний файл де лежить його імплементація. Вони мають свою власну теку, і їх імпортує модуль який має рівну область видимості для всіх елементів застосунку. Ігрові об’єкти не належать лише до області режиму гри, бо вони активно використовуються іншими компонетами застосунку через інтерфейси.

2.7.1 Гравець (Player)

Цей об’єкт має параметр гравця. Йому передається ввід з клавіатури, після чого він рухається і ламає інші об’єкти. Також об’єкт гравця перевіряє камінь зверху щоб визначити чи може він впасти і закінчити гру програшом.

Згідно з умовами гравець має свій особистий клас який відповідає за поведінку об’єкту, його властивості та відображення. Також цей об’єкт за допомогою своїх інтерфейсів може взаємодіяти з ішними об’єктами використовуючи механізми ігрового рівня.

2.7.2 Кристал (Gem)

Кристал має параметр знищення. При ініціалізації він додає один бал до максимального значення лічильнику, а при знищенні збільшує лічильник. Якщо при знищенні лічильник досягне максимального значення, то рівень гри закінчується перемогою.

Згідно з умовами кристал має свій особистий клас який відповідає за поведінку об’єкту, його властивості та відображення. Також цей об’єкт за допомогою своїх інтерфейсів може взаємодіяти з ішними об’єктами використовуючи механізми ігрового рівня.

2.7.3 Каміння (Rock)

Каміння має параметр руху. Йому передається рівень та його поточні кординати. Камінь дивиться на стан об’єкту знизу, і якщо там порожнеча, то

він падає вниз. Якщо камінь не зміг впасти в низ, то він перевіряє диагоналі, якщо вони вільні, то падіння відбудеться по диагоналі.

Згідно з умовами каміння має свій особистий клас який відповідає за поведінку об’єкту, його властивості та відображення. Також цей об’єкт за допомогою своїх інтерфейсів може взаємодіяти з ішними об’єктами використовуючи механізми ігрового рівня.

2.7.4 Пісок (Dirt)

Пісок має параметр знищення. В нього немає поведінки і він слугує тільки для втримування каміння на місці.

Згідно з умовами пісок має свій особистий клас який відповідає за поведінку об’єкту, його властивості та відображення. Також цей об’єкт за допомогою своїх інтерфейсів може взаємодіяти з ішними об’єктами використовуючи механізми ігрового рівня.

2.7.5 Стіна (Wall)

Стіна не має ані власних властивостей, ані поведінки. Від піску цей об’єкт відрізняє тільки те, що його не можна зламати.

Згідно з умовами стіна має свій особистий клас який відповідає за поведінку об’єкту, його властивості та відображення. Також цей об’єкт за допомогою своїх інтерфейсів може взаємодіяти з ішними об’єктами використовуючи механізми ігрового рівня.

2.7.6 Пороженча (Void)

Порожнеча має властивість заповнювача і не має поведінки. Він створений просто для заповнювання порожніх клітин рівня, і не впливає на рух або існування об’єктів.

Порожнечі не було в умовах, але цей об’єкт теж має свій особистий клас який відповідає за поведінку об’єкту, його властивості та відображення.

За допомогою своїх інтерфейсів він може взаємодіяти з ішними об’єктами використовуючи механізми ігрового рівня.

2.7.7 Невідомий об’єкт (Unknown)

Невідомий об’єкт має взалстивість заповнювача і не має поведінки. Від пороженчі його відрізняє тільки імплементація інтерфейсу позначення Labels, тобто цей елемент малюється по іншому. Він був створений при написанні режиму редактору рівнів для того що б позначати помилкові об’єкти, або об’єкти для яких не існує текстури.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1 Аргументи консолі

Застосунок має такі аргументи командної строки:

– для отримання списку всіх аргументів застосунку ”−−h”або ”−−help”;
– обрати режим інтеракції можна за допомогою опції”−−m”або ”−−mode”і написати ”gui ”tui”або ”cli”через пробіл після неї;
– обрати підпрограму можна за допомогою опції ”−−r”або ”−−run”і написав ”game”або ”editor”через пробіл після неї;
– поставити паузу в грі при запуску можна за допомогою опції ”−−p”або ”−−pause”;
– обрати розмір елементів у пікселях в графічному режимі можна за допомогою опції ”−−s”або ”−−size”та число через пробіл;
– обрати затримку між кадрами в мілісекундах для гри можна за допомогою опції ”−−d”або ”−−delay”та число через пробіл.

Всі інші аргументи сприймаються як назви файлів рівнів.

3.2 Керування застосунком

Керування застосунком здійснюється з клавіатури виключно. Таке рішення було прийнято через багато факторів пов’язаних з легкістю імлементації та використання застосунка.

3.2.1 Керування грою

– рух здійснюється за допомогою стрілочек клавіатури;
– перезапустити рівень можна за допомогою клавіші R;
– пауза або її зняття виконоуються натисканням пробіла;
– зменшення та збільшення затримки між кадрами здійснюється комою і крапою відповідно;
– вийти з гри можна за допомогою клавіши Q, або натиснув Ctrl+C.

3.2.2 Керування редактором рівнів

– рух здійснюється за допомогою стрілочек клавіатури;
– перезагрузити рівень з файлу можна за допомогою клавіші R;
– опустити або підняти перо виконоується натисканням пробіла;
– обирати об’єкт можна комою та крапкою, вони обераються попередній і наступний об’єкт відповідно;
– вийти та зберегти рівень можна за допомогою клавіши Q, або натиснув сполуку Ctrl+C.

3.3 Позначення об’єктів

В консольному режимі взаємодії та у файлі рівня ігрові об’єкти позначаються однаковими симоволами. Ось позначення кожного ігрового об’єкту в дужках:

– гравець: ”p”;
– кристал: ”+”;
– каміння: ”O”;
– пісок: ”*”;
– стіна: ”#”;
– порожнеча: ” ”;
– невідомий об’єкт: ”?”.

Якщо в рівні видно невідомий об’єкт, значить в файлі рівня є символи які гра не може інтепретувати. Вирішити це можна використав режим редактору, або, якщо рівень вже запущений в грі, власноруч відредагувати файл у текствому редакторі та натиснути клавішу R для перезагрузки рівня.

ВИСНОВКИ

Було створено застосунок який має три режими вводу/виводу, гру та редактор рівнів. За допомогою інтерфейсів та поліморфізму можна дуже легко додати ще один режим вводу/виводу або розширити програму як це було зроблено з редактором рівнів. Застосунок дуже швидко працює та має такі оптимізації як відстеження пошкоджених клітинок та буферизацію графічних елементів.

В ході виконання цієї роботи я отримав багато практики, знань і навичок програмування на Rust, і став комфортно працювати з великою кількістю файлів, системою контролю версій git і пакетним менеджером cargo.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Grafiati: Оформити списки використаних джерел онлайн. Grafiati: Оформити списки використаних джерел онлайн. URL: https://www.grafiati.com/uk/ (дата звернення: 09.05.2024).

2. GitHub - Rust-SDL2/rust-sdl2: SDL2 bindings for Rust. GitHub. URL: https://github.com/Rust-SDL2/rust-sdl2 (дата звернення: 2.06.2024).

3. Rust Programming Language. Rust Programming Language. URL: https://www.rust-lang.org/ (дата звернення: 10.05.2024).

ДОДАТОК А

Код програми

Актуальну версію коду можна занйти за посиланням

`https://github.com/oxid8/nure/tree/main/2/coursework/src`