**АКАДЕМИЯ ГОСУДАРСТВЕННОГО УПРАВЛЕНИЯ**

**ПРИ ПРЕЗИДЕНТЕ АЗЕРБАЙДЖАНСКОЙ РЕСПУБЛИКИ**

|  |  |
| --- | --- |
| **Факультет:** | Административное управление |
| **Кафедра:** | Управление интеллектуальными системами |
| **Специальность:** | Компьютерные науки |
| **Группа:** | K192 |
| **Отдел:** | Очное |

|  |
| --- |
| **В Ы П У С К Н А Я Р А Б О Т А**  на тему:  **“Игра “Pac-man” на языке программирования С++ с** |
| **использованием библиотеки SFML”** |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент: |  | | | |
|  | (подпись) | | фамилия, имя, отчество | |
| Руководитель: |  | | | |
|  | (подпись) | | научная степень, фамилия, имя, отчество | |
| Заведующий кафедры: | |  | | |
| (подпись) | | | | научная степень, фамилия, имя, отчество | |

**БАКУ - 2023**

**ОГЛАВЛЕНИЕ**

**ВВЕДЕНИЕ**………………………………………………………..…….........**3**

**ГЛАВА I. РАССМОТРЕНИЕ ТЕОРЕТИЧЕСКИХ АСПЕКТОВ** ……..**5**

1.1 Язык программирования С++………………………………………....5

1.2 Библиотека SFML. Подключение и настройка…………..…….…….6

1.3 История игры “Pac-man”……………………………………………...15

**ГЛАВА II. РАЗРАБОТКА КОДА ИГРЫ**………...……………………...**17**

2.1 Основные компоненты игры…………………………………………17

2.2 Создание “игрока и противников”…………………………………...25

2.3 Объединение компонентов и функция Main………………………..35

**ЗАКЛЮЧЕНИЕ**…………………………………………………………….**44**

**СПИСОК ЛИТЕРАТУРЫ**……………………………………………...…**46**

**ВВЕДЕНИЕ**

**Актуальность дипломной работы.** Компьютерные игры - это неотъемлемая часть современной культуры. Они стали настолько популярными, что сегодня их создают не только для развлечения, но и для обучения, терапии и других целей. Игры могут быть разнообразными: от простых игр на мобильных устройствах до сложных симуляторов и стратегий на компьютере. Одной из самых популярных и любимых игр всех времен является игра "Pac-man", созданная в 1980 году компанией Namco, ныне купленная американской компанией Bandai и переименованная в Bandai Namco.

Цель данной дипломной работы заключается в создании собственной версии игры "Pac-man" на языке программирования C++ с использованием библиотеки SFML (Simple and Fast Multimedia Library). Это будет полноценная компьютерная игра. Сфера компьютерных игр развивается быстро, и на сегодняшний день уже есть множество версий этой игры, но наша версия будет классической.

В этой дипломной работе мы рассмотрим весь процесс создания игры "Pac-man" на языке программирования C++ с использованием библиотеки SFML. Мы начнем с анализа игровых механик и компонентов игры, продвинемся к созданию графического интерфейса, игровой логики и искусственного интеллекта привидений.

Создание игр - это творческий и сложный процесс, который требует не только знаний в области программирования, но и художественного вкуса и понимания принципов дизайна. Мы будем уделять внимание каждому аспекту разработки игры "Pac-man", чтобы получить качественный и интересный продукт.

В ходе работы мы также рассмотрим теоретические аспекты создания игр, включая алгоритмы и структуры данных, используемые в играх, а также принципы объектно-ориентированного программирования и паттерны проектирования. Эти знания помогут нам создать более эффективный и легко расширяемый код.

Данная дипломная работа имеет теоретическую и практическую значимость. Создание игры "Pac-man" на языке программирования C++ с использованием библиотеки SFML является примером применения технологий программирования в области разработки игр.

Разработка игр является сложным и многопроцессным процессом, который требует знаний в области программирования, алгоритмов, графики, звуков и т.д. Эта работа представляет собой пример реализации игры, которая может быть использована в качестве образца для создания других игр.

Итак, cоздание игры "Pac-man" на языке программирования C++ с использованием библиотеки SFML является интересным и сложным проектом, который поможет улучшить навыки программирования и расширить знания в области разработки игр.

**Структура дипломной работы** состоит из титульного листа, введения, двух глав, заключения и списка литературы, что в общем составило 46 страниц.

**ГЛАВА I. РАССМОТРЕНИЕ ТЕОРЕТИЧЕСКИХ АСПЕКТОВ.**

**1.1 Язык программирования С++**

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

Своими корнями он уходит в язык Си, который был разработан в 1969—1973 годах в компании Bell Labs программистом Деннисом Ритчи (Dennis Ritchie). В начале 1980-х годов датский программист Бьерн Страуструп (Bjarne Stroustrup), который в то время работал в компании Bell Labs, разработал С++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

Впоследствии новый язык стал набирать популярность. В него были добавлены новые возможности, которые делали его не просто дополнением к Си, а совершенно новым языком программирования. В итоге "Си с классами" был переименован в С++. И с тех по оба языка стали развиваться независимо друг от друга.

С++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко С++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на С++. Но только системным программированием применение данного языка не ограничивается. С++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ. Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход мобильное направление, где С++ тоже нашел свое применение. И даже в веб-разработке также можно использовать С++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем С++ - язык широкого пользования, на котором можно создавать практически любые виды программ.

С++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на С++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на С++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на С++ в приложения под эти платформы.

В отличие от Си язык C++ позволяет писать приложения в объектно-ориентированном стиле, представляя программу как совокупность взаимодействующих между собой классов и объектов. Что упрощает создание крупных приложений.

**Философия С++**

Смысл философии языка C++ можно определить выражением «доверять программисту». Например, компилятор не будет вам мешать сделать что-то новое, что имеет смысл, но также не будет мешать вам сделать что-то такое, что может привести к сбою. Это одна из главных причин, почему так важно знать то, что вы не должны делать, как и то, что вы должны делать, создавая программы на языке С++.

**1.2 Библиотека SFML. Подключение и настройка**

SFML (Simple and Fast Multimedia Library) является популярной библиотекой для разработки игр и мультимедийных приложений на C++. С помощью SFML разработчики могут создавать 2D и 3D игры, включая платформеры, шутеры, RPG, стратегические игры и многое другое. В этой статье мы рассмотрим основные возможности SFML для создания игр.

**Графика**

SFML обеспечивает мощные возможности для работы с графикой, что позволяет создавать высококачественную графику для игр. Библиотека поддерживает отображение спрайтов, текстур, примитивов и других объектов. SFML также позволяет работать с шейдерами, что дает больше контроля над тем, как графика отображается в игре.

SFML поддерживает множество форматов файлов, включая PNG, JPEG, BMP, GIF, TGA и многие другие. Это делает ее удобной для работы с ресурсами игры, такими как текстуры, изображения, звуки и музыка.

**Звук**

SFML также предоставляет возможности для работы со звуком. Библиотека позволяет проигрывать звуковые эффекты и музыку в игре. SFML поддерживает множество форматов звуковых файлов, включая WAV, OGG, FLAC и многие другие.

**Ввод**

SFML позволяет получать ввод данных от игроков, такие как мышь, клавиатура и джойстики. Библиотека также обеспечивает возможности для работы с тачскринами и гироскопами на мобильных устройствах. Это позволяет создавать игры с различным управлением, что делает их более доступными и интересными для игроков.

**Сеть**

SFML также предоставляет возможности для работы с сетью. Это позволяет создавать многопользовательские игры, где игроки могут играть друг против друга или совместно выполнять задания в игре. Библиотека поддерживает различные протоколы сетевого взаимодействия, включая TCP, UDP и HTTP.

**Кроссплатформенность**

SFML является кроссплатформенной библиотекой, что означает, что она работает на различных операционных системах, включая Windows, Linux, macOS и мобильных платформах iOS и Android. Это делает ее удобной для разработки игр, которые могут быть запущены на многих устройствах, без необходимости переписывать код для каждой платформы.

**Простота использования**

SFML обеспечивает простоту использования, что делает ее идеальной для начинающих разработчиков игр. Библиотека имеет понятный интерфейс и хорошо задокументирована, что позволяет быстро освоить ее основы. Более того, SFML имеет активное сообщество разработчиков, которое помогает новичкам советами и решением проблем.

Для создания игр с помощью SFML необходимо иметь знания языка программирования C++ и основы работы с графикой, звуком и вводом данных. Однако, библиотека предоставляет удобный набор классов и функций, что упрощает работу с графикой и звуком.

SFML также предоставляет ряд примеров и учебных материалов, которые помогут разработчикам понять, как работает библиотека и как ее использовать для создания игр.

SFML является мощной и удобной библиотекой для создания игр и мультимедийных приложений на C++. Она обеспечивает возможности для работы с графикой, звуком, вводом данных и сетью, что позволяет создавать разнообразные игры на различных платформах. Библиотека также обладает простотой использования и хорошо задокументирована, что делает ее идеальной для начинающих разработчиков игр.

**Установка и подключение.**

Для загрузки SFML необходимо скачать соответствующие файлы библиотеки с официального сайта, https://www.sfml-dev.org/. При этом необходимо выбрать версию библиотеки, подходящую для вашей операционной системы и компилятора, в моем случае это самая последняя версия библиотека для 64-битной системы Windows 10.

После скачивания файлов необходимо установить библиотеку в систему, чтобы ее можно было использовать при разработке программ. Для этого необходимо выполнить инструкции по установке, приведенные на официальном сайте.

Перед тем, как использовать библиотеку SFNL, необходимо ее настроить в проекте. Для этого мы должны выполнить следующие шаги:

1. Создать пустой проект в Visual Studio:

См. Вложения Рис. 1.1 - Пустой проект.

1. В Solution Explorer выбираем наш проект:

Рис. 1.2 - Solution Explorer.

1. В меню проекта выбираем Add, затем New Item…:

См. Вложения Рис. 1.3 - New Item.

1. Добавляем файл Pac.cpp:

См. Вложения Рис. 1.4 - Pac.cpp.

Обращаем внимание на платформу решения, она должна совпадать с разрядностью скачанной библиотеки SFML. В нашем случае мы настраиваем 64-х разрядную библиотеку. Если же мы настраивали бы 32-х разрядную систему, тогда нужно было бы указать x86.

1. Выбираем меню Project и меню Pac-man Properties:

|  |
| --- |
|  |
| **Рис. 1.5 - Proporties** |

6. В свойствах проекта добавляем путь к заголовочным файлам SFML, то есть к файлу include:

C/C++ - General - Additional Include Directories

|  |
| --- |
|  |
| **Рис. 1.6 - include** |

7. Далее указываем компоновщику, где найти библиотеки SFML, то есть файл lib:

Linker – General – Additional Library Directories

|  |
| --- |
|  |
| **Рис. 1.7 - lib** |

8. Связываем наше приложение с библиотеками SFML файлами \*.lib:

Linker – Input – Additional Dependencies.

Копируем название библиотек в поле окна дополнительные зависимости

|  |
| --- |
|  |
| **Рис. 1.8 - Связывание** |

9. Устанавливаем отображения графического окна без консоли:

Для этого переходим в Linker, далее в System, затем в SubSystem выбираем Windows (/SUBSYSTEM:WINDOWS).

|  |
| --- |
|  |
| **Рис. 1.9 - SubSystem** |

10. Создаём точку входа: mainCRTStartup:

**mainCRTStartup** - это стандартная функция запуска, которая предоставляется компилятором для программ на языке C/C++, написанных для операционных систем семейства Windows.

Эта функция инициализирует среду выполнения C/C++, перед тем, как вызвать функцию **main()** нашей программы. Она выполняет такие задачи, как настройка стандартных потоков ввода/вывода, передача аргументов командной строки в функцию main(), вызов конструкторов глобальных объектов, а также управление памятью и исключениями.

Кроме того, mainCRTStartupможет быть использована для изменения стандартного поведения программы при запуске. Например, можно изменить настройки консоли, создать лог-файл, настроить режим отладки или изменить внутренние параметры программы.

В целом, mainCRTStartup необходима для успешного запуска и работы программ на языке C/C++ в операционных системах Windows.

Для создания точки входа мы переходим в **Linker**, далее в **Advanced**, затем в **Entry Point** мы вписываем значение **mainCRTStartup**

|  |
| --- |
|  |
| **Рис. 1.10 - Entry Point** |

11. Копируем все файлы из папки bin библиотеки SFML и вставляем в наш проект.

После успешной установки библиотеки, ее можно использовать в проекте на языке программирования C++ для создания игры Pac-Man.

**1.3 История игры “Pac-man”**

В далеком 1979 году появилась компьютерная игра Pac-man, перевернувшая представление об аркадах. Популярность лабиринтов пришла именно благодаря Тору Иватани, создателю игры. Считается, что вдохновило его поедание пиццы. Оказывается, обычный для каждого из нас процесс может привести к появлению гениальной идеи. Пицца без одного кусочка и правда выглядит в точности как Пакман. Сам Тору позже рассказал, что эта теория отчасти правда. Окончательный вид герою игры создатель придал, округлив иероглиф «кути», означающий рот или поедание чего-либо. Официальным годом появления игры считается 1980.

**Что такое Pacman**

Если вы никогда не играли в Pacman, стоит обязательно попробовать. Несмотря на то, что игра пришла из 80-х, она претерпела изменения и дополнения. При этом даже оригинальный вариант затянет не на один час.

Разработчик японской компании Namco Тору Иватани целых восемнадцать месяцев трудился над игрой. Он пытался внести интересные элементы, чтобы путешествие по лабиринту привлекло не только подростков, но и людей постарше. Изначально название было Pakku-Man. В Японии появление игры осталось практически незамеченным, так как на рынке уже существовали Space Invaders или Defender. Но через год американская компания Midway решает распространить игру у себя на родине. Так появилось название Puck Man. Но Молодежь США тут же заменила название на неприличное слово. Поэтому было решено сменить название еще раз, появилось ныне всем известное Pac-Man.

Суть Pacman в прохождении лабиринта и сборе точек или небольших шариков. При этом, нельзя нарываться на привидений, периодически появляющихся на пути.

**Как играть в Pacman**

Основная задача игрока – собрать все точки, появляющиеся в лабиринте. Во время беспомощности привидений можно поедать и их. Они не исчезают навсегда, но, по крайней мере, возвращаются в центр и какое-то время не мешают. Враги находятся в центре экрана в начале игры, но потом начинают расползаться по лабиринту.

Преимуществом для «колобка» является переход из одного угла лабиринта в другой. Если выйти из лабиринта направо или налево за границу игры, то вы появитесь с другой стороны. Плюс в том, что призраки замедляются при таком переходе, а вы нет. Помимо этого на разных уровнях появляются дополнительные бонусы в виде фруктов. Стоит помнить, что у каждого из привидений есть свои навыки передвижения и свои очки за их поедание.

После такого грандиозного успеха создатель Иватани участвовал еще в некоторых проектах, среди которых Pac-Land и Pac-Mania. Тору признавался, что, несмотря на появление других игр, “Пакман” так и остался его любимой.

Влияние игры

Из-за бешеной популярности не могли не появиться песни, фильмы и сериалы. Pac-Man Fever – музыкальный альбом с треками, где использовались названия популярных игр в 80-е. Также вышел сериал, но про него быстро забыли.

Отсылки к “Пакману” есть во многих видеоиграх. В Wolfenstein 3D есть специальный секретный уровень, повторяющий знаменитую игру. Ultima Underworld сделала то же самое с одним из своих уровней.

**ГЛАВА II. РАЗРАБОТКА КОДА ИГРЫ**

**2.1 Основные компоненты игры**

Основные компоненты компьютерных игр являются существенными элементами игрового процесса, предназначенными для обеспечения взаимодействия между игроком и игровым миром. В общем случае, основными компонентами игры являются игровое поле, игровые объекты, интерфейс, звуковые эффекты и музыка.

Игровое поле представляет собой пространство, в котором игрок перемещается и взаимодействует с игровыми объектами. Оно может иметь различные размеры и формы, в зависимости от конкретной игры. Игровые объекты представляют собой персонажей, предметы и препятствия, которые игрок встречает на своем пути и с которыми взаимодействует. Интерфейс отображает информацию об игровом процессе, такую как количество жизней, очки, уровень и т.д. Звуковые эффекты и музыка воспроизводятся во время игры, чтобы создать соответствующую атмосферу и усилить эмоциональное воздействие на игрока.

В разработке компьютерных игр основные компоненты реализуются с использованием программных абстракций, таких как классы и объекты. Они представляют собой абстрактные сущности, которые описывают поведение игровых объектов и взаимодействия между компонентами. Применение подобных концепций позволяет создавать сложные игровые миры, реалистичные персонажи и интересный игровой процесс.

Нам необходимо создать карту, где будет происходить прохождение игры. Но до этого мы создадим эдакий глобальный Header файл, в котором будет необходимая для игры информация. Мы ее пишем отдельно, да и многие строчки кода тоже будем писать отдельно, во избежание путаницы в коде. И так:

|  |
| --- |
| #pragma once  #include <SFML/Graphics.hpp>  using namespace sf;  const int H = 21;  const int W = 19;  const int Split = 25;  int q = 0;  int RestartTime = 5000;  bool Life = true; |
| **Код 2.1 - Global** |

Где #pragma once - это директива препроцессора в языке программирования C++, которая используется для предотвращения многократной компиляции одного и того же заголовочного файла.

Когда компилятор встречает директиву #pragma once в заголовочном файле, он создает маркер, который указывает на то, что файл был уже включен в процесс компиляции. При следующей попытке включения этого же файла компилятор будет использовать маркер и пропустит повторное включение файла, что позволяет избежать ошибок компиляции, связанных с множественным определением одного и того же класса, функции или переменной.

#include <SFML/Graphics.hpp> - это директива препроцессора языка программирования C++, которая позволяет включить заголовочный файл SFML для работы с графикой. SFML (Simple and Fast Multimedia Library) - это кроссплатформенная библиотека, которая предоставляет инструменты для создания мультимедийных приложений, включая возможности для работы с графикой, звуком, сетью и вводом. Заголовочный файл SFML/Graphics.hpp содержит определения классов и функций для работы с графическими объектами, такими как спрайты, текстуры, окна и другие. Он является одним из основных заголовочных файлов SFML и должен быть включен в любое приложение, которое использует графический движок SFML.

namespace sf - это пространством имён в языке программирования C++, которое используется в библиотеке SFML. Пространство имен sf содержит все классы, функции и переменные, которые используются для работы с графическим интерфейсом, звуковыми эффектами, сетью и другими мультимедийными возможностями, предоставляемыми этой библиотекой.

Конструкция using namespace sf; в коде означает, что все идентификаторы (например, классы, функции и переменные), определенные в пространстве имён sf, могут использоваться в данном файле без явного указания этого пространства имён. Таким образом, при вызове класса sf::RenderWindow, который мы будем вызывать в дальнейшем, можно использовать сокращенную форму RenderWindow.

const int H и const int W – это две переменные, которые в данном контексте предназначены для определения размеров игрового поля, на котором будет развиваться игровой процесс. Они устанавливают количество строк и столбцов, соответственно, и определяют общее количество ячеек на поле. Эта информация может использоваться для генерации игровых объектов, расчета их координат на поле и определения границ поля, которые могут оказывать влияние на поведение игровых объектов и персонажей. Использование констант для задания размеров игрового поля позволяет сделать код более читаемым и удобным для поддержки и дальнейшей разработки.

const int Split является сплитом, в котором мы будем рисовать элементы игры. В играх сплиты (англ. "splits") - это разделения игрового процесса на отдельные сегменты для измерения времени прохождения каждого сегмента. Обычно это используется в скоростных играх, таких как гонки, платформеры или аркады, коим является наш проект, Pac-man, чтобы игроки могли оптимизировать свои результаты на каждом участке игры.

В представленном коде, Split означает размер тайла (игровой плитки) в пикселях, который используется для определения размера игрового поля и размеров игровых объектов. В данном случае, Split может использоваться как сплит для измерения времени перемещения игровых объектов через определенное количество пикселей. Значение 25 – это идеальное значение для нашего проекта.

Переменная int q отвечает за счет игрока. Счет в игре - это числовой показатель, который отражает количество очков, заработанных игроком за время игры. Он является одним из основных элементов игрового процесса и представляет собой метрику, по которой игрок может оценить свой успех. Счет обновляется при съедании игровых объектов и увеличивается на определенное количество очков за каждый съеденный объект. В игре Pac-man счет может быть использован для достижения игровых целей, таких как прохождение уровня или победа в игре, а также для мотивации игрока продолжать игру и улучшать свой результат.

Переменная int RestartTime в нашем проекте представляет собой целочисленное значение, которое используется для определения времени, через которое игра должна быть перезапущена после того, как игрок проиграл. В данном случае значение переменной равно 5000, что означает, что игра будет перезапущена через 5000 миллисекунд (или 5 секунд) после того, как игрок проиграет. Это значение может быть изменено, если требуется изменить время, через которое игра должна быть перезапущена после проигрыша.

Булевая переменная bool Life – это переменная отвечающая за жизнь игрока. Жизнь представляет собой количество попыток, которые игрок имеет для прохождения уровня. Игрок теряет жизнь, если его персонаж сталкивается с привидением. Когда игрок теряет жизнь, игра завершается и начинается сначала. Таким образом, жизнь в игре "Pac-man" является важным игровым ресурсом, который игрок должен использовать с умом, чтобы успешно завершить уровень.

После создания переменных для нашей игры, нам необходимо создать карту, игровое поле, где будет передвигаться сам Пакман.

Создаем ещё один Header файл для карты и назовем её Map. После создания файла Map.h, мы можем писать код для нашего поля.

|  |
| --- |
| #include <SFML/Graphics.hpp>  #include "Global.h"  using namespace sf;  String Map[H] = {  "AAAAAAAAAAAAAAAAAAA",  "A1 A 2A",  "A AA AAA A AAA AA A",  "A A A",  "A AA A AAAAA A AA A",  "A A A A A",  "AAAA AAA A AAA AAAA",  "A A A A A A",  "AA A A AA AA A A AA",  " A A ",  "AA A A AAAAA A A AA",  "A A A A A A",  "AAAA A AAAAA A AAAA",  "A A A",  "A AA AAA A AAA AA A",  "A A C A A",  "AA A A AAAAA A A AA",  "A A A A A",  "A AAAAAA A AAAAAA A",  "A3 4A",  "AAAAAAAAAAAAAAAAAAA"}; |
| **Код 2.2 - Мар** |

**#include "Global.h"** является заголовочным файлом, разработанным специально для данного проекта, и содержит необходимую информацию, которая используется в коде программы. Заголовочные файлы являются важными элементами в разработке программного обеспечения, поскольку они позволяют разработчикам использовать код повторно и делать его более организованным и понятным для сопровождения и управления. В конкретном случае, Global.h содержит определения переменных, констант, которые используются в проекте, и может быть включен в различные файлы исходного кода для обеспечения доступа к этим ресурсам.

**sf::String**, но в нашем случае, благодаря using namespace sf, мы можем писать просто String, - представляет собой класс, реализованный в библиотеке SFML, который предоставляет функциональность для работы со строковыми данными в контексте графического приложения.

Как и в большинстве языков программирования, строка в SFML представляет набор символов, которые могут быть использованы для отображения текста на экране или для манипуляций с данными в процессе выполнения программы.

Класс sf::String включает в себя методы для работы с символами в строке, а также для получения информации о строке, такой как ее длина и кодировка. Он также обеспечивает поддержку многих языков и символьных наборов, включая Unicode.

Благодаря использованию класса sf::String, разработчики могут создавать и манипулировать строковыми данными в своих графических приложениях с помощью простого и эффективного API.

**Map** – название объекта класса String, а Н – это размер нашей карты, который мы прописали в Global.h.

Переменная Map является массивом строк, где каждый символ в строке обозначает определенный элемент на карте игры. Символ "A" обозначает границы карты и стены, выход за которые невозможен, символ "C" обозначает персонажа пакмана, пустые ячейки обозначают шарики, которые пакману необходимо съесть, а числа от 1 до 4 обозначают изначальное расположение противников. Таким образом, массив Map описывает игровое поле, на котором пакман может перемещаться и взаимодействовать с другими объектами.

Итак, чтобы прорисовать нашу карту, мы будем использовать .png изображение title, которое будем использовать в главное методе main.

|  |
| --- |
|  |
| **Рис. 2.1 - Элементы игры.** |

Для прорисовки карты мы будем использовать две плитки:

* Синий квадрат, который будет чертить границы карты;
* Маленький круг – точки лабиринта, который нам необходимо собирать по мере продвижения вперед. Мы обвели его черным цветом для того, чтобы было видно сам круг, в игре он будет полностью белым.

Далее нам необходимо отображать счет игрока в окне игры. За это будет отвечать класс Score.

В конструкторе класса инициализируется массив спрайтов sec, который содержит три спрайта. Каждый спрайт устанавливается на позицию, зависящую от значения H, константы, определённой вне этого класса, и параметра Split. Также каждому спрайту устанавливается текстура из параметра image и начальный прямоугольник текстуры с помощью функции setTextureRect.

В методе Update() определяется, какое значение отображать на спрайтах счёта. Значение счёта хранится в переменной c, а его цифры хранятся в массиве s. Также массив vid содержит флаги, которые указывают, какие спрайты должны быть отображены.

Спрайт (англ. "sprite") - это объект графического изображения в компьютерных играх и других программах, который может быть перемещен по экрану и взаимодействовать с другими объектами. Он представляет собой графический элемент, который может содержать одну или несколько изображений (кадров), и используется для отображения анимации, персонажей, предметов и других элементов в играх.

В программировании спрайт обычно создается из текстуры (изображения), которая может быть загружена из файла или создана динамически. Кроме того, спрайт может иметь свои свойства, такие как позиция на экране, размеры, угол поворота, прозрачность и другие атрибуты, которые могут изменяться в зависимости от разных условий и действий пользователя.

|  |
| --- |
| class Score {  public:  Sprite sec[3];  int s[3], c;  bool vid[3];  Score(Texture& image) {  for (int i = 0; i < 3; i++) {  sec[i].setTexture(image);  sec[i].setPosition(10 + 22 \* i, H \* Split + 10);  sec[i].setTextureRect(IntRect(0, 0, 22, 45));  s[i] = 0;  c = 0;  }  }  void Update() {  if (c < 10) {  s[0] = c;  vid[0] = true;  vid[1] = false;  vid[2] = false;  }  else if (c >= 10 && c < 100) {  s[0] = c / 10;  s[1] = c % 10;  vid[0] = true;  vid[1] = true;  vid[2] = false;  }  else if (c >= 100 && c < 1000) {  s[0] = c / 100;  s[1] = (c / 10) % 10;  s[2] = (c % 100) % 10;  vid[0] = true;  vid[1] = true;  vid[2] = true;  }  for (int i = 0; i < 3; i++)  sec[i].setTextureRect(IntRect(22 \* s[i], 0, 22, 45));  }  }; |
| **Код 2.3 - Score** |

В зависимости от значения c, метод Update() изменяет значения элементов массива s и флагов в массиве vid, чтобы отобразить текущее значение счёта. Затем каждому спрайту sec[i] устанавливается прямоугольник текстуры, определяющий, какую цифру нужно отобразить.

Таким образом, класс Score позволяет отображать текущий счёт игрока в игре с помощью трёх спрайтов.

**2.2 Создание “игрока и противников”**

После того, как мы создали основные элементы и задали им значения, а также создали карту, переходим к созданию игрока и противников. Для этого мы используем классы Player и Enemy.

Игрок и противники имеют разные параметры: координаты, анимации, направление движения. Все эти параметры задаются внутри метода классов.

Для начала создадим класс Player, в котором мы пропишем координаты нашего пакмана, а также метод для их обновления на каждой итерации основного цикла игры.

|  |
| --- |
| class Player {  public:  float frame = 0;  int x = 9, y = 15;  int NewX = 0, NewY = 0;  int Rotate = 1, Time = 0;  bool Teleport = false;  void Update() {  if (Teleport) {  Map[y][x] = ' ';  x = 9;  y = 15;  Map[y][x] = 'C';  NewX = x;  NewY = y;  Teleport = false;  } |
| **Код 2.4 – Update Player** |

Переменная frame в данном коде является счётчиком кадров анимации игрока. Её значение увеличивается на 0.01 на каждой итерации цикла и используется для определения номера текущего кадра из спрайтового листа игрока. Если frame превышает 5, то значение уменьшается на 5, чтобы анимация начиналась с начала. Этот механизм обновления кадров используется для создания эффекта плавной анимации движения игрока.

В данном коде x и y представляют собой координаты текущего положения игрока на игровой карте, а NewX и NewY - координаты, в которые игрок может переместиться на следующем шаге. Изменение значений x и y происходит только после того, как игрок переместился в позицию, заданную значениями NewX и NewY, и эта позиция была подтверждена как возможная.

В контексте нашего кода, Rotate является переменной, которая хранит текущее направление движения игрока в игре "Pac-man". Значение переменной Rotate может быть 1, 2, 3 или 4, что соответствует направлениям вправо, влево, вверх и вниз соответственно. Эта переменная используется в функции Update для проверки возможности движения игрока в заданном направлении.

Переменная Time используется для подсчета времени, которое прошло с последнего изменения позиции игрока на карте. Она увеличивается на каждой итерации цикла в методе Update(), и когда её значение превышает 300 (это значение задается условием if), то меняется позиция игрока на карте в соответствии с текущим значением переменной Rotate.

Переменная Restart является логической переменной типа bool, используемой для перезапуска игры. Если значение Restart равно true, то игра перезапускается.

В данном коде, если Restart равно true, то выполняется следующее:

1. Очистка текущей позиции игрока на карте: элемент массива Map с индексами x и y устанавливается в символ пробела.
2. Затем игрок устанавливается на исходную позицию в середине карты: x и y устанавливаются в 9 и 15 соответственно, а элемент массива Map с индексами x и y устанавливается в символ C.
3. Переменная Restart устанавливается обратно в значение false.

Это происходит, когда игрок проигрывает или выигрывает.

Метод Update в данном коде отвечает за обновление состояния объекта класса Player в зависимости от текущего состояния игры. В частности, метод Update переключает фреймы анимации игрока, обрабатывает перемещение игрока по карте в зависимости от выбранного направления Rotate и времени Time, проверяет возможность перемещения игрока по карте на новое положение NewX и NewY, а также проверяет и обрабатывает столкновения игрока со стенами и точками на карте.

**Первый** **if** проверяет, установлен ли флаг **Teleport** в true. Если да, то происходит перемещение игрока на начальную позицию, заданную координатами (9, 15) на карте. Затем флаг Teleport устанавливается в false.

Данное условие применяется тогда, когда мы выиграли или же наоборот, проиграли. Таким образом, игра начинается заново.

|  |
| --- |
| else {  frame += 0.01;  if (frame > 5)  frame -= 5;  Time++;  if (Time >= 300)  switch (Rotate) {  case 1:  if (Map[y][NewX + 1] != 'A')  NewX += 1;  break;  case 2:  if (Map[y][NewX - 1] != 'A')  NewX -= 1;  break;  case 3:  if (Map[NewY - 1][x] != 'A')  NewY -= 1;  break;  case 4:  if (Map[NewY + 1][x] != 'A')  NewY += 1;  break;  }  Time = 0;  } |
| **Код 2.5 – Блок else** |

В случае, если условия первого if не выполнены, мы переходим в блок else, где проверяем, находится ли игрок в движении. Если игрок находится в движении, то переменная frame увеличивается на 0.01. Эта переменная используется для анимации движения игрока. Если frame больше 5, то переменная уменьшается на 5.

После этого происходит инкремент переменной Time. Если Time больше или равна 300, то происходит переключение игрока в новое положение, заданное переменными NewX и NewY. Оператор switch используется для определения движения игрока в зависимости от текущего направления (Rotate).

Внутри блока **switch** происходит проверка условий для каждого из возможных направлений (1 - вправо, 2 - влево, 3 - вверх, 4 - вниз) и осуществляется проверка доступности следующей клетки на карте. Если следующая клетка свободна (то есть не содержит препятствия и не занята другим объектом), то устанавливается новая позиция игрока (NewX, NewY).

Таким образом, оператор switch в данном коде отвечает за определение возможности передвижения игрока в зависимости от текущего направления, и обновляет значения позиции игрока, если следующая клетка доступна для передвижения.

|  |
| --- |
| if ((Map[NewY][NewX] == ' ' || [NewY][NewX] ==  'B')) {  if (Map[NewY][NewX] == ' ')  q++;  Map[y][x] = 'B';  Map[NewY][NewX] = 'C';  x = NewX;  y = NewY;  }  if (NewY == 9 && (NewX == 0 || NewX == 18)) {  if (NewX == 0)  NewX = 17;  else  NewX = 1;  Map[y][x] = 'B';  Map[NewY][NewX] = 'C';  x = NewX;  y = NewY;  } |
| **Код 2.6 – Перемещение игрока** |

Далее, если игрок попадает на клетку со значением " " (пробел) или "B", то происходит обновление карты: клетка, на которой находится игрок, становится "B", а новая клетка, на которую игрок переместился, становится "C". Также обновляются координаты игрока (x и y) и переменные NewX и NewY. Символ "B" обозначает пустые пространства внутри границ карты, "C" – это положение пакмана на карте.

Если игрок находится на клетке (9, 0) или (9, 18), то он переносится на клетку (9, 17) или (9, 1) соответственно. Также происходит обновление карты и координат игрока.

Далее мы переходим к созданию класса Enemy, что является еще одной частью разработки игры. Класс Enemy будет отвечать за поведение противников в игре. Он будет содержать методы для движения противников по карте. Кроме того, класс Enemy будет хранить информацию о позиции противников на карте.

|  |
| --- |
| class Enemy{  public:  int x[4] = { 1, 17, 1, 17 }, y[4] = { 1, 1, 19, 19 };  int NewX[4] = { 0, 0, 0, 0 }, NewY[4] = { 0, 0, 0, 0 };  int Rotate[4] = { 1, 1, 1, 1 }, Time = 0;  bool Restart = false;  void Update(){  if (Restart){  for (int i = 0; i < 4; i++)  Map[y[i]][x[i]] = ' ';  x[0] = 1; x[1] = 17; x[2] = 1; x[3] = 17;  y[0] = 1; y[1] = 1; y[2] = 19; y[3] = 19;  Map[y[0]][x[0]] = '1';  Map[y[1]][x[1]] = '2';  Map[y[2]][x[2]] = '3';  Map[y[3]][x[3]] = '4';  for (int i = 0; i < 4; i++) {  NewX[i] = x[i];  NewY[i] = y[i];  }  Restart = false;  } |
| **Код 2.7 - Enemy** |

В данном коде **x[4]** и **y[4]** представляют собой координаты текущего положения противников на игровой карте, а **NewX[4]** и **NewY[4]** - координаты, в которые противники могут переместиться на следующем шаге. Изменение значений x и y происходит только после того, как противник переместился в позицию, заданную значениями NewX и NewY, и эта позиция была подтверждена как возможная. Стоит учитывать, что противники передвигаются рандомно, поэтому мы используем библиотеку #include <time.h> и свойство rand(), которое отвечает за рандом.

В контексте нашего кода, **Rotate** является переменной, которая хранит текущее направление движения противников в игре "Pac-man". Значение переменной Rotate может быть 1, 2, 3 или 4, что соответствует направлениям вправо, влево, вверх и вниз соответственно. Эта переменная используется в функции Update для проверки возможности движения противников в выбранном направлении.

Метод **Update** в классе Enemy служит для обновления положения врагов на игровой карте. Если значение переменной Restart равно true, метод обнуляет координаты врагов и устанавливает их в исходные позиции на карте. В противном случае метод перемещает врагов на одну клетку в случайном направлении каждые 300 единиц времени. Если враг достигает клетки, занятой игроком, игрок теряет жизнь. Кроме того, если враг достигает клетки со значением 'C', игра заканчивается. После перемещения врагов метод также обновляет их координаты на карте и обрабатывает случаи, когда враг достигает края карты и должен появиться с противоположной стороны.

|  |
| --- |
| else {  Time++;  if (Time >= 300) {  for (int i = 0; i < 4; i++) {  Rotate[i] = rand() % 4 + 1;  NewX[i] = x[i];  NewY[i] = y[i];  switch (Rotate[i]) {  case 1:  if (Map[y[i]][NewX[i] + 1] != 'A')  NewX[i] += 1;  break;  case 2:  if (Map[y[i]][NewX[i] - 1] != 'A')  NewX[i] -= 1;  break;  case 3:  if (Map[NewY[i] - 1][x[i]] != 'A')  NewY[i] -= 1;  break;  case 4:  if (Map[NewY[i] + 1][x[i]] != 'A')  NewY[i] += 1;  break;  }  }  Time = 0;  }  } |
| **Код 2.8 – Условие else** |

В этой части кода происходит движение вражеских персонажей на игровом поле. Если условие else выполняется, то переменная Time увеличивается на единицу. Затем происходит проверка, если значение переменной Time достигло или превысило 300 (пороговое значение), то выполняется следующий блок кода.

Внутри блока кода происходит обновление положения каждого из вражеских персонажей. Для каждого персонажа выполняются следующие действия:

1. Генерируется случайное число в диапазоне от 1 до 4 с помощью функции rand() % 4 + 1. Это число определяет направление движения вражеского персонажа.
2. Затем сохраняются текущие координаты (x и y) персонажа в переменные NewX[i] и NewY[i].
3. С помощью оператора switch проверяется значение переменной Rotate[i] и в зависимости от значения выбирается соответствующее направление движения:

* Если Rotate[i] равно 1, то проверяется, не является ли клетка справа от персонажа (Map[y[i]][NewX[i] + 1]) стеной ('A'). Если нет, то координата NewX[i] увеличивается на 1, что означает движение вправо.
* Если Rotate[i] равно 2, то проверяется, не является ли клетка слева от персонажа (Map[y[i]][NewX[i] - 1]) стеной ('A'). Если нет, то координата NewX[i] уменьшается на 1, что означает движение влево.
* Если Rotate[i] равно 3, то проверяется, не является ли клетка выше персонажа (Map[NewY[i] - 1][x[i]]) стеной ('A'). Если нет, то координата NewY[i] уменьшается на 1, что означает движение вверх.
* Если Rotate[i] равно 4, то проверяется, не является ли клетка ниже персонажа (Map[NewY[i] + 1][x[i]]) стеной ('A'). Если нет, то координата NewY[i] увеличивается на 1, что означает движение вниз.

1. После обновления координат персонажа, переменная Time сбрасывается обратно в ноль.

Таким образом, этот код отвечает за случайное движение вражеских персонажей на игровом поле каждые 300 единиц времени. Каждый вражеский персонаж выбирает случайное направление и проверяет возможность перемещения в этом направлении, и, если это возможно, обновляет свои координаты соответствующим образом.

|  |
| --- |
| for (int i = 0; i < 4; i++) {  if (Map[NewY[i]][NewX[i]] == ' ' ||  Map[NewY[i]][NewX[i]] == 'B' ||  Map[NewY[i]][NewX[i]] == 'C') {  if (Map[NewY[i]][NewX[i]] == 'B')  Map[y[i]][x[i]] = 'B';  else if (Map[NewY[i]][NewX[i]] == ' ')  Map[y[i]][x[i]] = ' ';  else if (Map[NewY[i]][NewX[i]] == 'C')  Life = false;  if (i == 0)  Map[NewY[i]][NewX[i]] = '1';  if (i == 1)  Map[NewY[i]][NewX[i]] = '2';  if (i == 2)  Map[NewY[i]][NewX[i]] = '3';  if (i == 3)  Map[NewY[i]][NewX[i]] = '4';  x[i] = NewX[i];  y[i] = NewY[i];  } |
| **Код 2.9 – Цикл for** |

Данный код представляет собой цикл for, в котором происходит проверка условий для каждого из четырех вражеских персонажей.

Внутри цикла происходит следующее:

1. Проверяется содержимое клетки с координатами NewY[i] и NewX[i] на игровом поле Map. В коде используются условия Map[NewY[i]][NewX[i]] == ' ', Map[NewY[i]][NewX[i]] == 'B' и Map[NewY[i]][NewX[i]] == 'C'.
2. Если содержимое клетки является пробелом (' '), бонусом ('B') или монеткой ('C'), выполняются действия внутри блока if.

* Если содержимое клетки равно 'B', то значение в клетке с предыдущими координатами персонажа (Map[y[i]][x[i]]) устанавливается равным 'B'. Это может означать, что враг поглотил бонус.
* Если содержимое клетки равно пробелу (' '), то значение в клетке с предыдущими координатами персонажа (Map[y[i]][x[i]]) устанавливается равным пробелу. Это может означать, что враг прошел через пустую клетку.
* Если содержимое клетки равно 'C', то переменная Life устанавливается в значение false. Это может означать, что враг столкнулся с персонажем игрока и игрок потерял жизнь.
* Затем, в зависимости от значения i (индекса персонажа) выполняется установка соответствующего значения в клетке с новыми координатами (Map[NewY[i]][NewX[i]]). Например, если i равно 0, то значение в клетке устанавливается равным '1', если i равно 1, то значение в клетке устанавливается равным '2' и так далее.
* Затем значения переменных x[i] и y[i] (координаты персонажа) обновляются значениями NewX[i] и NewY[i].

Таким образом, данный код отвечает за взаимодействие вражеских персонажей с игровым полем и другими объектами на поле. В зависимости от содержимого клетки, враг может взаимодействовать с бонусами, монетками или сталкиваться с персонажем игрока, что влияет на состояние игры и положение вражеских персонажей на поле.

|  |
| --- |
| if (NewY[i] == 9 && (NewX[i] == 0 || NewX[i] ==  18)) {  if (NewX[i] == 0)  NewX[i] = 17;  else  NewX[i] = 1;  Map[y[i]][x[i]] = 'B';  if (i == 0)  Map[NewY[i]][NewX[i]] = '1';  if (i == 1)  Map[NewY[i]][NewX[i]] = '2';  if (i == 2)  Map[NewY[i]][NewX[i]] = '3';  if (i == 3)  Map[NewY[i]][NewX[i]] = '4';  x[i] = NewX[i];  y[i] = NewY[i];  } |
| **Код 2.10 – Условие if** |

Данный код проверяет условие, что координата NewY[i] равна 9 и одна из координат NewX[i] равна 0 или 18.

Если условие выполняется, то выполняются следующие действия:

* Если NewX[i] равно 0, то NewX[i] устанавливается в значение 17. Если NewX[i] равно 18, то NewX[i] устанавливается в значение 1. Это означает, что если враг достигает края игрового поля по горизонтали, то он перемещается на противоположный край поля.
* Значение в клетке с предыдущими координатами персонажа (Map[y[i]][x[i]]) устанавливается равным 'B'. Это может означать, что враг поглотил бонус.
* Затем, в зависимости от значения i (индекса персонажа) выполняется установка соответствующего значения в клетке с новыми координатами (Map[NewY[i]][NewX[i]]). Например, если i равно 0, то значение в клетке устанавливается равным '1', если i равно 1, то значение в клетке устанавливается равным '2' и так далее.
* Затем значения переменных x[i] и y[i] (координаты персонажа) обновляются значениями NewX[i] и NewY[i].

Таким образом, данный код обрабатывает ситуацию, когда враг достигает края игрового поля по горизонтали и выполняет перемещение на противоположный край, а также взаимодействие с бонусами и обновление координат вражеского персонажа на поле.

В принципе метод Update в классе Enemy имеет тот же функционал, что и одноименный метод Update в классе Player, за исключением того, что противников 4 и двигаются они рандомно за счет функции rand(), а не по своей определенной траектории.

**2.3 Объединение компонентов и функция Main**

Данный раздел посвящен объединению компонентов приложения и функции main. Компоненты приложения в данном случае включают в себя классы, отвечающие за карту (Map), игрока (Player), врагов (Enemy), счетчик очков (Score) и глобальные переменные (Global).

В функции main происходит инициализация приложения, загрузка ресурсов (изображений), создание окна и установка иконки. Затем запускается главный цикл приложения, в котором обрабатываются события (например, закрытие окна) и обновляются компоненты игры (позиция игрока и врагов, количество очков и т.д.). После обновления компонентов происходит отрисовка игровых объектов на экране.

Функция main является входной точкой приложения и выполняет все необходимые действия для запуска и корректной работы игры. Она использует объекты классов Map, Player, Enemy, Score и глобальные переменные, объединяя их в единое целое. Благодаря этому, приложение работает как одно целое, а не как отдельные части, что облегчает его разработку и поддержку в будущем.

|  |
| --- |
| int main()  {  srand(time(0));  RenderWindow window(VideoMode(W \* Split, H \* Split + 3 \* Split),  "Pac-man!");  Image icon;  icon.loadFromFile("Image/logo.png");  const Uint8\* pixels = icon.getPixelsPtr();  window.setIcon(icon.getSize().x, icon.getSize().y, pixels);  Texture t;  t.loadFromFile("Image/title.png");  Sprite plat(t);  Texture yw;  yw.loadFromFile("Image/youwin.png");  Sprite youwin(yw);  youwin.setPosition(100, 210);  Texture yl;  yl.loadFromFile("Image/youlose.png");  Sprite youlose(yl);  youlose.setPosition(100, 210);  Texture sc;  sc.loadFromFile("Image/score.png");  Score score(sc);  Player p;  Enemy en; |
| **Код 2.11 - main** |

В функции **main()** происходит инициализация окна приложения, текстур, спрайтов, объектов игрока и врагов, обработка событий и отрисовка игрового поля, персонажей, счётчика очков и сообщений о победе или поражении.

В данном коде происходит отрисовка игровых объектов на окне с помощью функций из библиотеки SFML. Это включает загрузку изображений в текстуры и настройку их параметров, определение позиций игровых объектов на карте, установку соответствующих текстур для каждого объекта и их отображение на экране. При достижении игроком определенного количества очков происходит вывод соответствующего сообщения о победе, а при потере жизней - о поражении.

**RenderWindow window(VideoMode(W \* Split, H \* Split + 3 \* Split),**

**"Pac-man!")** - это создание окна приложения с заданными размерами и заголовком.

Класс **RenderWindow** в библиотеке SFML предоставляет окно, в котором можно отображать графические объекты, включая изображения, текстуры и спрайты. Он представляет собой простой интерфейс для создания и управления окном, в котором можно отображать 2D-графику.

RenderWindow обеспечивает связь между окном и системными событиями, такими как нажатие клавиш, перемещение мыши и закрытие окна. Он также обеспечивает возможность рисования графики на окне, используя буферы кадров для отображения изменений в режиме реального времени.

Кроме того, RenderWindow предоставляет удобный интерфейс для создания и управления спрайтами, текстурами и другими объектами, которые можно отображать на окне. Он также обладает рядом дополнительных функций, таких как установка иконки окна, управление размером и положением окна и другие.

Класс **VideoMode** в библиотеке SFML используется для хранения режима видео-режима экрана, таких как размер окна и глубина цвета. Он используется в конструкторе класса RenderWindow для создания окна приложения с определенными параметрами, такими как размер и заголовок окна.

**Image icon;**

**icon.loadFromFile("Image/logo.png");**

**const Uint8\* pixels = icon.getPixelsPtr();**

**window.setIcon(icon.getSize().x, icon.getSize().y, pixels);** - это загрузка изображения в переменную icon, получение указателя на пиксели изображения и установка иконки окна приложения с помощью функции setIcon().

|  |
| --- |
|  |
| **Рис. 2.1 - logo.png** |

Класс **Image** в библиотеке SFML представляет изображение, которое может быть загружено из файла или создано в памяти. Этот класс позволяет манипулировать пикселями изображения и выполнять различные операции с ними, например, изменять цвета, изменять размеры.

В контексте игровых приложений, класс Image используется для загрузки изображений в память, которые затем могут быть отображены на экране с помощью других классов библиотеки SFML, таких как **Texture** и **Sprite**. Например, в игре можно использовать класс Image для загрузки спрайтов персонажей, объектов игры, фона, интерфейсных элементов.

**Texture t;**

**t.loadFromFile("Image/title.png");**

**Sprite plat(t);** - это загрузка текстуры из файла и создание спрайта на основе этой текстуры.

|  |
| --- |
|  |
| **Рис. 2.2 -title.png** |

**Texture yw;**

**yw.loadFromFile("Image/youwin.png");**

**Sprite youwin(yw);**

**youwin.setPosition(100, 210);**

И

**Texture yl; yl.loadFromFile("Image/youlose.png");**

**Sprite youlose(yl);**

**youlose.setPosition(100, 210);** - это загрузка текстур и создание

спрайтов для изображений, отображаемых при выигрыше или проигрыше игрока, и установка позиции этих спрайтов на экране.

|  |  |
| --- | --- |
|  |  |
| **Рис. 2.3 - youwin.png** | **Рис. 2.4 - youlose.png** |

**Texture sc;**

**sc.loadFromFile("Image/score.png");**

**Score score(sc);** - это загрузка текстуры для счета и создание объекта Score, который отвечает за отображение и обновление текущего счета в игре.

|  |
| --- |
|  |
| **Рис. 2.5 - score.png** |

|  |
| --- |
| while (window.isOpen()) {  Event event;  while (window.pollEvent(event)) {  if (event.type == Event::Closed)  window.close();  if (q < 194 && Life) {  if (event.type == Event::KeyPressed) {  p.NewX = p.x;  p.NewY = p.y;  if (event.key.code == Keyboard::Right)  p.Rotate = 1;  if (event.key.code == Keyboard::Left)  p.Rotate = 2;  if (event.key.code == Keyboard::Up)  p.Rotate = 3;  if (event.key.code == Keyboard::Down)  p.Rotate = 4; }  }  } |
| **Код 2.12 – Цикл while** |

Данный код представляет собой основной цикл игры, который выполняется, пока окно игры открыто (window.isOpen() возвращает true).

Внутри цикла происходит обработка событий (window.pollEvent(event)), таких как закрытие окна. Если событие типа "закрытие окна" (event.type == Event::Closed), то окно закрывается (window.close()).

Далее проверяется условие, что переменная q меньше 194 и переменная Life равна true. Если это условие выполняется, то происходит обработка событий нажатия клавиш. Если событие типа "нажатие клавиши" (event.type == Event::KeyPressed), то выполняются следующие действия:

Значения p.NewX и p.NewY (новые координаты игрока) устанавливаются равными текущим координатам игрока (p.x и p.y соответственно).

Затем, в зависимости от кода нажатой клавиши (event.key.code), устанавливается значение переменной p.Rotate (направление вращения игрока). Если нажата клавиша "Вправо", то p.Rotate устанавливается равным 1, если нажата клавиша "Влево", то p.Rotate устанавливается равным 2, если нажата клавиша "Вверх", то p.Rotate устанавливается равным 3, и если нажата клавиша "Вниз", то p.Rotate устанавливается равным 4.

Таким образом, данный код обрабатывает события закрытия окна и нажатия клавиш, связанные с управлением игроком.

|  |
| --- |
| if ((q == 194 || !Life) && RestartTime != 0) {  RestartTime--;  if (RestartTime == 0) {  for (int i = 0; i < H; i++)  for (int j = 0; j < W; j++)  if (Map[i][j] == 'B')  Map[i][j] = ' ';  p.Restart = true;  en.Restart = true;  q = 0;  Life = true;  RestartTime = 5000;  }  }  score.c = q;  if (!Life)  for (int i = 0; i < H; i++)  for (int j = 0; j < W; j++)  if (Map[i][j] != 'A')  Map[i][j] = 'B';  if (q < 194 && Life) {  p.Update();  en.Update();  }  score.Update();  window.clear(Color::Black); |
| **Код 2.13 – Условия обновления** |

Данный код содержит логику для обновления состояния игры и рендеринга графики. Он выполняет следующие действия:

Если условие (q == 194 || !Life) && RestartTime != 0 выполняется, то значение переменной RestartTime уменьшается на единицу.

Если RestartTime становится равным нулю, происходит перезапуск игры.

Если игра неактивна (игрок умер), то все элементы на игровом поле, кроме символов 'A', заменяются на символы 'B'.

Если значение q меньше 194 и игра активна (игрок жив), то происходит обновление состояния объектов игрока и врагов.

Обновляется состояние объекта score.

Очищается окно от предыдущих отрисованных элементов, заполняя его черным цветом.

Код в целом обеспечивает обновление игровой логики, проверку условий победы или поражения, а также обновление графического интерфейса игры для отображения текущего состояния.

|  |
| --- |
| for (int i = 0; i < H; i++)  for (int j = 0; j < W; j++) {  if (Map[i][j] == 'A')  plat.setTextureRect(IntRect(0, 0, Split, Split));  if (Map[i][j] == 'C')  plat.setTextureRect(IntRect(Split \* int(p.frame),  Split \* p.Rotate, Split, Split));  if (Map[i][j] == ' ')  plat.setTextureRect(IntRect(Split, 0, Split, Split));  if (Map[i][j] == '1')  plat.setTextureRect(IntRect(Split \* 5, Split \*  en.Rotate[0], Split, Split));  if (Map[i][j] == '2')  plat.setTextureRect(IntRect(Split \* 5, Split \*  en.Rotate[1], Split, Split));  if (Map[i][j] == '3')  plat.setTextureRect(IntRect(Split \* 5, Split \*  en.Rotate[2], Split, Split));  if (Map[i][j] == '4')  plat.setTextureRect(IntRect(Split \* 5, Split \*  en.Rotate[3], Split, Split));  if (Map[i][j] == 'B')  continue;  plat.setPosition(j \* Split, i \* Split);  window.draw(plat);  }  if (q == 194)  window.draw(youwin);  if (!Life)  window.draw(youlose);  for (int i = 0; i < 3; i++)  if (score.vid[i])  window.draw(score.sec[i]);  window.display();  }  return 0; |
| **Код 2.14 – Визуализация игрового поля** |

Данный код отвечает за визуальное представление игрового поля и элементов интерфейса на экране. Он содержит цикл, который проходит по каждому элементу игрового поля, представленного в виде двумерного массива под названием Map. Внутри цикла определяется текстурный прямоугольник для каждого элемента в зависимости от его значения в массиве Map.

После определения текстурного прямоугольника, он устанавливается на нужную позицию на экране и выполняется его отрисовка с помощью функции window.draw(). Таким образом, каждый элемент игрового поля отображается на экране в соответствии с его текстурой.

Далее в коде проверяется условие q == 194, которое означает, что игрок достиг определенного условия победы в игре. В случае истинности этого условия, на экране отображается специальный элемент с надписью "youwin", который информирует игрока о победе.

Если переменная Life равна false, что указывает на проигрыш игрока, на экране отображается элемент с надписью "youlose", который информирует игрока о поражении.

Также в коде есть цикл, который отвечает за отрисовку секундомеров в зависимости от значения переменной score.vid[i]. Этот цикл обычно используется для отображения времени игры или другой информации, связанной со временем.

В конце кода происходит отображение всех отрисованных элементов на экране с помощью функции window.display(). После этого код возвращает значение 0.

**ЗАКЛЮЧЕНИЕ**

Наш проект, написанный на языке С++, использующий библиотеку SFML и вдохновленный игрой "Pac-Man", является примером того, как с помощью программирования и различных классов можно создавать интересные и захватывающие игры. Каждый из классов, включая Global, Map, Score, Player и Enemy, выполняет определенные функции, отвечая за логику игры, визуализацию и взаимодействие объектов.

Этот проект позволяет нам увидеть, как разработчики игр используют инструменты и библиотеки, такие как SFML, для создания игрового окна, обработки событий, загрузки текстур и управления спрайтами. Каждый класс имеет свою роль в игровом процессе: Map отвечает за создание игрового поля, Player управляет персонажем Pac-Man, Enemy управляет вражескими привидениями, а Score отслеживает и отображает счет игрока.

Основываясь на этом проекте и в целом на геймдеве (игровая разработка), можно сделать несколько выводов о настоящем и будущем этой области:

Индустрия геймдева продолжает расти и развиваться. С каждым годом все больше людей интересуются созданием игр и стремятся освоить соответствующие навыки программирования и дизайна.

Инструменты разработки игр становятся более доступными и удобными. Библиотеки и фреймворки, такие как SFML, предоставляют разработчикам мощные инструменты для создания игровых проектов любого масштаба.

Ключевыми элементами успешного геймдева остаются креативность и инновационный подход. Хороший геймдев проект должен предлагать уникальный игровой опыт, привлекать внимание игроков и предлагать что-то новое и интересное.

Технологический прогресс влияет на развитие геймдева. Новые технологии, такие как виртуальная реальность, дополненная реальность и искусственный интеллект, открывают новые возможности для создания более реалистичных и захватывающих игр.

Рост популярности инди-игр. С появлением платформы для самостоятельной публикации игр и разветвлением сообщества инди-разработчиков, небольшие команды и отдельные разработчики получают возможность выпускать свои игры на мировой рынок. Это способствует разнообразию игрового контента и позволяет новым талантам проявить себя.

Будущее геймдева обещает еще больше инноваций и возможностей. С развитием искусственного интеллекта, машинного обучения и глубокого обучения, игры будут способны создавать более умных и реалистичных виртуальных миров, а алгоритмы противников будут становиться все более сложными и адаптивными. Развитие виртуальной и дополненной реальности предложит игрокам новые способы взаимодействия с игровым миром, создавая еще более увлекательные и погружающие впечатления.

В целом, геймдев является динамичной и перспективной областью, которая будет продолжать привлекать новых талантов и развиваться в направлении создания увлекательных, инновационных и социально-ориентированных игровых проектов. Наш проект, написанный на языке С++, использующий библиотеку SFML и вдохновленный игрой "Pac-Man", является лишь небольшим примером этого потенциала.

**СПИСОК ЛИТЕРАТУРЫ**

1. https://stoneforest.ru/look/hobby/videogames/istoriya-igry-pacman/
2. https://ravesli.com/urok-2-vvedenie-v-yazyki-programmirovaniya-c-i-s/
3. https://metanit.com/cpp/tutorial/1.1.php
4. https://tproger.ru/articles/kak-nastroit-biblioteku-sfml-v-visual-studio/
5. Арьян Терпстра, Тим Лапетино. Pac-Man: Birth of an Icon.
6. Джейми Лендино. Pac-Man: За лабиринтом.
7. Дэвид Кардинал. Идеальный забег: Руководство по Pac-Man.
8. Габи Альтер, Джош Элдер. Лихорадка Pac-Man: История невероятного

хита 80-х, который определил целое поколение.

1. Джесси Шелл. Искусство геймдизайна: Книга линз.
2. Роберт Найстром. Шаблоны программирования игр.
3. Джейсон Грегори. Архитектура игрового движка.
4. Бьярне Страуструп. Язык программирования C++.
5. Скотт Мейерс. Эффективный современный C++.
6. Стэнли Липпман, Джозе Ладжуа, Барбара Му. Учебник по C++.
7. Энтони Уильямс. Параллелизм C++ в действии.
8. Эккель Брюс. Философия C++.
9. Раймондас Пупиус. Разработка игры на SFML на примере.
10. Максим Барбье. SFML Blueprints.
11. Милчо Мильчев. Основы SFML.
12. Ян Халлер, Хенрик Фогелиус Ханссон и Артур Морейра.

Разработка игр на SFML.

**Вложения**

|  |
| --- |
|  |
| **Рис. 1.1 - Пустой проект** |

|  |
| --- |
|  |
| **Рис. 1.2 - Solution Explorer** |

|  |
| --- |
|  |
| **Рис. 1.3 - New Item** |

|  |
| --- |
|  |
| Рис. 1.4. Pac.cpp |