Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Петрозаводский государственный университет»
Физико-технический институт

КУРСОВОЙ ПРОЕКТ

по дисциплине «Технологии программирования»
тема: «Разработка игры Станция Кисараги»

Авторы работы:
студенты группы 21312
А. Запорожец
А. В. Селецкая
А. Г. Смирнов
А. И. Шаркевич
Научный руководитель:
канд. физ.-мат. наук, доцент КИИСиФЭ А. В. Бульба

Петрозаводск 2023

Содержание

1. Введение

1.1. Цель и назначение работы

Целью работы является командная реализация простой 2D игры на языке C++ (IDE QT Creator) с использованием библиотеки SFML. В качестве названия игры была выбрана формулировка “Станция Кисараги”.

1.2 Актуальность

Обучение ООП: Проект включает в себя реализацию сущностей, наследование, модульное программирование и другие основные концепции объектно-ориентированного программирования (ООП).

Практический опыт с графикой: Игровой проект включает в себя работу с изображениями, текстурами, спрайтами и другими элементами графики. Наша команда смогла получить практический опыт в области обработки графики и взаимодействия с пользователем в графическом интерфейсе.

Работа с событиями: Игра включает в себя обработку пользовательских событий, таких как нажатия клавиш и взаимодействие с объектами на экране. Это предоставило нам опыт работы с событийно-управляемым программированием.

Анализ UML-диаграмм классов позволило нам изучить применение структурных паттернов проектирования, таких как композиция и наследование, для построения гибкой и структурированной программы.

Работа в команде: Мы получили опыт совместной разработки, совместной работы с исходным кодом и взаимодействия в команде.

Таким образом, проект по созданию игры является актуальным с точки зрения изучения конкретных технологий программирования и применения их в практических задачах.

2. Основная часть

2.1 Программная реализация

В качестве среды разработки использовалась программа “Qt Creator” версии 5.4.2. Программа написана на языке программирования высокого уровня C++.

Заголовочные файлы и их краткое описание:

"arrmaps.h"

Хранит объявление и инициализацию массивов с масками карт.

"libs.h"

Хранит включение необходимых библиотек для проекта. Включает заголовочные файлы для ввода/вывода, работы со строками, списками, графикой и звуком с использованием библиотеки SFML. Также использует пространство имен sf и std для упрощения доступа к классам и функциям этих библиотек.

"entity.h"

Хранит интерфейсную часть класса Entity, представляющего базовую сущность в игровом мире. Включает в себя переменные для отслеживания состояния, координат, скорости, размеров, здоровья и текстуры объекта. Также содержит перечисление для возможных состояний объекта. Класс имеет методы для получения прямоугольника, представляющего границы объекта, и виртуальный метод update, который должен быть реализован в производных классах для обновления состояния объекта.

"bullet.h"

Хранит описание класса Bullet, представляющего объект пули в игровом мире. Этот класс наследуется от базового класса Entity. Включает в себя

переменную для отслеживания направления полета пули. Класс также содержит метод update, который обновляет состояние пули в игре.

"deathanimation.h"

Содержит объявление класса DeathAnimation, предназначенного для воспроизведения анимации смерти в игре. Включает заголовочный файл "libs.h" для использования необходимых библиотек.

"ending.h"

Содержит объявление класса Ending, предназначенного для воспроизведения анимации завершения игры. Класс включает в себя окно отрисовки SFML и использует звуковые эффекты. Включает заголовочный файл "libs.h" для использования необходимых библиотек.

"enemy.h"

Содержит объявление класса Enemy, представляющего врага в игре. Класс унаследован от класса Entity и включает в себя дополнительные методы для проверки столкновений с картой, обновления состояния и спавна монетки после смерти врага.

"game.h"

Содержит объявление класса Game, представляющего основной игровой процесс. Включает в себя методы и поля для обработки событий, обновления состояния игры, отрисовки элементов и управления основными логическими аспектами игры.

"map.h"

Хранит объявление класса Map, предназначенного для работы с картой в игре. Включает в себя константы для размеров карты, а также методы для случайной генерации карты, получения массива с маской карты и отрисовки карты на экране.

"player.h"

Хранит объявление класса Player, представляющего игрового персонажа. Включает в себя поля для хранения очков игрока, номера текущей комнаты, флага уничтожения всех врагов, а также звуковых буферов и звуков для воспроизведения звуков при открытии двери. Класс также содержит методы для управления игроком, проверки столкновений с элементами карты, обновления состояния и проверки столкновения с дверью.

"vendingmachine.h"

Хранит объявление класса VendingMachine, представляющего торговый автомат в игре. Включает в себя поле для хранения цены товара, а также методы для проверки возможности покупки товара игроком, обмена монет игрока на товар и обновления состояния автомата.

Файлы реализации и их краткое описание:

"entity.cpp"

В данном файле представлена реализация конструктора и метода getRect для класса Entity. Конструктор устанавливает начальные параметры сущности, такие как координаты, размеры, имя, текущий таймер движения и т.д. Метод getRect возвращает объект типа FloatRect, представляющий

прямоугольную область, охватывающую сущность. Этот прямоугольник используется для проверки пересечений с другими объектами в игре.

"bullet.cpp"

Реализация методов класса Bullet. В конструкторе инициализируются начальные параметры пули, такие как координаты, направление движения, скорость и размеры. Метод update отвечает за движение пули и обработку столкновений с элементами карты. Если пуля сталкивается с землей, дверью или другими преградами, она исчезает (life = false).

"deathanimation.cpp"

Реализация методов класса DeathAnimation. В конструкторе происходит загрузка шрифта и установка начальных параметров текста. Метод playAnimation() запускает анимацию смерти, включая воспроизведение звукового эффекта. В течение заданного времени отображается текст "You died", а затем происходит плавное исчезновение текста с использованием изменения прозрачности. После завершения анимации окно игры закрывается.

"ending.cpp"

Реализация методов класса Ending. В конструкторе происходит загрузка шрифта и установка начальных параметров текста для завершающей анимации. Метод playAnimation() запускает анимацию завершения, включая воспроизведение звукового эффекта. В течение заданного времени отображается текст "Congratulations! You escaped the damned Kisaragu Station", а затем происходит плавное исчезновение текста с использованием изменения прозрачности. После завершения анимации окно закрывается.

"enemy.cpp"

Реализация методов класса Enemy. В конструкторе устанавливаются начальные параметры врага в зависимости от его типа. Метод checkCollisionWithMap обрабатывает столкновения врага с элементами карты. Метод SpawnCoin спавнит монету после смерти врага. Метод update обновляет состояние врага в соответствии с его текущим направлением, анимацией и взаимодействием с картой. Если у врага заканчиваются жизни, он умирает.

"game.cpp"

Файл реализации класса Game. В конструкторе инициализируется игровое окно, игрок, текстовые элементы и звуки. Также происходит загрузка текстур и звуков. Деструктор осуществляет очистку списков врагов и пуль при завершении игры.

Метод run() реализует основной цикл игры, в котором обрабатываются события, обновляются объекты и отрисовывается игровое окно. Метод loadTextures() загружает текстуры, шрифты и звуковые файлы.

Методы handleEvents() и update() отвечают за обработку событий и обновление логики игры соответственно. Метод draw() отвечает за отрисовку объектов на экране.

Метод checkOptionForWindow() отслеживает события окончания игры, такие как смерть игрока или успешное прохождение уровня. Метод changeMapLogic() управляет сменой карт и обновлением соответствующих флагов.

"main.cpp"

Файл main.cpp содержит точку входа в программу. В функции main() создается объект класса Game, передается изображение игрока и запускается основной цикл игры с вызовом метода run().

"map.cpp"

В данном коде реализованы методы класса Map. Конструкторы инициализируют поля объекта, также есть метод для случайной генерации расположения камней на карте. Метод draw отрисовывает карту на экране, устанавливая текстуру спрайта в зависимости от символов в массиве TileMap.

"player.cpp"

В файле player.cpp реализованы методы класса Player, который представляет игрового персонажа. Реализованы функции управления, обработки столкновений с картой и дверьми, обновления состояния персонажа и проверки его жизни.

"vendingmachine.cpp"

В файле vendingmachine.cpp определены методы класса VendingMachine. Метод canAfford() проверяет, может ли игрок позволить себе покупку от автомата, и exchangeCoins() осуществляет обмен монет игрока на товар, если покупка возможна. Метод update() устанавливает позицию спрайта автомата.

2.2 Пошаговое описание процесса разработки

2.2.1 Описание сюжета игры

Станция Кисараги - игра, сочетающая в себе элементы головоломки, ужаса и аркады.

Главная героиня - Юки, старшеклассница, возвращается с учёбы домой. Чтобы оказаться дома ей нужно сесть на поезд, поэтому она прибывает на станцию Кисараги. Оказавшись на станции, девочка засыпает от усталости и пропускает свой поезд. Очнувшись, она замечает, что станция опустела и населена призраками. Теперь Юки нужно разобраться со всеми странностями вокруг, решить головоломки и сесть на поезд, который обязательно отвезёт её домой.

Сюжет будет подаваться заставками с текстом (в начале игры). По ходу самой игры, будут подаваться через всплывающие плашки-диалоги.

Игрок управляет персонажем, который может передвигаться по локации, подбирать предметы и атаковать врагов. Основная часть геймплея это головоломки. Они будут двух типов: предметные и логические.
Для решения первых, игроку нужно будет найти подходящий предмет, который может находиться в одной из доступных локаций. Для того, чтобы подобрать, нужно подойти к предмету и нажать клавишу действия. После чего, нужно вернуться к головоломке и нажать клавишу действия.
Для решения логических, нужно будет активировать предметы в правильном порядке. Необходимо подойти к нужному предмету и нажать клавишу действия.
По ходу игры мы будем сталкиваться с разными врагами. Их условно можно разделить на два типа: ближники и дальники. Ближники подлетают в упор и наносят урон, дальники же атакуют издалека, выпуская в сторону персонажа снаряд. При гибели оба вида призраков могут оставить после себя монетку.
Для своей защиты персонаж может использовать благовония и мешочек с рисом.
Благовония поджигаются и выпускают в сторону одного врага сгусток дыма, который уничтожает его. Мешочек с рисом бросается на землю и все враги вокруг летят к нему, теряя на какое-то время возможность атаковать игрока. Оба предмета тратятся после использования, но их

можно носить с собой в количестве нескольких штук, а так же находить в комнатах по всей станции.

Каждая вражеская атака отнимает у персонажа часть здоровья. Когда здоровье опускается до нуля, персонаж умирает и игру придётся начать сначала. Чтобы это избежать, можно прятаться в специальной безопасной комнате. В ней находится торговый автомат, который выдаёт предметы в обмен на монетки. Для лечения используются еда или бинты. Оба предмета восстанавливают часть здоровья, но различия только в количестве восстановленных очков.

Для завершения игры нужно решить все головоломки и сесть на поезд.

2.2.2 Действующие субъекты

Игрок (управление персонажем)

2.2.3 Варианты использования и их описание, диаграмма

Список вариантов использования:

- Запустить игру;
- Переместить персонажа на карте;
- Переместиться в другую локацию;
- Взаимодействовать с окружением;
- Подобрать предмет;
- Атаковать врага;
- Бросить мешок с рисом;
- Потерять здоровье;
- Восстановить здоровье;
- Умереть;
- Закрыть игру.

Диаграмма вариантов использования:

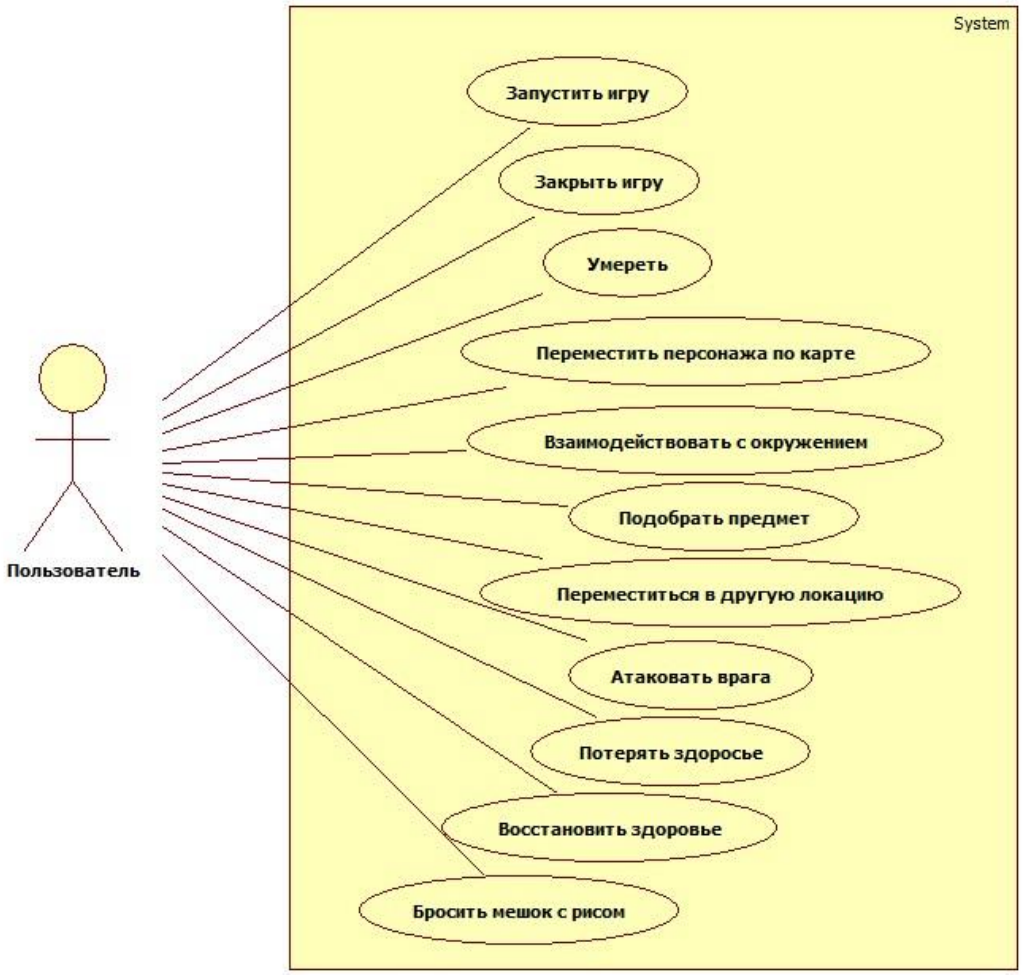


Рис. 1 Диаграмма прецедентов

Описание вариантов использования:
“Запустить игру”:

- Когда запускается игра, на экран должно выводиться меню, из которого можно начать игру или выйти. На экране в этот момент написано название игры.

“Переместить персонажа на карте”:

- Во время самой игры наш персонаж сможет передвигаться по карте. При нажатии клавиши движения он будет передвигаться в нужную сторону.

“Переместиться в другую локацию”:

- При приближении к краю локации можно будет перейти в другую, если там есть дверь или другой переход. При переходе текущая локация стирается с экрана и выводится следующая. Положение персонажа будет зависеть от положения перехода в обратную локацию.

“Взаимодействовать с окружением”:

- По ходу игры будут на карте будут встречаться объекты, которые нельзя подобрать, но с которыми можно взаимодействовать. Это камни, двери, зелёные лужи и тд. Для их использования нужно подойти персонажем к ним вплотную.

“Подобрать предмет”:

- Есть объекты, которые герой может подбирать и хранить. Это монетки. Персонаж игрока должен будет наступить на них, чтобы подобрать.Они добавляются в список, сверху слева выводится их кол-во.

“Атаковать врага”:

- Для атаки врага нужно нажать специальную клавишу. После этого персонаж “родит” объект-”пулю”, который летит по прямой в направлении, куда смотрит игрок. В случае пересечения(попадания) отнимает у него часть здоровья/убивает и пропадает с карты. В случае столкновения с предметом или краем карты тоже исчезает.

“Бросить мешок с рисом”:

- При нажатии специальной клавиши из персонажа вылетит мешочек с рисом. Он летит по прямой, но меньше чем пуля. По окончанию движения он какое-то время лежит на земле, притягивая к себе призраков. По истечении времени он исчезает с карты.

“Потерять здоровье”:

- Когда враг попадает по персонажу атакой или персонаж наступает на зелёную лужу, он получает урон. Из его общего здоровья вычитается столько, сколько урона наносит атака. Если он уже был ранен, то вычитается из текущего уровня здоровья. При значении здоровья 0, персонаж умирает.

“Восстановить здоровье”:

- Когда персонаж находит еду, он подбирает её и восстанавливает здоровье. К его текущему значению здоровья прибавляется столько, сколько восстанавливает еда. Нельзя восстановить здоровье выше максимального значения. Если здоровье персонажа полное, он не сможет подобрать еду и на экране высветится “Вы полностью здоровы”.

“Умереть”:

- Когда здоровье персонажа падает до нуля или меньше, игра прекращается. Карта, предметы и сущности стираются. На экран выводится сообщение “Вы мертвы. Сдаться?” и две надписи “да” и “нет”. Если нажать “да”, то игра начнётся с самого начала. Если выбрать “нет”, то переход в меню закрытия игры.

“Закрыть игру”:

- Во время игры можно нажать на кнопку закрытия окна (крестик в правом верхнем углу окна), тогда игра остановится и прогресс сбросится.

2.2.4 Классы программы, их описание и диаграмма классов

Список классов:

1. Класс игры
2. Сущность (род. класс для персонажа и врагов)
3. Персонаж
4. Враг (общий для всех типов врагов)
5. Снаряд
6. Карта
7. Торговый автомат
8. Экран смерти
9. Экран выхода из игры

Описание классов программы:

- Класс игры (Game):

Отвечает за управление игровым процессом. Загружает изображения, инициализирует объекты и запускает основной цикл игры.

- Сущность (Entity):

Абстрактный родительский класс для персонажа и врагов. Содержит общие свойства и методы, такие как координаты, скорость, и обработка столкновений.

- Персонаж (Player):

Представляет игрового персонажа, управляемого игроком. Обрабатывает ввод пользователя, столкновения с картой и дверьми, а также обновление состояния персонажа.

- Враг (Enemy):

Представляет врагов, с которыми может столкнуться персонаж. Включает общие характеристики и методы для врагов различных типов.

- Снаряд (Bullet):

Класс объектов, представляющих собой снаряд, выпущенный персонажем..

- Карта (Map):

Хранит информацию о расположении объектов на карте и отвечает за её отрисовку.

- Торговый автомат (VendingMachine):

Представляет объект торгового автомата в игре. Отвечает за расположение автомата на карте, проверку возможности покупки и обмен монет игрока на товар.

- Экран смерти (DeathAnimation):

Отображает информацию о смерти персонажа и закрывает окно игры.

- Экран выхода из игры (Ending):

Отображает информацию о завершении игры и закрывает окно.

Диаграмма классов:

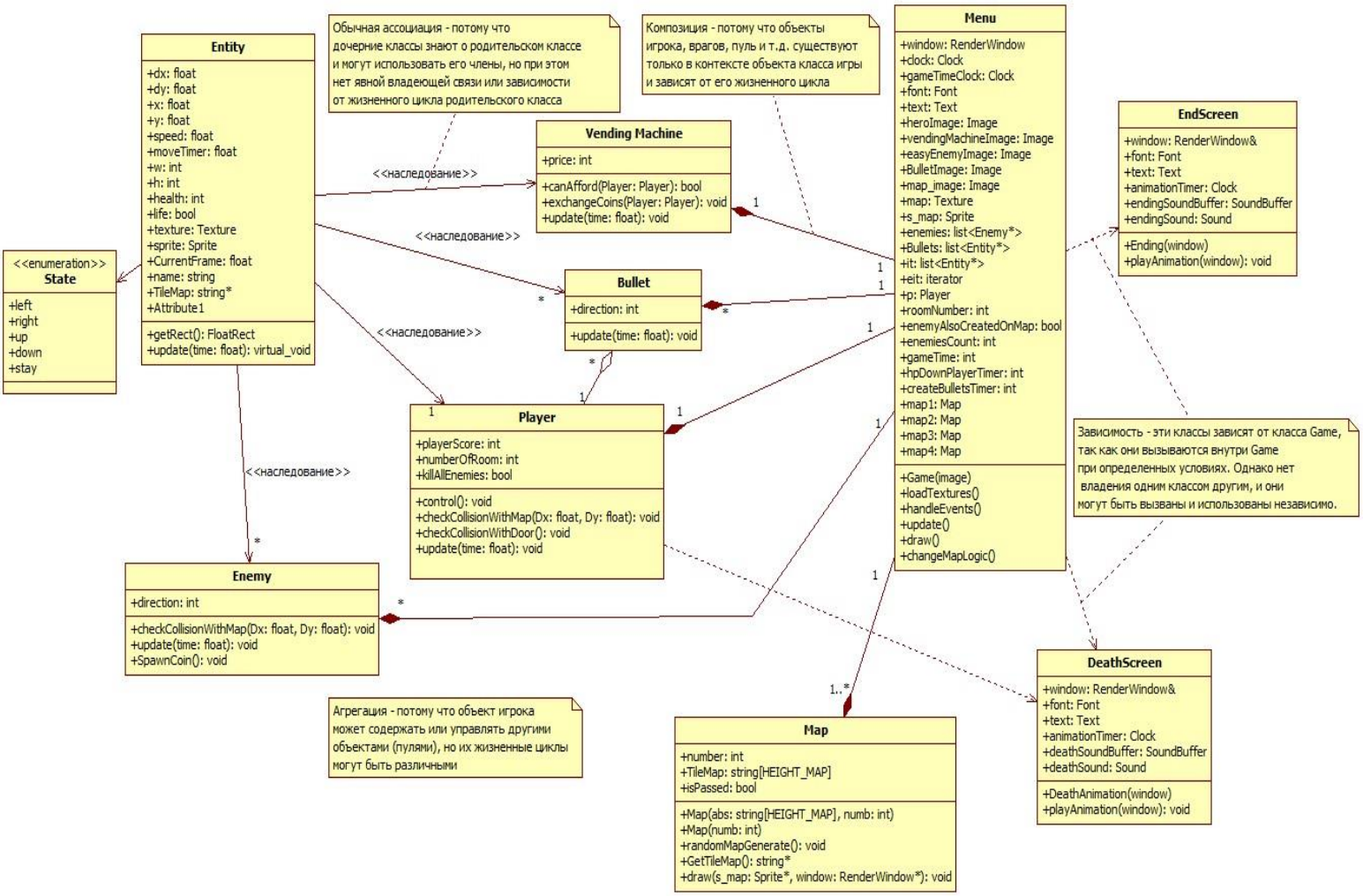


Рис. 2 Диаграмма прецедентов

2.2.5 Написание кода программы

Полный код заголовочных файлов представлен в Приложении 1. (не для печати)

2.2.6 Написание исходных файлов (.cpp)

Полный код файлов реализации представлен в Приложении 2. (не для печати)

2.2.7 Руководство пользователя

Запустив программу, пользователя встречает начальная локация.

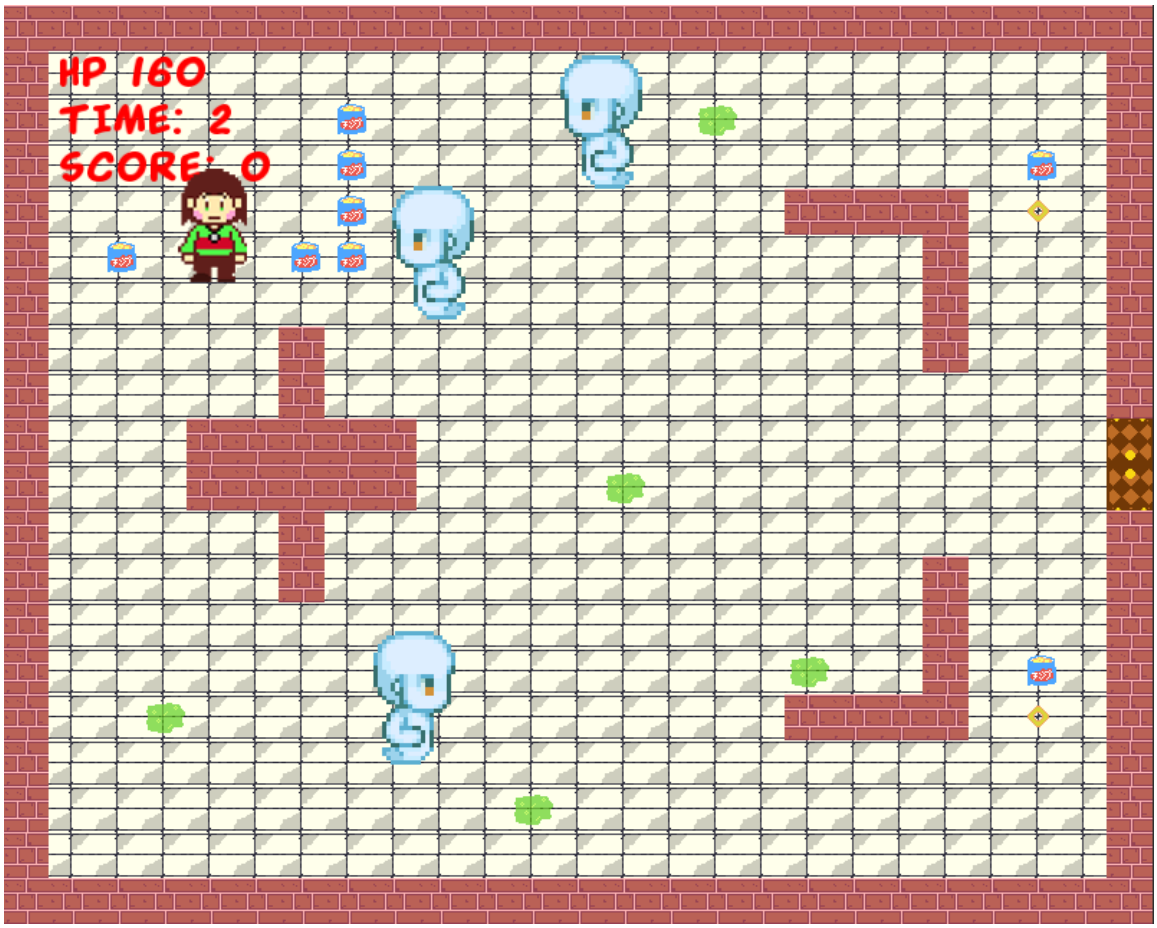


Рис. 3 Начальная комната

Управление

Тут можно ознакомиться с управлением:
По нажатию клавиш WASD, персонаж перемещается по карте в то направление, которому соответствует нажатая клавиша (W - вверх, A - влево, S - вниз, D - вправо).

Для защиты от врагов персонаж может стрелять. При нажатии клавиши E из персонажа, с характерным звуком, вылетает пуля в то направление, в которое повёрнута его моделька.

Рис. 4 Стрельба

Это всё, что нужно знать по управлению.

Враги и препятствия

Уже в начальной комнате можно встретиться со всем, что будем мешать персонажу по ходу игры. Это враги и препятствия:

Враги
В каждой комнате находится по три врага. Они произвольно перемещаются по комнате и наносят урон игровому персонажу только тогда, когда пересекаются с ним.

Рис. 5 Враги

Рис. 6 Момент нанесения урона

Если враги нанесут достаточно урона, то персонаж умирает и на экран выводится соответствующее сообщение.

Рис. 7 Экран смерти

Чтобы победить врага, в него нужно стрелять. После трёх попаданий он исчезает с карты с характерным звуком, а на его месте остаётся четыре монетки.

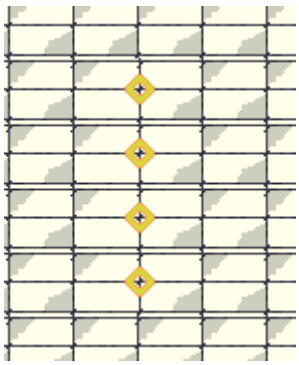


Рис. 8 Монетки

После своей смерти враги больше не появляются в комнате.

Препятствия

Кроме врагов в каждой комнате находится несколько луж слизи, которые наносят персонажу урон. Для того, чтобы не получать урон, их нужно обходить и стараться не наступить на них.

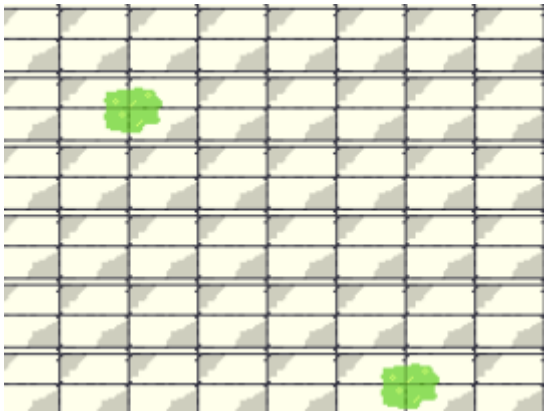


Рис. 9 Лужи слизи

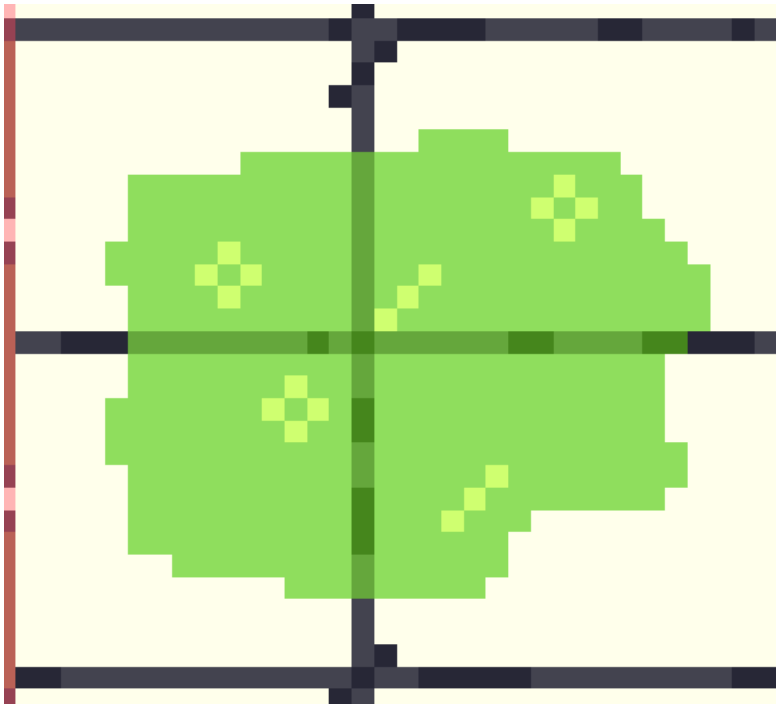


Рис. 10 Текстура лужи вблизи

Ещё как препятствие можно выделить двери между комнатами. Они бывают двух видов: вертикальные и горизонтальные. Изначально они закрыты. Чтобы пройти в другую комнату, нужно уничтожить всех врагов в текущей, тогда двери откроются.

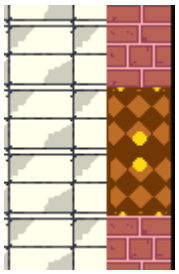


Рис. 11 Двери вертикальные

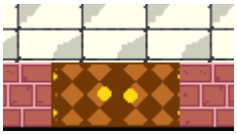


Рис. 12 Двери горизонтальные

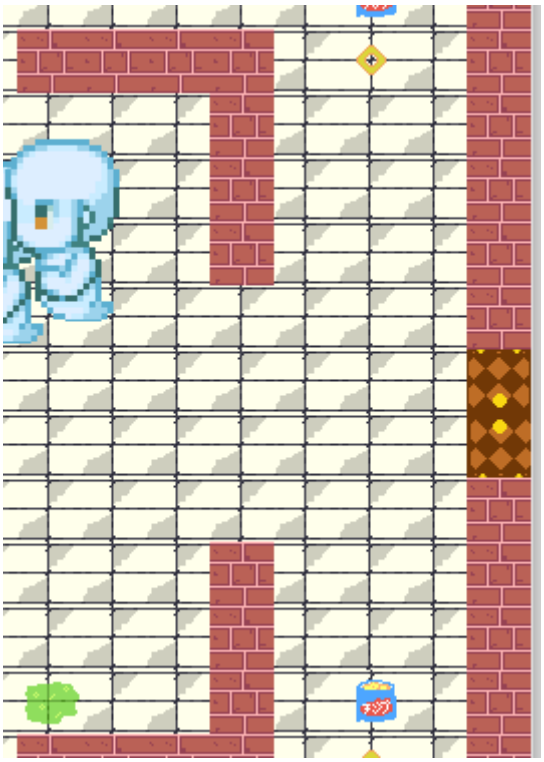


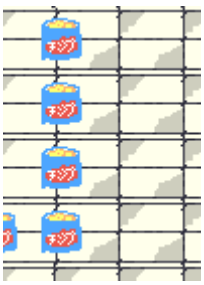
Рис. 13 Пример расположения двери в локации

Предметы

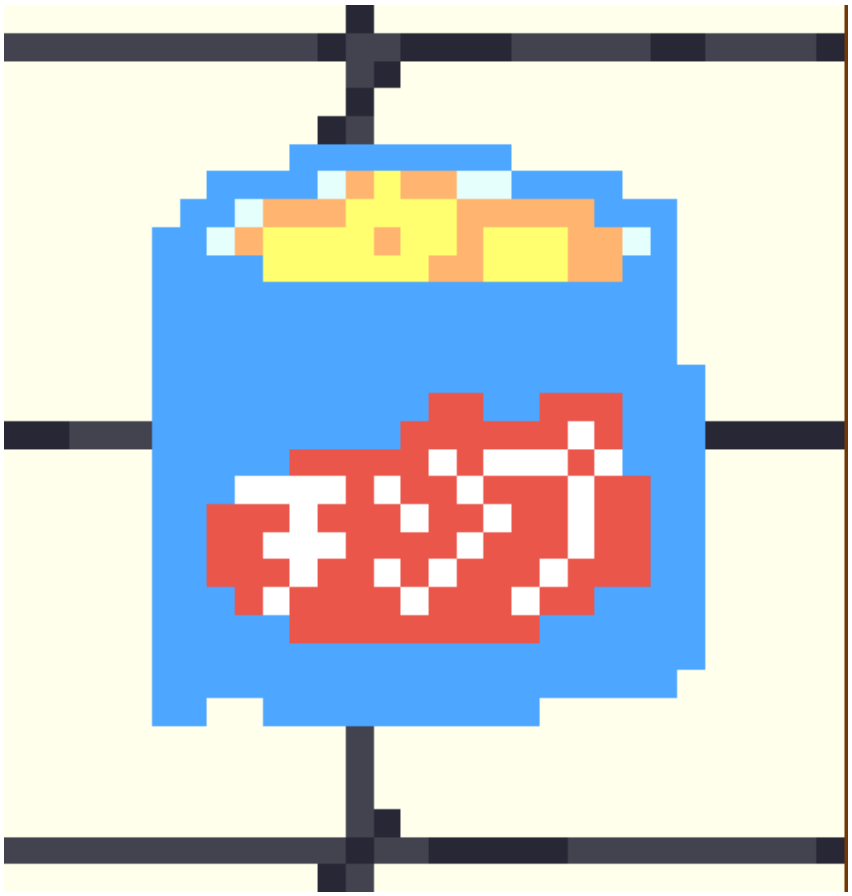
Кроме врагов и препятствий в игре присутствуют предметы, которые должны помогать персонажу и поощрять его за прохождение комнаты.

Еда
Предмет, который помогает восстановить здоровье или временно увеличить его общий лимит. В игре еда представлена пачкой чипсов. В каждой комнате она генерируется в определённых местах и

фиксированном количестве, не обновляясь со временем или сменой комнаты.



Несколько единиц еды



Текстура еды вблизи

(Забавный факт - на пачке чипсов на японском написано слово “чипсы”)

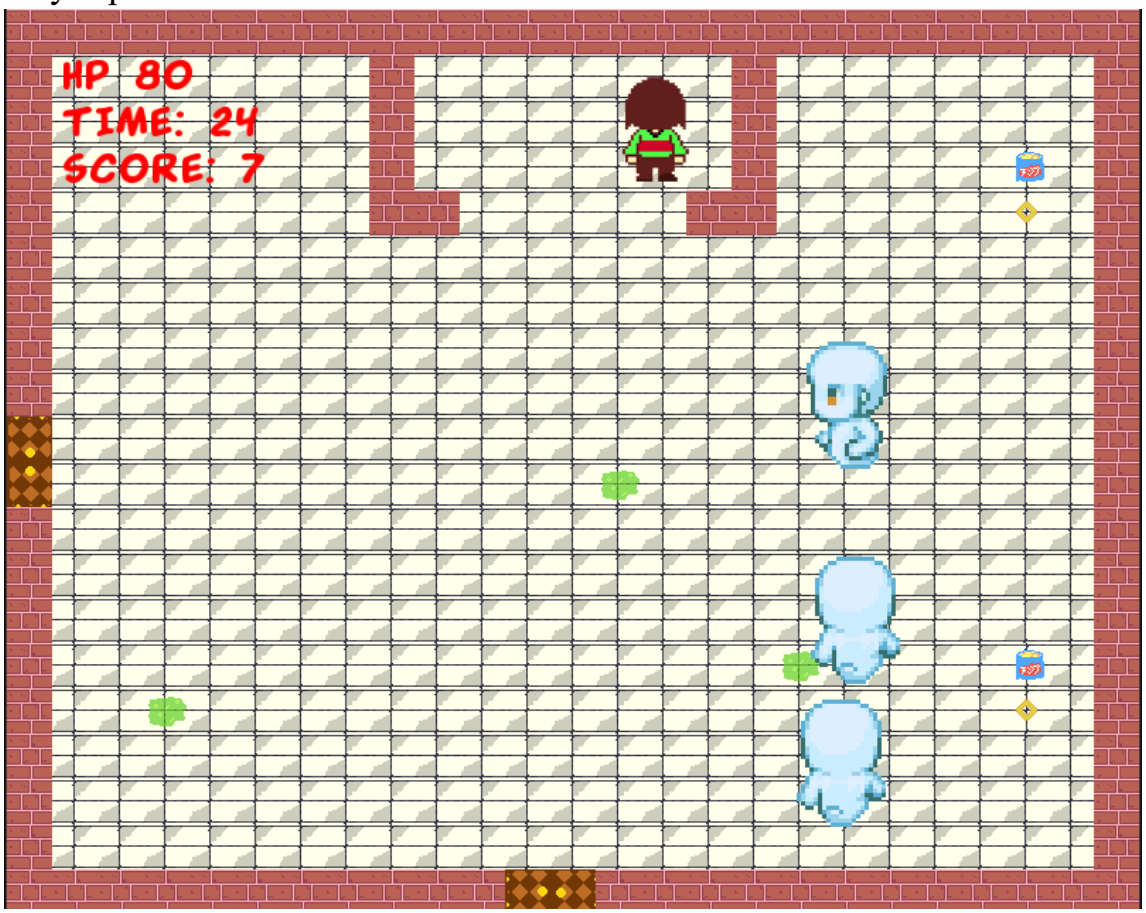
Монетки

Монетки валяются на полу каждой комнаты в фиксированном количестве и не обновляются со временем или сменой локации. Так же они выпадают из врагов, как поощрение за то, что вы их победили.

Прохождение игры

Для завершения игры нужно пройти все комнаты, победить всех врагов и

не умереть.



Комната номер 2

2.3 История проекта на GitHub, gitk

Ссылка на репозиторий: https://github.com/Artazar2033/KisaraguStation.git

Для контроля версий проекта был создан публичный репозиторий KisaraguStation. Разработка осуществляется с использованием ветвей – master (помещаются стабильные версии с тэгом), develop-general (рабочие, но не готовые к релизу версии), а также от ветви develop-general создаются ветви функциональностей, в которых работает каждый из участников команды: develop-Alex (Алексей Смирнов), develop-Artem (Артем Запорожец), develop-Senya (Арсений Шаркевич) и feature-Anna (Анна Селецкая). Ветви функциональностей используются для создания и

отладки новых функций, после того, как они доведены до стабильного состояния, ветка сливается с develop-general.

В разработке участвуют все члены команды. Автор коммита будет подписан никнеймом GitHub:

- Запорожец Артем – Artazar2033
- Смирнов Алексей – GigaBuddy0x0
- Шаркевич Арсений – StoneFree2011
- Селецкая Анна – woopydoopy

Все коммиты:

1) StoneFree2011 - beta - реализована первая beta-версия игры с возможностями героя передвигаться по комнате, ограниченной стенами. В верхнем левом углу экрана показаны показатели здоровья, очков и таймер от начала игры. Герой может выпускать снаряды, собирать с пола монетки, повышающие очки, сердца, восстанавливающие здоровье, ядовитые поля, отнимающие здоровье. В комнате присутствуют враги, которые умирают при пересечении с пулей. Пули исчезают при пересечении со стеной или врагом. Игрок умирает при пересечении с врагом.
2) StoneFree2011 - beta- изменена текстура главного героя
3) GigaBuddy0x0 - First develop commit, downdate the version to 2.3.2 - версия SFML понижена до более стабильной 2.3.2
4) GigaBuddy0x0 - Разделили классы Entity, Player и Enemy на .h и .cpp, Создали класс Map. Изменили реализацию классов с учётом нового класса. - Упразднен файл classes.cpp, вместо этого созданы заголовочные и реализационные файлы для всех классов
5) GigaBuddy0x0 - Разбили классы на отдельные файлы, создали файл для массивов маски карты arrmaps.h - завершен перенос классов в свои файлы, создан специальный заголовочный файл с масками карт комнат
6) GigaBuddy0x0 - убрали из под отслеживания файл *.pro
7) Artazar2033 - Изменены текстуры карты
8) Artazar2033 - Текстуры добавлены в актуальную версию

9) StoneFree2011-Улучшена структура проекта, убраны из-под отслеживания ненужные файлы
10) GigaBuddy0x0 - Исправил проблему с отсутствием задержек выстрелов и получения урона (со временем), перенёс отрисовку карты в метод класса Map, добавил логику перехода между комнатами. - Пули теперь создаются с задержкой по таймеру, после получения урона от врага игроком появляется время неуязвимости от урона.
11) GigaBuddy0x0 - Полностью реализован переход между комнатами, нарисовал несколько масок карт - Переход между комнатами через двери работает полностью стабильно, добавлены новые комнаты
12) GigaBuddy0x0 - Merge branch 'develop-Alex' into develop-general - в основную ветку разработки влита ветка с разработкой функций перехода между комнатами, таймеров для выстрелов и урона врагов.
13) Artazar2033 - Локально-бесполезный коммит - убраны из-под отслеживания мусорные файлы
14) Artazar2033 - Локально-бесполезный коммит 2.0;(- добавлен под отслеживание файл gitignore, дабы никто из участников разработки его больше потерял и он был настроен единообразно
15) Artazar2033 - fix conflict - разрешен конфликт вливания ветки develop-general в функциональную ветку develop-Artem
16) Artazar2033 - Добавлены текстуры двери
17) Artazar2033 - Добавлены текстуры двери, исправлен алгоритм отрисовки - новые текстуры дверей, полностью стабильная отрисовка карты
18) GigaBuddy0x0 - Окончательно разобрался с функцией перехода между комнатами - checkCollisionWithDoor(). Создание объектов врагов перенесено в main и теперь зависит от текущей комнаты. - упразднен метод класса Player::teleport и вместо него реализован более стабильный метод перехода между комнатами checkCollisionWithDoor(). Три экземпляра класса врагов теперь появляются в каждой комнате (до этого три врага появлялись один раз при запуске игры)

19) StoneFree2011 - Добавлен метод SpawnCoin()- создание монетки после смерти врага - создан метод для класса Enemy, создающий на его месте монеты после смерти
20) StoneFree2011 - слияние с develop-general - разрешен конфликт слияния функциональной ветки Develop-Senya в основную ветку разработки. Перенос в нее метода создания монетки после смерти врага
21) StoneFree2011 - Изменен метод Bullet::update - проверка на столкновение с дверью. Список с экземплярами врагов изменен на класс Enemy и добавлен итератор для них для корректного обращения к методу SpawnCoin - в метод класса Bullet::update добавлена проверка на то, чтобы снаряд удалялся после столкновения с дверью. Список, в который заносятся экземпляры Enemy() теперь создается не от базового класса Entity, а от Enemy, чтобы можно было использовать его методы.
22) GigaBuddy0x0 - Добавил поля для классов Map и Player, реализовал ситуации, при которых 1) невозможно перейти в др комнату, не убив всех врагов 2) при входе в ранее зачищенную комнату враги снова не появляются. - теперь для использования двери для перехода в другую комнату необходимо убить всех врагов в ней, после чего в этой комнате враги появляться больше не будут.
23) GigaBuddy0x0 - Merge branch 'develop-Alex' into develop-general - функционал из прошлого коммита добавлен в основную ветку разработки
24) GigaBuddy0x0 - Конфликты разрешены - разрешение конфликта слияния
25) GigaBuddy0x0 - Размер игрового окна теперь зависит от размеров карты (2 константы) - размер окна программы определяется от констант высоты и ширины карты
26) GigaBuddy0x0 - Merge branch 'develop-Alex' into develop-general - добавление функционала из прошлого коммита в основную ветку разработки

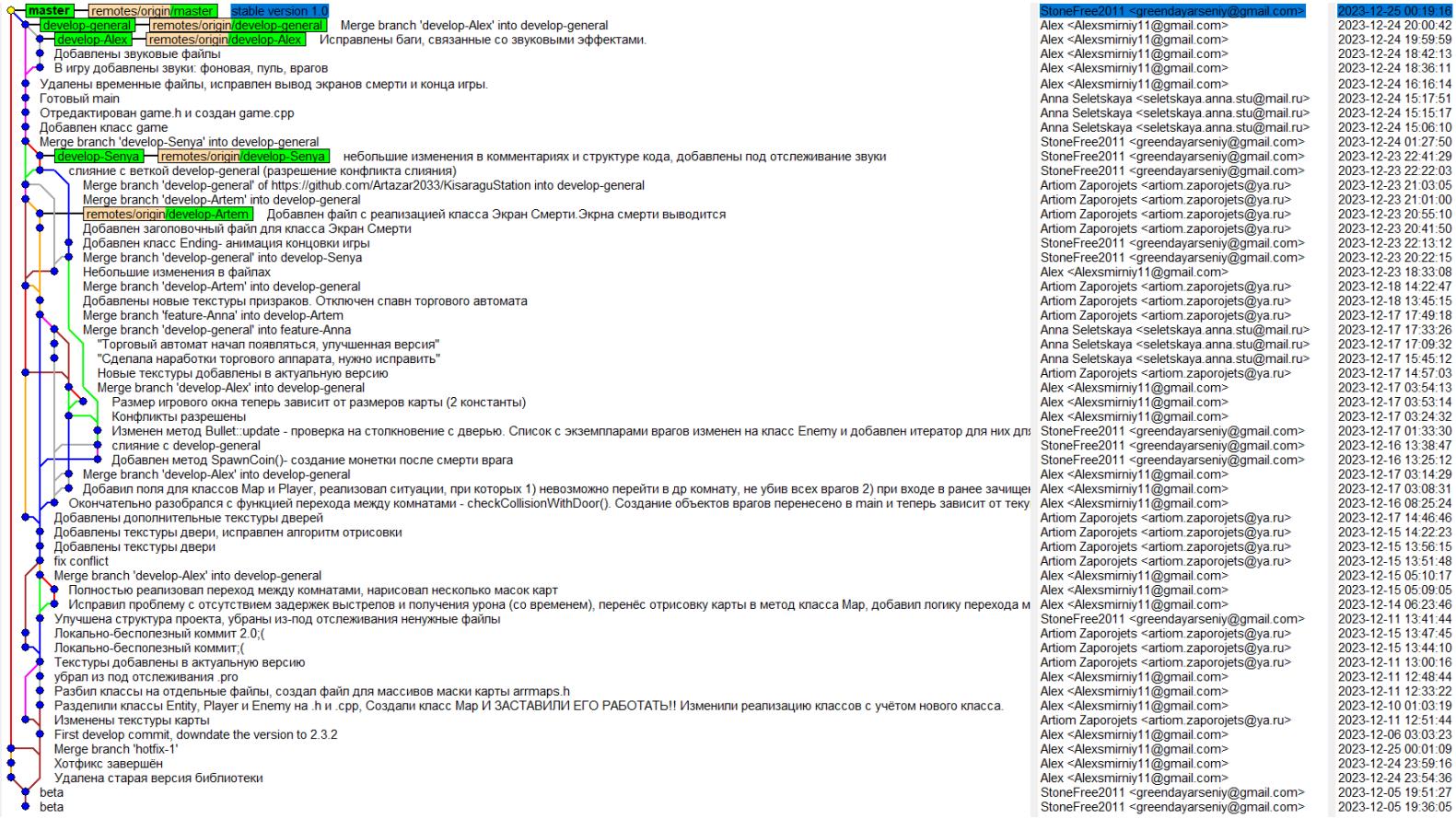
27) Artazar2033 - Добавлены дополнительные текстуры дверей - текстуры для вертикально и горизонтально расположенных дверей теперь разные
28) Artazar2033 - Новые текстуры добавлены в актуальную версию
29) woopydoopy - "Сделала наработки торгового аппарата, нужно исправить" - добавлен класс игрового автомата VENDINGMACHINE от родительского класса Entity и класс Coin для монетки (впоследствии решено не выносить реализацию монеты в отдельный класс). Игровой автомат должен обменивать очки персонажа в обмен на поднятие здоровья.
30) woopydoopy -"Торговый автомат начал появляться, улучшенная версия" - добавлена текстура игрового автомата, в main добавлена отрисовка игрового автомата в первой комнате. Однако, методы взаимодействия игрока с автоматом решено пока что не вызывать ввиду их не полной доработки.
31) woopydoopy - Merge branch 'develop-general' into feature-Anna
32) Artazar2033 - Merge branch 'feature-Anna' into develop-Artem
33) Artazar2033 - Добавлены новые текстуры призраков. Отключен спавн торгового автомата - изменены текстуры врагов, временно отключено создание на карте торгового автомата
34) Artazar2033 - Merge branch 'develop-Artem' into develop-general - изменения из предыдущих коммитов внесены в основную ветку разработки
35) GigaBuddy0x0 -Небольшие изменения в файлах - структуризация кода в main.cpp, player.cpp, player.h
36) StoneFree2011 -Merge branch 'develop-general' into develop-Senya

37) Artazar2033 - Добавлен заголовочный файл для класса Экран Смерти - описан новый класс deathanimation, воспроизводящий анимацию при смерти игрока
38) Artazar2033 - Добавлен файл с реализацией класса Экран Смерти.Экрна смерти выводится - класс deathanimation полностью реализован, его метод playAnimation(window) вызывается при смерти игрока
39) StoneFree2011 - Добавлен класс Ending- анимация концовки игры - описан и реализован класс Ending, вызывающий метод playAnimation(window) когда игрок проходит игру (то есть убивает последнего врага в последней комнате)
40) StoneFree2011 - слияние с веткой develop-general (разрешение конфликта слияния)
41) StoneFree2011 - небольшие изменения в комментариях и структуре кода, добавлены под отслеживание звуки - звуки death_sound.wav и ending_sound.wav добавлены под отслеживание
42) StoneFree2011 - Merge branch 'develop-Senya' into develop-general - все стабильные изменения внесены в основную ветку разработки
43) woopydoopy - Добавлен класс game - в заголовочном файле game.h описан абсолютно новый класс game, в который впоследствии предполагается перенести всю логику игры, на тот момент находящуюся в main.cpp.
44) woopydoopy - Отредактирован game.h и создан game.cpp - полностью реализован класс game, в котором находится абсолютно вся логика игры
45) woopydoopy - Готовый main - полностью переписан файл main.cpp, теперь в него только подключается заголовочный файл

класса game, создается его экземпляр и вызывается метод game.run(), отвечающий за всю логику игры

46) GigaBuddy0x0 - Удалены временные файлы, исправлен вывод экранов смерти и конца игры. - более стабильная работа классов deathanimation и ending
47) GigaBuddy0x0 - В игру добавлены звуки: фоновая, пуль, врагов - добавлены звуки выстрелов, смерти врагов и фоновая музыка
48) GigaBuddy0x0 - Добавлены звуковые файлы - все звуки перенесены в специальную папку sounds, она добавлена под отслеживание
49) GigaBuddy0x0 - Исправлены баги, связанные со звуковыми эффектами. - из main убран явный вызов деструктора ~game, вызывавший ошибки, в самом деструкторе убрана неправильная очистка буферов звука
50) GigaBuddy0x0 - Merge branch 'develop-Alex' into develop-general - звуки перенесены в основную ветку разработки
51) GigaBuddy0x0 - Удалена старая версия библиотеки - от ветки master создана ветка hotfix, в которой была проведена очистка от старых ненужных файлов, для более легкого последующего слияния
52) GigaBuddy0x0 - Хотфикс завершён - окончательная очистка, хотфикс готов к слиянию
53) GigaBuddy0x0 - Merge branch 'hotfix-1' - хотфикс влит в master
54) StoneFree2011 - stable version 1.0 - ветка разработки влита в master. Выпущена стабильная версия игры

Скриншот из gitk:



3. Заключение

Нашей командой была разработана 2D игра “Станция Кисараги” с использованием среды разработки Qt-Creator и языка программирования высокого уровня C++. При разработке использовалась система контроля версий Git. Во время программирования в промежуточных версиях могли наблюдаться сбои, но такие версии не попадали в историю Git’a. Их локально исправляли и только после этого коммитили. На текущий момент в ходе тестирования программы ошибок и сбоев в её работе не выявлено. Использованы принципы раздельной компиляции. Классы разделены на заголовочные файлы, в которых есть интерфейсная часть, и файлы с реализацией, в которых есть тела самих классов. Функции реализованы с использованием свойств модульного программирования и наследования. Приложены диаграмма вариантов использования и диаграмма классов. Главная цель работы была достигнута.

Приложение 1

1. arrmaps.h

```
#ifndef ARRMAPS
#define ARRMAPS

#include "map.h"

string ArrMap1[HEIGHT_MAP] = {
  "000000000000000000000000",
  "0                      0",
  "0      h       f       0",
  "0      h              h 0",
  "0      h         0000 s 0",
  "0 hhhhhh             0   0",
  "0                  0   0",
  "0     0            0   0",
  "0     0                0",
  "0   00000              ?",
  "0   00000    f         !",
  "0     0                0",
  "0     0            0   0",
  "0                  0   0",
  "0               f  0 h 0",
  "0  f            0000 s 0",
  "0                      0",
  "0         f            0",
  "0                      0",
  "000000000000000000000000",
};

string ArrMap2[HEIGHT_MAP] = {
  "000000000000000000000000",
  "0       0       0       0",
  "0       0       0       0",
  "0       0       0     h 0",
  "0       00      00      s 0",
  "0                       0",
  "0                       0",
  "0    f                  0",
  "0                       0",
  "?                       0",
  "!           f           0",
  "0                       0",
```

```
  "0                    0",
  "0                    0",
  "0               f    h 0",
  "0  f                 s 0",
  "0                    0",
  "0                    0",
  "0                    0",
  "00000000000()000000000000",
};

string ArrMap3[HEIGHT_MAP] = {
  "00000000000()000000000000",
  "0                    0",
  "0             f      0",
  "0                   h 0",
  "0                    s 0",
  "0                    0",
  "0                    0",
  "0                    0",
  "0                    0",
  "000000000000000000     ?",
  "000000000000000000     !",
  "0                    0",
  "0     s        s     0",
  "0    s s      s s    0",
  "0   s s s    s s s   0",
  "0    s s      s s    0",
  "0     s        s     0",
  "0         hhh        0",
  "0                    0",
  "000000000000000000000000000",
};

string ArrMap4[HEIGHT_MAP] = {
  "000000000000000000000000000",
  "0                    0",
  "0             f      0",
  "0                   h 0",
  "0        00000       s 0",
  "0                    0",
  "0                    0",
  "0                    0",
  "0                    0",
  "?       000000000      0",
```

```
  "!            f          0",
  "0                       0",
  "0                       0",
  "0                       0",
  "0                f    h 0",
  "0  f      00000       s 0",
  "0                       0",
  "0          f            0",
  "0                       0",
  "00000000000000000000000000",
};

#endif // ARRMAPS
```

2. bullet.h

```
#ifndef BULLET
#define BULLET

# include "entity.h"

//////////////////////////КЛАСС ПУЛИ//////////////////////
class Bullet :public Entity{//класс пули
public:
  int direction;//направление пули
  //всё так же, только взяли в конце состояние игрока (int dir)
  //для задания направления полёта пули
  Bullet(Image &image, float X, float Y, int W, int H, string Name, int dir, string* MapMap);
  void update(float time);
  //void SpawnCoin(); //метод спавна монет (не используется)
};


#endif // BULLET
```

3. DeathAnimation.h

```
#ifndef DEATHANIMATION
#define DEATHANIMATION

#include "libs.h"

class DeathAnimation {
public:
  DeathAnimation(RenderWindow& window);
  void playAnimation(RenderWindow& originalWindow);
```

```
private:
  RenderWindow& window;
  Font font;
  Text text;
  Clock animationTimer;
  SoundBuffer deathSoundBuffer;
  Sound deathSound;
};

#endif // DEATHANIMATION
```

4. ending.h

```
#ifndef ENDING
#define ENDING
#include "libs.h"

class Ending {
public:
  Ending(RenderWindow& window);
  void playAnimation();

private:
  RenderWindow& window;
  Font font;
  Text text;
  Clock animationTimer;
  SoundBuffer endingSoundBuffer;
  Sound endingSound;
};
#endif // ENDING
```

5. enemy.h

```
#ifndef ENEMY
#define ENEMY

#include "entity.h"

//////////////////////////КЛАСС ВРАГА//////////////////////
class Enemy :public Entity{
public:
  int direction;//направление движения врага

  Enemy(Image &image, float X, float Y, int W, int H, string Name, string* MapMap);
```

```
  void checkCollisionWithMap(float Dx, float Dy);//ф-ция проверки столкновений с картой
  void update(float time);
  void SpawnCoin(); //спавн монетки после смерти
};//класс Enemy закрыт

#endif // ENEMY
```

6. entity.h

```
#ifndef ENTITY
#define ENTITY

#include "map.h"

//////////////////////////КЛАСС СУЩНОСТЬ//////////////////////
class Entity {
public:
  enum { left, right, up, down, stay} state;// тип перечисления - состояние объекта
  float dx, dy, x, y, speed, moveTimer;//добавили переменную таймер для будущих целей
  int w, h, health; //переменная “health”, хранящая жизни
  bool life; //переменная “life” жизнь, логическая
  Texture texture;//сфмл текстура
  Sprite sprite;//сфмл спрайт
  float CurrentFrame;//хранит текущий кадр
  string name;//враги могут быть разные, врагов можно различать по именам
  //каждому можно дать свое действие в update() в зависимости от имени

  string* TileMap;

  Entity(Image &image, float X, float Y, int W, int H, string Name, string* MapMap);
  FloatRect getRect();//метод получения прямоугольника
  virtual void update(float time) = 0;
  //virtual void SpawnCoin() = 0;
};

#endif // ENTITY
```

7. game.h

```
#ifndef GAME
#define GAME

#include "bullet.h"
#include "enemy.h"
#include "player.h"
```

```
#include "vendingmachine.h"
#include "deathanimation.h"
#include "ending.h"

const int PLAYER_DAMAGE = 40; //урон от пули по врагу
const int FIRE_SPEED = 500; //скорострельность в мс
const int ENEMY_COUNT = 3; //максимальное количество врагов в игре

class Game {
public:
  Game(Image& im);

  ~Game(); //на всякий случай

  void run();

private:
  RenderWindow window;
  Clock clock;
  Clock gameTimeClock;//переменная игрового времени, будем здесь хранить время
игры
  Font font;
  Text text;
  Image heroImage;
  Image vendingMachineImage;
  Image easyEnemyImage;
  Image BulletImage;
  Image map_image;//объект изображения для карты
  Texture map;
  Sprite s_map;
  list<Enemy*> enemies;
  list<Entity*> Bullets;
  list<Entity*>::iterator it;//итератор класса Entity
  list<Enemy*>::iterator eit;//итератор по классу Enemy
  Player p;

  SoundBuffer backgroundBuffer;
  Sound backgroundSound; //3:58
  SoundBuffer gunshotBuffer;
  Sound gunshotSound;
  SoundBuffer gostBuffer;
  Sound gostSound;
  float backgroundMusicTimer;
```

```
    int roomNumber = 1;
    bool enemyAlsoCreatedOnMap = true;
    int enemiesCount = 0;
    int gameTime;
    int hpDownPlayerTimer;//Переменная под время для неуязвимости игрока после получения урона
    int createBulletsTimer;//Переменная под время для задержки выстрела
    Map map1;
    Map map2;
    Map map3;
    Map map4;

    void loadTextures();

    void handleEvents();

    void update();

    void draw();

    void checkOptionForWindow();

    void changeMapLogic();
};

#endif // GAME
```

8. libs.h

```
#ifndef LIBS_H
#define LIBS_H

#include <iostream>
#include <sstream>
#include <list>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

using namespace sf;
using namespace std;

#endif // LIBS_H
```

9. map.h

```
#ifndef MAP
```

```
#define MAP

#include "libs.h"

const int HEIGHT_MAP = 20;//размер карты высота
const int WIDTH_MAP = 25;//размер карты ширина

class Map{
public:
  int number;
  string TileMap[HEIGHT_MAP]; // = массив с маской карты (тогда использовать конструктор без параметров)
  bool isPassed;
  Map(string abs[HEIGHT_MAP], int numb);
  Map(int numb);
  void randomMapGenerate();
  string* GetTileMap();
  void draw(Sprite* s_map, RenderWindow* window);
};

#endif // MAP
```

10. player.h

```
#ifndef PLAYER
#define PLAYER

#include "entity.h"

//////////////////////////КЛАСС ИГРОКА//////////////////////
class Player :public Entity {
public:
  int playerScore;//эта переменная может быть только у игрока
  int numberOfRoom;
  bool killAllEnemies;

  SoundBuffer doorBuffer;
  Sound doorSound;

  Player(Image &image, float X, float Y, int W, int H, string Name, string* MapMap);
  void control();
  //Метод проверки столкновений с элементами карты
  void checkCollisionWithMap(float Dx, float Dy);/////////////////////////////////////////////

  //void teleport(int* i, int* j, int* num); //метод проверки столкновения с комнатой
```

```
  void checkCollisionWithDoor();

  void update(float time);

  //void gainCoin();
  //void SpawnCoin(); //метод спавна монет (не используется)

};

#endif // PLAYER
```

11. vendingmachine.h

```
#ifndef VENDINGMACHINE
#define VENDINGMACHINE
#include "entity.h"
#include "player.h"


class VendingMachine:public Entity {
public:
  int price;

  VendingMachine(Image &image, float X, float Y, int W, int H, string Name, string*
MapMap);

  bool canAfford(Player& Player);

  void exchangeCoins(Player& Player);

   void update(float time); //метод "оживления/обновления" объекта класса.

   //void GainCoin
};


#endif // VENDINGMACHINE
```

a) bullet.cpp

```
#include "bullet.h"

Bullet::Bullet(Image &image, float X, float Y, int W, int H, string Name, int dir, string*
MapMap)
  :Entity(image, X, Y, W, H, Name, MapMap){
  x = X;
  y = Y;
  direction = dir;
  speed = 0.8;
  w = h = 16;
  life = true;
  //выше инициализация в конструкторе
}

/*void Bullet::SpawnCoin() //метод спавна монет (не используется)
{
  //пусто
}*/

void Bullet::update(float time)
{
  switch (direction)
  {
  case 0: dx = -speed; dy = 0; break;// state = left
  case 1: dx = speed; dy = 0; break;// state = right
  case 2: dx = 0; dy = -speed; break;// state = up
  case 3: dx = 0; dy = speed; break;// state = down
  //default: dx = 0; dy = 0; break;
  }
  if (life){
     x += dx*time;//само движение пули по x
     y += dy*time;//по y
     if (x <= 0) x = 20;// задержка пули в левой стене, чтобы при проседании кадров
     //она случайно не вылетела за предел карты и не было ошибки (сервер может
тормозить!)
     if (y <= 0) y = 20;
     if (x >= 800) x = 780;// задержка пули в правой стене, чтобы при проседании
     //кадров она случайно не вылетела за предел карты и не было ошибки (сервер
может тормозить!)
     if (y >= 640) y = 620;
     for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты
```

```
            for (int j = x / 32; j < (x + w) / 32; j++)
            {
                if ((TileMap[i][j] == '0') || (TileMap[i][j] == '?') ||
                    (TileMap[i][j] == '!') || (TileMap[i][j] == '(') || (TileMap[i][j] == ')'))
                //если элемент наш тайлик земли или двери, то
                    life = false;// то пуля умирает
            }
        sprite.setPosition(x + w / 2, y + h / 2);//задается позицию пули
    }
}
```

б) deathanimation.cpp

```
#include "deathanimation.h"

DeathAnimation::DeathAnimation(RenderWindow& window) : window(window) {
    // Загрузите текстуры и настройте спрайты
    if (!font.loadFromFile("MP Manga.ttf")) {
        // Обработка ошибки загрузки шрифта
    }
    text.setFont(font);
    text.setString("You died");
    text.setCharacterSize(48);
    text.setColor(Color::Red);

    // Настройка начальной позиции текста (выравнивание по центру экрана)
    FloatRect textBounds = text.getLocalBounds();
    text.setPosition((window.getSize().x - textBounds.width) / 2, (window.getSize().y -
textBounds.height) / 2);

    // Настройка таймера для анимации (если нужно)
    animationTimer.restart();
}

void DeathAnimation::playAnimation(RenderWindow& originalWindow) {
    // Загрузка звукового файла
    SoundBuffer deathSoundBuffer;
            if (!deathSoundBuffer.loadFromFile("death_sound.wav")) {
                // Обработка ошибки загрузки звукового файла
            }

            // Создание объекта звука
            Sound deathSound(deathSoundBuffer);

            // Воспроизведение звука
```

```
        deathSound.play();

    while (animationTimer.getElapsedTime().asSeconds() < 5.0) { // Продолжайте анимацию в течение 3 секунд (настройте под свои нужды)
        window.clear();

        // Отображение анимации
        window.draw(text);

        window.display();
    }
    // Плавное исчезновение
    Clock fadeTimer;
    while (fadeTimer.getElapsedTime().asSeconds() < 1.0) { // Например, плавное исчезновение за 1 секунду
        window.clear();

        // Изменение прозрачности текста
        Color textColor = text.getColor();
        textColor.a = static_cast<Uint8>(255 - 255 * (fadeTimer.getElapsedTime().asSeconds() / 1.0));
        text.setColor(textColor);

        // Отображение анимации
        window.draw(text);

        window.display();
    }
    // Возврат к основному окну игры
    window.close();
    originalWindow.display();
  }
```

в) ending.cpp

```
#include "ending.h"

Ending::Ending(RenderWindow& window) : window(window) {
  // Загрузите текстуры и настройте спрайты
  if (!font.loadFromFile("MP Manga.ttf")) {
    // Обработка ошибки загрузки шрифта
  }
  text.setFont(font);
  text.setString("          Congratulations!\nYou escaped the damned Kisaragu Station\n");
  text.setCharacterSize(28);
```

```
  text.setColor(Color::Red);

  // Настройка начальной позиции текста (выравнивание по центру экрана)
  FloatRect textBounds = text.getLocalBounds();
  text.setPosition((window.getSize().x - textBounds.width) / 2, (window.getSize().y -
textBounds.height) / 2);

  // Настройка таймера для анимации (если нужно)
  animationTimer.restart();
}

void Ending::playAnimation(RenderWindow& originalWindow) {
  // Загрузка звукового файла
  SoundBuffer endingSoundBuffer;
        if (!endingSoundBuffer.loadFromFile("ending_sound.wav")) {
           // Обработка ошибки загрузки звукового файла
        }

        // Создание объекта звука
        Sound endingSound(endingSoundBuffer);

        // Воспроизведение звука
        endingSound.play();

      while (animationTimer.getElapsedTime().asSeconds() < 10) {
        window.clear();

        // Отображение анимации
        window.draw(text);

        window.display();
      }
      // Плавное исчезновение за 6 секунд
      Clock fadeTimer;
      while (fadeTimer.getElapsedTime().asSeconds() < 3) {
        window.clear();

        // Изменение прозрачности текста
        Color textColor = text.getColor();
        textColor.a = static_cast<Uint8>(255 - 255 *
(fadeTimer.getElapsedTime().asSeconds() / 3.0));
        text.setColor(textColor);

        // Отображение анимации
```

```
            window.draw(text);

            window.display();
        }
        // Возврат к основному окну игры
        window.close();
        //originalWindow.display();
    }
```

г) enemy.cpp

```
#include "enemy.h"

Enemy::Enemy(Image &image, float X, float Y, int W, int H, string Name, string* MapMap)
:Entity(image, X,
                                                    Y, W, H, Name, MapMap){
    if (name == "EasyEnemy"){
        //Задаем спрайту один прямоугольник для
        //вывода одного игрока. IntRect – для приведения типов
        sprite.setTextureRect(IntRect(0, 0, w, h));
        direction = rand() % (3); //Направление движения врага задаём случайным образом
        //через генератор случайных чисел
        speed = 0.1;//даем скорость.этот объект всегда двигается
        dx = speed;
    }
}
void Enemy::checkCollisionWithMap(float Dx, float Dy)//ф-ция проверки столкновений с
картой
{
    for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты
        for (int j = x / 32; j < (x + w) / 32; j++)
        {
            if (TileMap[i][j] == '0')//если элемент - тайлик земли
            {
                if (Dy > 0) {
                    y = i * 32 - h; dy = -0.1;
                    direction = rand() % (3); //Направление движения врага
                }//по Y
                if (Dy < 0) {
                    y = i * 32 + 32; dy = 0.1;
                    direction = rand() % (3);//Направление движения врага
                }//столкновение с верхними краями
                if (Dx > 0) {
                    x = j * 32 - w; dx = -0.1;
                    direction = rand() % (3);//Направление движения врага
```

```
            }//с правым краем карты
            if (Dx < 0) {
               x = j * 32 + 32; dx = 0.1;
               direction = rand() % (3); //Направление движения врага
            }// с левым краем карты
         }
      }
}

void Enemy::SpawnCoin()
{
  for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты
     for (int j = x / 32; j < (x + w) / 32; j++)
     {
        if (TileMap[i][j] == ' ')//если элемент - пустое поле
        {
           TileMap[i][j]='s';
           break;
        }
     }
}

void Enemy::update(float time)
{
  if (name == "EasyEnemy"){//для персонажа с таким именем логика будет такой
     if (life) {//проверяем, жив ли герой
        switch (direction)//делаются различные действия в зависимости от состояния
        {
        case 0:{//состояние идти вправо
           dx = speed;
           CurrentFrame += 0.005*time;
           if (CurrentFrame > 3) CurrentFrame -= 3;
           sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 192, 96, 96));
           break;
        }
        case 1:{//состояние идти влево
           dx = -speed;
           CurrentFrame += 0.005*time;
           if (CurrentFrame > 3) CurrentFrame -= 3;
           sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 96, 96, 96));
           break;
        }
        case 2:{//идти вверх
           dy = -speed;
```

```
            CurrentFrame += 0.005*time;
            if (CurrentFrame > 3) CurrentFrame -= 3;
            sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 288, 96, 96));
            break;
          }
          case 3:{//идти вниз
            dy = speed;
            CurrentFrame += 0.005*time;
            if (CurrentFrame > 3) CurrentFrame -= 3;
            sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 0, 96, 96));
            break;
          }
          }
          x += dx*time; //движение по “X”
          checkCollisionWithMap(dx, 0);//обрабатываем столкновение по X
          y += dy*time; //движение по “Y”
          checkCollisionWithMap(0, dy);//обрабатываем столкновение по Y
          sprite.setPosition(x, y); //спрайт в позиции (x, y).
          if (health <= 0){ life = false; }//если жизней меньше 0, либо равно 0, то умираем
      }
    }
}
```

д) entity.cpp

```
#include "entity.h"

Entity::Entity(Image &image, float X, float Y, int W, int H, string Name, string* MapMap){
  x = X; y = Y; //координата появления спрайта
  w = W; h = H;
  name = Name;
  moveTimer = 0;
  dx = 0; dy = 0;
  speed = 0;
  CurrentFrame = 0;
  health = 100;
  life = true; //инициализировали логическую переменную жизни, герой жив
  texture.loadFromImage(image); //заносим наше изображение в текстуру
  sprite.setTexture(texture); //заливаем спрайт текстурой

  TileMap = MapMap;
}

FloatRect Entity::getRect(){//метод получения прямоугольника. его коорд, размеры (шир,высот).
```

```
  FloatRect FR(x, y, w, h); // переменная FR типа FloatRect
  return FR;
  //Тип данных (класс) "sf::FloatRect" позволяет хранить четыре координаты прямоугольника
  //в нашей игре это координаты текущего расположения тайла на карте
  //далее это позволит спросить, есть ли ещё какой-либо тайл на этом месте
  //эта ф-ция нужна для проверки пересечений
}
```

e) game.cpp

```
#include "game.h"
#include "arrmaps.h"

Game::Game(Image& im): //loadTextures(), //loadMaps(),
  window(VideoMode(WIDTH_MAP * 32, HEIGHT_MAP * 32), "Station Demo"),
  p(im, 100, 100, 70, 96, "Player1", map1.GetTileMap()),
  text("", font, 30), map1(ArrMap1, 1), map2(ArrMap2, 2),
  map3(ArrMap3, 3), map4(ArrMap4, 4)
{
  gameTime = 0;
  hpDownPlayerTimer = 0;//Переменная под время для неуязвимости игрока после получения урона
  createBulletsTimer = 0;//Переменная под время для задержки выстрела

  loadTextures();
}

Game::~Game() { //на всякий случай
  for (it = Bullets.begin(); it != Bullets.end(); it++)//проходимся по списку
     it = Bullets.erase(it);

  for (eit = enemies.begin(); eit != enemies.end(); eit++)//проходимся по списку
        eit = enemies.erase(eit);
}

void Game::run() {
  while (window.isOpen()) {
     handleEvents();
     update();
     draw();
     checkOptionForWindow();
  }
}
```

```
void Game::loadTextures() {
  font.loadFromFile("MP Manga.ttf");

  text.setColor(Color::Red);//покрасили текст в красный
  text.setStyle(Text::Bold);//жирный текст.

  //heroImage.loadFromFile("images/hero.png"); // загружаем изображение игрока
  //heroImage.createMaskFromColor(Color(255, 255, 255));

  vendingMachineImage.loadFromFile("images/vending.png");
  vendingMachineImage.createMaskFromColor(Color(255, 255, 255));

  easyEnemyImage.loadFromFile("images/enemy.png");
  easyEnemyImage.loadFromFile("images/enemy.png"); // загружаем изображение врага
  easyEnemyImage.createMaskFromColor(Color(255, 255, 255)); //убираем белый цвет

  BulletImage.loadFromFile("images/bullet.png");//загрузили картинку в объект
изображения
  BulletImage.createMaskFromColor(Color(255, 255, 255));
  BulletImage.createMaskFromColor(Color(0, 0, 0)); //убираем черный цвет

  map_image.loadFromFile("images/map_new.png");//загружаем файл для карты
  map.loadFromImage(map_image);//заряжаем текстуру картинкой
  s_map.setTexture(map);//заливаем текстуру спрайтом
  // Дополните этот метод, если есть еще текстуры
}

void Game::handleEvents() {
  Event event;
  while (window.pollEvent(event))
  {
    if (event.type == Event::Closed)
      window.close();

    //стреляем по нажатию клавиши "E"
    if (event.type == Event::KeyPressed)
    {
      if ((event.key.code == Keyboard::E) && (createBulletsTimer > FIRE_SPEED) &&
(p.life))
        //стреляем по нажатию клавиши "E"
        //если нарастили меньше 1 секунды, то пуля не рождается
      {
        Bullets.push_back(new Bullet(BulletImage, p.x+32, p.y+50, 16, 16, "Bullet",
                          p.state, map1.GetTileMap()));
```

```
            createBulletsTimer = 0;//обнуляем таймер
        }
        if (event.key.code == Keyboard::Space) { //обмен монет на жизнь по нажатию клавиши "space"
            //vm.exchangeCoins(p);
        }
    }
  }
}

void Game::update() {
  float time = clock.getElapsedTime().asMicroseconds();
  if (p.life) gameTime = gameTimeClock.getElapsedTime().asSeconds();//игровое время в
        //секундах идёт вперед, пока жив игрок. Перезагружать как time его не надо.
        //оно не обновляет логику игры
  clock.restart();
  time /= 700;

  createBulletsTimer += time; //наращиваем таймеры
  hpDownPlayerTimer += time;

  if (enemyAlsoCreatedOnMap){ //Если ещё не были созданы на текущей карте, то создаём
    //Заполняем список объектами врагами
    for (int i = 0; i < ENEMY_COUNT; i++)
    {
        float xr = 150 + rand() % 500; // случайная координата врага на поле игры по оси “x”
        float yr = 150 + rand() % 350; // случайная координата врага на поле игры по оси “y”
        //создаем врагов и помещаем в список
        enemies.push_back(new Enemy(easyEnemyImage, xr, yr, 96, 96, "EasyEnemy", map1.GetTileMap()));
        enemiesCount += 1; //увеличили счётчик врагов
    }
  enemyAlsoCreatedOnMap = false;
  }

  p.update(time);

  //оживляем врагов
  for (eit = enemies.begin(); eit != enemies.end(); eit++)
  {
    (*eit)->update(time); //запускаем метод update()
```

```
    }

    //оживляем пули
    for (it = Bullets.begin(); it != Bullets.end(); it++)
    {
        (*it)->update(time); //запускаем метод update()
    }

    //Проверяем список на наличие "мертвых" пуль и удаляем их
    for (it = Bullets.begin(); it != Bullets.end(); )//говорим что проходимся от начала до
конца
    {// если этот объект мертв, то удаляем его
        if ((*it)-> life == false) { it = Bullets.erase(it); }
        else it++;//и идем курсором (итератором) к след объекту.
    }

    //Проверяем список на наличие "мертвых" врагов и удаляем их
    for (eit = enemies.begin(); eit != enemies.end(); )//говорим что проходимся от начала до
конца
    {// если этот объект мертв, то удаляем его
        if ((*eit)-> life == false)
        {
            (*eit)-> SpawnCoin();
            eit = enemies.erase(eit);
        }
        else eit++;//и идем курсором (итератором) к след объекту.
    }

    //Проверка пересечения игрока с врагами
    //Если пересечение произошло, то "health -= 20". После того, как здоровье опустится
до 0,
    //игрок обездвиживается и выводится сообщение "you are lose"
    if (p.life == true){//если игрок жив
        for (eit = enemies.begin(); eit != enemies.end(); eit++){//бежим по списку врагов
            if ((p.getRect().intersects((*eit)->getRect())) && ((*eit)->name == "EasyEnemy"))
            {
                if (hpDownPlayerTimer>1000){
                    p.health -= 20;
                    cout << "you take the damage!\n";
                    hpDownPlayerTimer = 0;//обнуляем таймер
                }
            }
        }
    }
```

```
  //пересечение пули с врагом
  for (eit = enemies.begin(); eit != enemies.end(); eit++){//бежим по списку врагов
    for (it = Bullets.begin(); it != Bullets.end(); it++){//по списку пуль
      if (((*it)->getRect().intersects((*eit)->getRect())) &&
        ((*eit)->name == "EasyEnemy") && ((*it)->name == "Bullet"))
      {
        cout << "Exellent hit!\n";

        //при попадании пули у врага отнимается здоровье
        (*eit)-> health -= PLAYER_DAMAGE;
        if ((*eit)-> health <= 0) {
          (*eit)-> life = false;
          enemiesCount -= 1; //уменьшаем количество врагов в игре
          cout << "Enemy destroyed!\n";
        }
        (*it)-> life = false;
      }
    }
  }
}

void Game::draw() {
  window.clear();

  if (enemiesCount==0) p.killAllEnemies = true;

  changeMapLogic();

  ////////////////////Рисуем нужную карту и передаём её в существующие
объекты/////////////////
  switch (roomNumber)
  {
  case 1:
    map1.draw(&s_map, &window);
    for (eit = enemies.begin(); eit != enemies.end(); eit++)//говорим что проходимся от
начала до конца
      {(*eit)-> TileMap = map1.GetTileMap();}

    for (it = Bullets.begin(); it != Bullets.end(); it++)
      {(*it)-> TileMap = map1.GetTileMap();}

    p.TileMap = map1.GetTileMap();
    break;
```

```
  case 2: map2.draw(&s_map, &window);
    for (eit = enemies.begin(); eit != enemies.end(); eit++)//говорим что проходимся от
начала до конца
        {(*eit)-> TileMap = map2.GetTileMap();}

    for (it = Bullets.begin(); it != Bullets.end(); it++)
        {(*it)-> TileMap = map2.GetTileMap();}

    p.TileMap = map2.GetTileMap();
    break;
  case 3: map3.draw(&s_map, &window);
    for (eit = enemies.begin(); eit != enemies.end(); eit++)//говорим что проходимся от
начала до конца
        {(*eit)-> TileMap = map3.GetTileMap();}

    for (it = Bullets.begin(); it != Bullets.end(); it++)
        {(*it)-> TileMap = map3.GetTileMap();}

    p.TileMap = map3.GetTileMap();
    break;
  case 4: map4.draw(&s_map, &window);
    for (eit = enemies.begin(); eit != enemies.end(); eit++)//говорим что проходимся от
начала до конца
        {(*eit)-> TileMap = map4.GetTileMap();}

    for (it = Bullets.begin(); it != Bullets.end(); it++)
        {(*it)-> TileMap = map4.GetTileMap();}

    p.TileMap = map4.GetTileMap();
    break;
  }         //сюда вместо s_map можно просто подать указатель на другой спрайт,
           //и тогда карта будет выглядеть по-другому

  //объявили переменную здоровья и времени
  ostringstream playerHealthString, gameTimeString, playerScoreString;
  playerHealthString << p.health; gameTimeString << gameTime;//формируем строку
  playerScoreString << p.playerScore; //занесли в нее число очков, то есть формируем
строку
  text.setString("HP " + playerHealthString.str() + "\nTime: " +
              gameTimeString.str()+ "\nScore: " + playerScoreString.str());//задаем строку
тексту
  text.setPosition(40, 27);//задаем позицию текста
  window.draw(text);//рисуем этот текст
```

```
    window.draw(p.sprite);//рисуем спрайт объекта “p” класса “Player”
    //window.draw(vm.sprite);

    //рисуем врагов
    for (eit = enemies.begin(); eit != enemies.end(); eit++)
    {
        if ((*eit)->life) //если враги живы
            window.draw((*eit)->sprite); //рисуем
    }
    //рисуем пули
    for (it = Bullets.begin(); it != Bullets.end(); it++)
    {
        if ((*it)->life) //если пули живы
            window.draw((*it)->sprite); //рисуем объекты
    }

    window.display();
}

void Game::checkOptionForWindow(){
    ////////////////Экран смерти/////////////////
    if (p.life == false)
    {
        DeathAnimation deathAnimation(window);
        deathAnimation.playAnimation(window);
    }

    ////////////////Конец игры///////////////////////
    if (map3.isPassed && p.killAllEnemies){
        Ending ending(window);
        ending.playAnimation(window);
        cout << "You won!!!" << endl;
    }
}

void Game::changeMapLogic(){
    if (roomNumber != p.numberOfRoom) //эти флаги меняются только при смене комнаты
    {
        switch (roomNumber) { //p.numberOfRoom обновляется только если все враги в
комнате убиты.
        case 1: map1.isPassed = true;
            break;
        case 2: map2.isPassed = true;
            break;
```

```
        case 3: map3.isPassed = true;
            break;
        case 4: map4.isPassed = true;
            break;
        }

        roomNumber = p.numberOfRoom; //обновляем комнату

        switch (roomNumber) { //если эта комната была пройдена
        case 1: if (!map1.isPassed) {enemyAlsoCreatedOnMap = true; p.killAllEnemies = false;}
            break;
        case 2: if (!map2.isPassed) {enemyAlsoCreatedOnMap = true; p.killAllEnemies = false;}
            break;
        case 3: if (!map3.isPassed) {enemyAlsoCreatedOnMap = true; p.killAllEnemies = false;}
            break;
        case 4: if (!map4.isPassed) {enemyAlsoCreatedOnMap = true; p.killAllEnemies = false;}
            break;
        }
    }
}
```

ё) main.cpp

```
#include "game.h"

int main() {
    Image image;
    image.loadFromFile("images/hero.png"); // загружаем изображение игрока
    image.createMaskFromColor(Color(255, 255, 255));

    Game game(image);
    game.run();

    return 0;
}
```

ж) map.h

```
#include "map.h"

Map::Map(string abs[HEIGHT_MAP], int numb){
    for (int i=0; i<HEIGHT_MAP; i++){
        TileMap[i]=abs[i];
    }
    isPassed = false;
    number = numb;
}
```

```
Map::Map(int numb) { isPassed = false; number = numb ;}

void Map::randomMapGenerate(){//рандомно расставляем камни
  int randomElementX = 0;//переменная для хранения случайного элемента по
горизонтали
  int randomElementY = 0;//переменная для хранения случайного элемента по вертикали
  int countStone = 5;//количество камней
  while (countStone > 0){
    randomElementX = 1 + rand() % (WIDTH_MAP - 1);//псевдослучайное значение по
“x” от 1 до
    //ширина карты-1. Ограничение введено чтобы не получать числа бордюра карты
    randomElementY = 1 + rand() % (HEIGHT_MAP - 1);//по “y”
    if (TileMap[randomElementY][randomElementX] == ' ') {//если встретили символ
пробел,
      TileMap[randomElementY][randomElementX] = 's'; //то ставим туда камень.
      countStone--;
    }
  }
}

string* Map::GetTileMap(){
  return TileMap;
}

void Map::draw(Sprite* s_map, RenderWindow* window){ //сюда можно подать
указатель на другой спрайт,
                                    //тогда карта будет выглядеть по-другому
  for (int i = 0; i < HEIGHT_MAP; i++)
    for (int j = 0; j < WIDTH_MAP; j++)
    {
      if (TileMap[i][j] == ' ') s_map->setTextureRect(IntRect(0, 0, 32, 32));
      if (TileMap[i][j] == 's') s_map->setTextureRect(IntRect(32, 0, 32, 32));
      if (TileMap[i][j] == '0') s_map->setTextureRect(IntRect(64, 0, 32, 32));
      if (TileMap[i][j] == 'f') s_map->setTextureRect(IntRect(96, 0, 32, 32));//цветок
      if (TileMap[i][j] == 'h') s_map->setTextureRect(IntRect(128, 0, 32, 32));//сердце
      if (TileMap[i][j] == '?') s_map->setTextureRect(IntRect(162, 0, 32,
32));//вертикальная дверь, верх
      if (TileMap[i][j] == '!') s_map->setTextureRect(IntRect(193, 0, 32,
32));//вертикальная дверь, низ
      if (TileMap[i][j] == '(') s_map->setTextureRect(IntRect(225, 0, 32,
32));//горизонтальная дверь, лево
      if (TileMap[i][j] == ')') s_map->setTextureRect(IntRect(257, 0, 32,
32));//горизонтальная дверь, право
```

```
        if (TileMap[i][j] == 'd') s_map->setTextureRect(IntRect(257, 0, 32, 32));//финальная
дверь
        s_map->setPosition(j * 32, i * 32);
        (*window).draw(*s_map);
      }
}
```

з) player.cpp

```
#include "player.h"

Player::Player(Image &image, float X, float Y, int W, int H, string Name, string* MapMap)
            :Entity(image, X, Y, W, H, Name, MapMap){
  numberOfRoom = 1; //начальная комната - 1
  killAllEnemies = false;
  playerScore = 0; //монеты
  state = stay;
  if (name == "Player1"){
    //Задаем спрайту один прямоугольник для
    //вывода одного игрока. IntRect – для приведения типов
    sprite.setTextureRect(IntRect(0, 0, w, h));
  }
}

/*void Player::SpawnCoin() //метод спавна монет (не используется)
{
  //пусто
}*/

void Player::control(){
  if (Keyboard::isKeyPressed(Keyboard::A)) {
    state = left;
    speed = 0.1;
  }
  if (Keyboard::isKeyPressed(Keyboard::D)) {
    state = right;
    speed = 0.1;
  }
  if (Keyboard::isKeyPressed(Keyboard::W)) {
    state = up;
    speed = 0.1;
  }
  if (Keyboard::isKeyPressed(Keyboard::S)) {
    state = down;
    speed = 0.1;
```

```
  }
}

void Player::checkCollisionWithMap(float Dx, float Dy) {
  for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты
    for (int j = x / 32; j<(x + w) / 32; j++)
    {
      if (TileMap[i][j] == '0')//если элемент тайлик земли
      {
        if (Dy > 0) { y = i * 32 - h; dy = 0; }//по Y
        if (Dy < 0) { y = i * 32 + 32; dy = 0; }//столкновение с верхними краями
        if (Dx > 0) { x = j * 32 - w; dx = 0; }//с правым краем карты
        if (Dx < 0) { x = j * 32 + 32; dx = 0; }// с левым краем карты
      }
      if (TileMap[i][j] == 's') {
        playerScore++; //если взяли монету
        TileMap[i][j] = ' ';
      }
      if (TileMap[i][j] == 'f') {
        health -= 40;//если взяли ядовитый цветок
        TileMap[i][j] = ' ';//убрали цветок
      }
      if (TileMap[i][j] == 'h') {
        health += 20;//если взяли сердечко
        TileMap[i][j] = ' ';//убрали сердечко
      }
    }
}

void Player::checkCollisionWithDoor(){
  int numb = numberOfRoom;
  int nextRoom = -1;  // Инициализация переменной для хранения следующей комнаты
  int oldI, oldJ; //переменные для запоминания позиции в каждой комнате

  for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты,
соприкасающихся с игроком
    for (int j = x / 32; j<(x + w) / 32; j++)
    {
      int checkDoor = -1;
      // Проверка, находится ли игрок рядом с дверью
      if ((TileMap[i][j + 1] == '?' || TileMap[i][j - 1] == '?' ||
        TileMap[i + 1][j] == '?' || TileMap[i - 1][j] == '?')||
        (TileMap[i][j + 1] == '!' || TileMap[i][j - 1] == '!' ||
        TileMap[i + 1][j] == '!' || TileMap[i - 1][j] == '!')) //право лево
```

```
        checkDoor = 0;
if ((TileMap[i][j + 1] == '(' || TileMap[i][j - 1] == '(' ||
    TileMap[i + 1][j] == '(' || TileMap[i - 1][j] == '('||
    (TileMap[i][j + 1] == ')' || TileMap[i][j - 1] == ')' ||
    TileMap[i + 1][j] == ')' || TileMap[i - 1][j] == ')')) //верх низ
        checkDoor = 1;
if (checkDoor != -1){
// Логика для определения следующей комнаты
switch (numb) {
    case 1:
        nextRoom = 2; //дверь всегда только справа
        oldJ = WIDTH_MAP - 2;
        oldI = HEIGHT_MAP/2 + 2;
        break;
    case 2:
        if (!checkDoor) { //дверь слева
            nextRoom = 1;
            oldJ = 5;
            oldI = HEIGHT_MAP/2 + 2;
        }
        else { //дверь снизу
            nextRoom = 3;
            oldJ = WIDTH_MAP/2 + 2; // x
            oldI = HEIGHT_MAP - 2; // y
        }
        break;
    case 3:
        if (checkDoor) { //дверь сверху
            nextRoom = 2;
            oldJ = WIDTH_MAP/2 + 2;
            oldI = 5;
        }
        else {//дверь справа
            nextRoom = 4;
            oldJ = WIDTH_MAP - 2;
            oldI = HEIGHT_MAP/2 + 2;
        }
        break;
    case 4:
        nextRoom = 3; //дверь всегда только слева
        oldJ = 5;
        oldI = HEIGHT_MAP/2 + 2;
        break;
    }
```

```
            }
        }

    // Обработка перехода в следующую комнату
    if (nextRoom != -1) {
        if (killAllEnemies){
            numberOfRoom = nextRoom;  // Обновление номера текущей комнаты

            int oldX = (oldJ * 32);  // X-координата центра двери
            int oldY = (oldI * 32);  // Y-координата центра двери

            // Вычисление новых координат в зависимости ОТ СТАРЫХ
            x = (WIDTH_MAP)*32 - oldX;
            y = (HEIGHT_MAP)*32 - oldY;

            cout << "You're in front of the door" << endl;
        }
        else {cout << "You should kill the enemies first!" << endl;}
    }
}

void Player::update(float time) //метод "оживления/обновления" объекта класса.
{
    if (life) {//проверяем, жив ли герой
        control();//функция управления персонажем
        switch (state)//делаются различные действия в зависимости от состояния
        {
        case right:{//состояние идти вправо
            dx = speed;
            CurrentFrame += 0.005*time;
            if (CurrentFrame > 3) CurrentFrame -= 3;
            sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 195, 96, 96));
            break;
        }
        case left:{//состояние идти влево
            dx = -speed;
            CurrentFrame += 0.005*time;
            if (CurrentFrame > 3) CurrentFrame -= 3;
            sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 100, 96, 96));
            break;
        }
        case up:{//идти вверх
            dy = -speed;
            CurrentFrame += 0.005*time;
```

```
            if (CurrentFrame > 3) CurrentFrame -= 3;
            sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 292, 96, 96));
            break;
        }
        case down:{//идти вниз
            dy = speed;
            CurrentFrame += 0.005*time;
            if (CurrentFrame > 3) CurrentFrame -= 3;
            sprite.setTextureRect(IntRect(96 * int(CurrentFrame), 0, 96, 96));
            break;
        }
        case stay:{//стоим
            dy = speed;
            dx = speed;
            break;
        }
        }
        x += dx*time; //движение по “X”
        checkCollisionWithMap(dx, 0);//обрабатываем столкновение по X
        y += dy*time; //движение по “Y”
        checkCollisionWithMap(0, dy);//обрабатываем столкновение по Y

        checkCollisionWithDoor();

        speed = 0; //обнуляем скорость, чтобы персонаж остановился.
        //state = stay;

        sprite.setPosition(x, y); //спрайт в позиции (x, y).
        if (health <= 0)
        {
            life = false; cout << "You're dead!" << endl;
        }//если жизней меньше 0, либо равно 0, то умираем
    }
}
```

и) vendingmachine.cpp

```
#include "vendingmachine.h"

VendingMachine::VendingMachine(Image &image, float X, float Y, int W, int H, string
Name, string* MapMap)
            :Entity(image, X, Y, W, H, Name, MapMap){

        price = 5;
```

```
    if (name == "vm"){
      //Задаем спрайту один прямоугольник для
      sprite.setTextureRect(IntRect(0, 0, w, h));
    }
  }

bool VendingMachine::canAfford(Player& player) {
  return (player.playerScore >= price);
}

void VendingMachine::exchangeCoins(Player& player) {
  if (canAfford(player)) {
    player.playerScore -= price;
    //player.gainLife();
  }
}

void VendingMachine::update(float time) //метод "оживления/обновления" объекта класса.
{
  if (name == "vm")
    sprite.setPosition(x, y);
}
```