Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное игровое средство «DOODLE JUMP»**

|  |  |  |
| --- | --- | --- |
| Студент |  | Е.А. Глазков |
| Руководитель |  | Д.С. Шулицкий |

Минск 2019

СОДЕРЖАНИЕ

[Введение](#_Toc513665878) 5

[1 Анализ предметной области](#_Toc513665879) 7

[1.1 Обзор аналогов](#_Toc513665880) 7

[1.2 Постановка задачи](#_Toc513665881) 10

[2 Разработка программного средства](#_Toc513665882) 11

[2.1 Структура программы](#_Toc513665883) 11

[2.2 Интерфейс программного средства](#_Toc513665884) 11

[2.3 Работа со звуковыми ресурсами](#_Toc513665885) 16

[2.4 Структура игры](#_Toc513665888) 18

[2.4.1 Инициализация](#_Toc513665886) 19

[2.4.2 Игровой цикл](#_Toc513665887) 21

[3 Тестирование программного средства](#_Toc513665890) 25

[4 Руководство пользователя](#_Toc513665891) 27

[4.1 Интерфейс программы](#_Toc513665892) 27

[4.2 Инструкция к сбросу рекорда](#_Toc513665893) 31

[Заключение](#_Toc513665894) 32

[Список использованных источников](#_Toc513665895) 33

[Приложение А. Исходный код программы](#_Toc513665896) 34

ВВЕДЕНИЕ

Doodle Jump — видеоигра, разработанная и выпущенная компанией Lima Sky для следующих платформ: iOS, BlackBerry, Android и Nokia S60. Для платформы iOS игра вышла 6 апреля 2009 года, а для платформ Android и Blackberry — 2 марта 2010 года. С самого момента выхода игра приобрела огромную популярность. Сделанная на примере игры PapiJump от SunFlat игра обладает простой, рисованной, но забавной графикой, а также несложным управлением. К 25 июня 2010 года было продано более 5 миллионов копий игры. Каждый день игру скачивают около 28000 человек.

Главная цель игры — помочь «Дудлеру» (Doodle) добраться как можно выше по платформам и набрать как можно большее количество очков. Героем можно управлять с помощью всего трёх кнопок на мобильном устройстве (влево, вправо, центральная кнопка). В версии для устройств с акселерометром управление происходит с помощью наклона устройства в нужном направлении.

Из-за столь давнего анонса проекта, он успел появиться и на не поддерживающемся ныне Symbian для кнопочных и сенсорных телефонов Nokia. В свое время вышла доработанная версия Deluxe, включающая в себя возможность менять локации.

Вскоре эпоха кнопочных телефонов кончилась: пришли Android, iOS и Windows Phone. В 2015 году в магазинах приложений крупных ОС появился Doodle Jump DC: специальное приложение на основе привычного «Дудлика», где помимо простых прыжков нужно собирать кристаллы и бороться с суперзлодеями Marvel. Валюта в приложении разработана, чтобы покупать дополнительные апгрейды: вертолет, прыжок, удвоенный удар и т.п.

В Doodle Jump, цель состоит в руководстве четвероногим существом, похожим на пылесосик, имя которого Дудлик, на пути по бесконечной серии платформ без падений. Игроки могут получить короткий толчок от различных объектов, таких, как пропеллер-шляпы, реактивные ранцы, ракеты, пружины или батуты. Есть также монстры и НЛО, в которых Дудлер должен стрелять или прыгать на них, ликвидируя. Стрельба осуществляется нажатием на различные части экрана. Не существует окончания игры, но конец каждой игры происходит, когда игрок падает (по достижении нижней части экрана), попадает на монстра, оказывается вовлеченным в чёрную дыру, или похищенным НЛО. Новейшая обновленная версия игры Doodle Jump: Deluxe позволяет игрокам выбирать между версиями игры (обычный, Кладбище, Рождество, Джунгли, Космос, Кубок мира, Пасха, Под водой, Хэллоуин, Ниндзя, Арктика или Пиратский корабль.) без изменения их имени.

Любопытно, что в приложение встроен статистический модуль, собирающий данные о количестве проведенных игр, о проведенном времени в приложении, количестве убитых монстров и НЛО, а также количестве смертей от мутантов и летающих тарелок.

Популярность этой, казалось бы, примитивной игры кроется в забавной графике и понятном исполнении. Стоит отличать оригинальный стиль от неумения программировать интерфейс: в этом заключается будущий успех приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Обзор версий и аналогов

У данной серии игр существует ряд версий со своей уникальной стилистикой а так же много аналогов, похожих по логике и геймплэю, однако отличающихся по стилистике и сюжету.

Dronius – увлекательная игра о маленьком беспилотнике, который остался в полном одиночестве после межпланетной войны. Тем не менее сдаваться беспилотник не собирается. Он хочет найти жизнь, которая, возможно, осталась где-нибудь на бескрайних просторах Солнечной системы. Соответственно, основной миссией корабля является исследование огромного по своим масштабам пространства, пролетая огромный путь от Солнца до Плутона.



Рисунок 1.1 – Dronius

Doodle Jump: Christmas Special — специальный выпуск игры Doodle Jump. Здесь представлена новогодняя тематика.Удивительно красивая специальная Рождественская версия одного из самых захватывающих и пользующихся спросом приложений iOS всего времени, Doodle Jump!

"Lima Sky сделало его снова! Это - действительно забавная игра, чтобы играть! Необходимая вещь для любого поклонника Doodle Jump! " - обзор iTunes Вы можете помочь Каракуле подпрыгнуть полностью в Северный полюс? Новая ошеломляющая графика характерных особенностей Doodle Jump Christmas! Все новые пути и проблемы, все новые монстры, очень прикольная ракета, новый загадочный монстр, который заморозит Вас, если Вы не будете осторожны, и много чего другого.



Рисунок 1.2 – Doodle Jump: Christmas Special

Doodle Jump DC Super Heroes - одна из версий игры "Doodle Jump".

Персонаж представлен в виде Batman, различные варианты скинов Batman можно получить, проходя уровни. Монстры делятся на 3 категории: воришки(2 вида), прислужники и "боссы" - персонажи из комиксов про Бэтмена. Валюта в игре представлена кристаллами, их можно тратить на прокачку: щита, джетпака, батута и вертолета Бэтмена. В игре доступна опция сохранения(в начале бесплатно, потом за кристаллы).



Рисунок 1.3 – Doodle Jump DC Superheroes

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработка программного игрового средства «Doodle Jump».

В программном игровом средстве планируется реализовать следующие геймплэйные особенности:

* Набор очков, фиксирование рекордного счета;
* Враг монстр;
* Реактивный ранец;
* Ложные панели.
* Телепортация при выходе за пределы карты

Логическая составляющая приложения будет включать в себя:

* Инициализация игры;
* Основной цикл анимации;
* Обновление карты;
* Звуковое сопровождениe;
* Окно с итогами игровой сессии.

Для разработки программного средства будет использоваться язык программирования C++, операционная система Mac OS X Mojave и среда разработки Apple XCode 10.1.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура программы

Данное приложение состоит из одного лишь модуля :

– Main – модуль, хранящий в себе логику игры, инициализацию карты,

анимацию и загрузку текстур и звуковых дорожек;

В ходе разработки программного продукта была использована среда разработки Apple XCode 10.1 , на это есть ряд причин. XCode 10.1 предоставляет огромные возможности для редактирования исходного кода, лексического, синтаксического и даже семантического анализа написанного кода в реальном времени, и удобную подсветку синтаксиса. Недостатком же является возможность простого создания интерфейса исключительно для языков Objective C и Swift. Именно поэтому для разработки клиентской части была выбрана библиотека SFML.

* 1. Интерфейс программного игрового средства

Внешний вид и удобность в использовании являются одними из главных критериев качества программного средства. Поэтому управление персонажем нужно сделать не только простым но еще и красиво анимированным. Клиентская часть разработана с использованием библиотеки SFML для быстрого доступа к компонентам, реализующих графический интерфейс.

В качестве текстур были выбраны фотографии в формате .png, предварительно обработанные в Adobe Photoshop CS6 для корректного взаимодействия между собой

Далее рассмотрим основные текстуры из нашего проекта

1. Дудл

Рисунок 2.1 – Главный герой

.

Тот самый четвероногий персонаж, которому предстоит карабкаться вверх. Для него использованы две текстуры(правая и левая), для того, чтобы игрок понимал, в какую сторону в данный момент пресонаж осуществляет движение. Данная текстура Дудла соответствует оригинальной версии игры Doodle Jump.

1. Монстр



Рисунок 2.2 - Монстр

Данный персонаж представляет для главного героя наибольшую опасность, так как при контакте с ним Дудл не может дальше прыгать и просто умирает. Стоит отметить, что данная текстура монстра не является оригинальной текстурой из игры Doodle Jump (там монстров было огромное множество) а скорее напоминает призрака из известной игры PacMan(в этой игре он так же выполняет роль антагониста).

1. Платформа



Рисунок 2.3 - Платформа

Данная текстура позволяет главному герою взбираться наверх по карте.

Ее внешний вид так же не является оригинальной версией платформы из игры Doodle Jump и скорее напоминает аналог из игры Terraria. Ее особенность в том что чтобы Дудлу от нее оттолкнуться ему нужно прыгнуть именно на нее, а не просто с ней законтактировать.

1. Ложная платформа





Рисунок 2.4 - Два состояния ложной платформы

Для данной платформы было использовано две текстуры, так как она имеет два состояния: целое и сломанное.

Как и в случае с обычной платформой, чтобы с ней взаимодействовать, Дудл должен прыгнуть на нее, но в случае с ложной платформой Дудл не получит никакого ускорения, а платформа перейдет в сломанное состояние. Отличительные признаки этой платформы: черный цвет, лава, разлом по середине.

1. Реактивный ранец

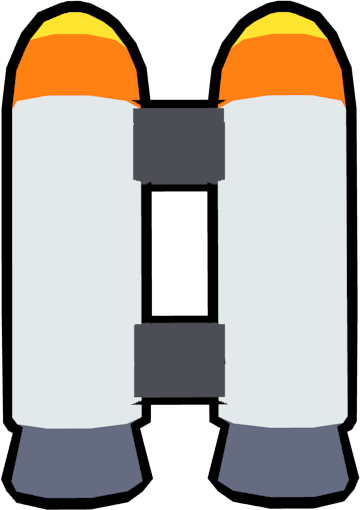


Рисунок 2.5 - Реактивный ранец

Реактивный ранец или проще говоря “ДжетПак”. Текстура была взята из оригинальной версии игры Doodle Jump и выполняет функцию быстрого перемещения и набора высоты. В режиме полета главный герой неуязвим, однако сразу после окончания действия реактивного ранца монстр сразу может убить Дудла. Встречается он на карте реже чем остальные объекты. Взаимодействие с реактивным ранцем в режиме полета продлит ваше время полета.

Для того, чтобы игрок понимал, когда он находится в режиме полета, для Дудла была взята еще одна текстура где он с реактивным ранцем на спине. После окончания действия режима полета Дудл снова приобретет свою обычную текстуру.

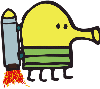
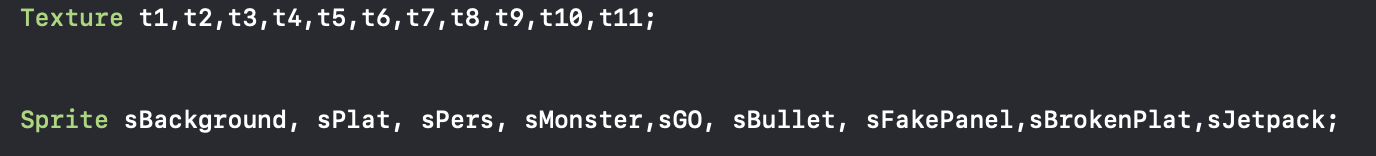


Рисунок 2.6 – Текстура Дудла в режиме полета

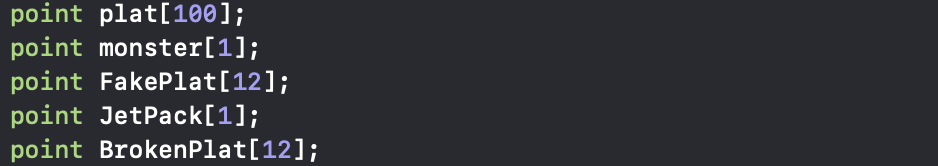
В качетве фона в своем игровом средстве я использовал обычный фон синего цвета. В оригинальной версии игры используется фон в виде листа тетради в клетку.

Работа с текстурами в программе имеет следующий принцип:

1. Объявляются текстуры и их буфферы.

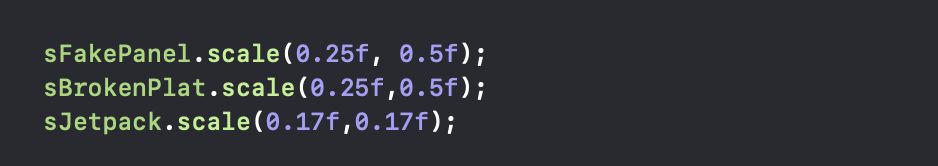


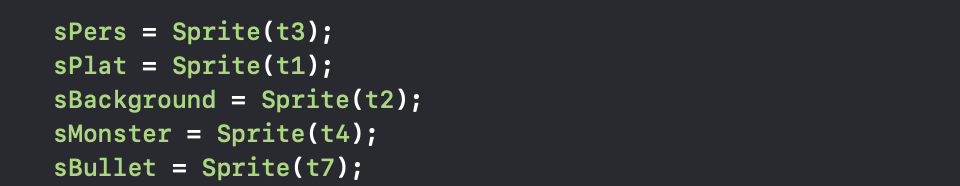
1. Объявляются указатели на координаты

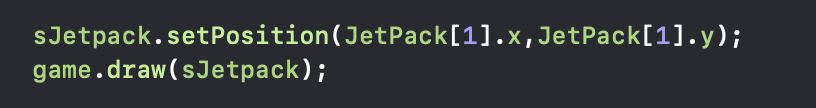


1. Текстуры загружаются из файлов и при необходимости масштабируются





1. Буфферам присваиваются текстуры 
2. Буфферу устанавливается координата согласно указателю и игра рисует тектуру буффера по заданным координатам.

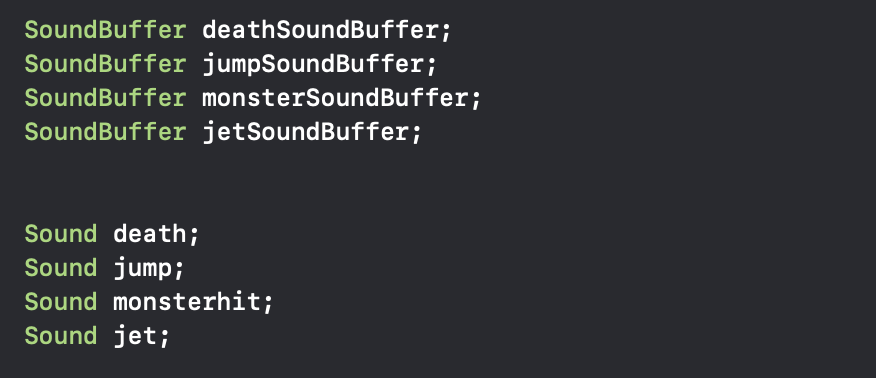


* 1. Работа со звуковыми ресурсами

Программное средство взаимодействует с определенным набором ресурсов, обеспечивающих некоторые звуковые эффекты.

Управление звуком реализовано при помощи библиотеки SFML и оно реализовывается очень схоже с текстурами:

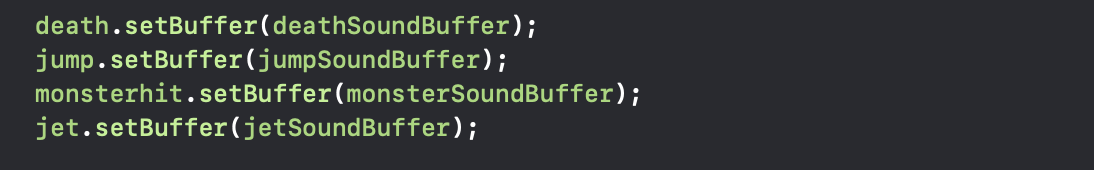
1. Инициализируются звуки и звуковые буфферы



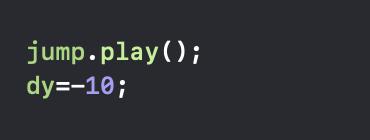
1. В буфферы загружаются звуковые файлы. SFML поддерживает только форматы .wav и .ogg



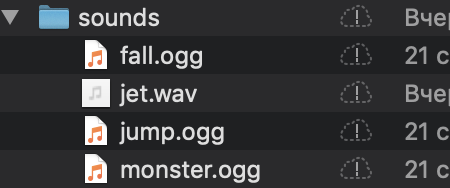
1. Звукам устанавливаются буфферы



1. Звук проигрывается

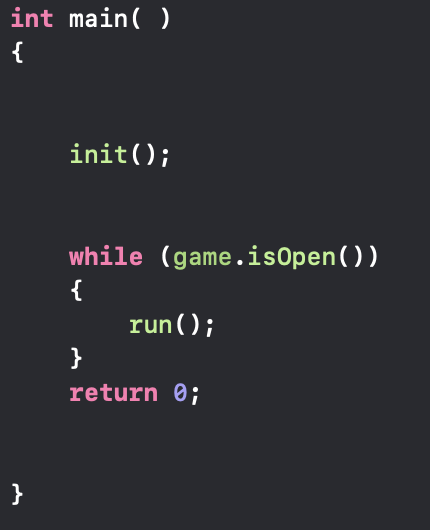


Используемые звуковые файлы были взяты из известной игры Minecraft, а так же из интернета.



* 1. Структура игры

Главный модуль вызывает процедуру инициализации карты и далее идет основной цикл анимации, который так же вынесен в отдельную подпрограмму.



* + 1. Инициализация

Основная задача данного структурного блока – загрузка всех нужных файлов(звуков и текстур) и настройка первоначального положения карты .

Главное условие для плавной анимации и комфортной игры – показатель FPS(Frame per second) количество кадров в секунду. Удовлетворительным показателем для ПК-гейминга принято считать 60, на других платформах по типу игровых консолей этот показатель должен быть не ниже 30. Вопреки мнению о том, что человеческий глаз не может различить разницу в FPS выше 24, разница в 30 FPS чувствуется.

Для начала нам нужно протестировать загрузку CPU при запущенной игре и понять, насколько она требовательна к железу. Тестируемое железо – Intel Core i5 3.1 GHz и интегрированная видеокарта Intel HD Graphics 650. Второй пункт нам не важен так как мы не настраиваем распределение мощности на видеокарту.

Получаем следующие показатели:

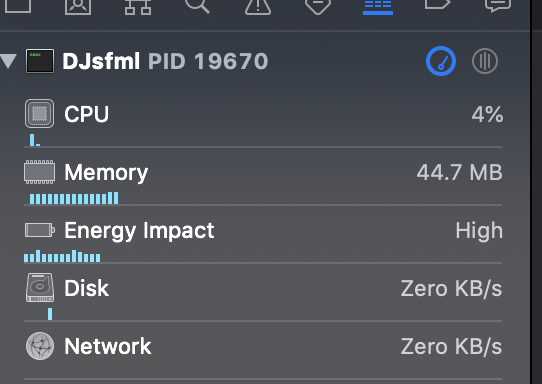
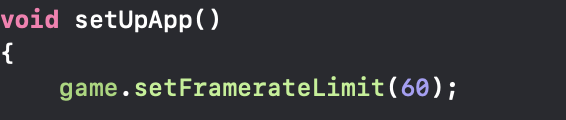


Рисунок 2.7 – показатели потребления мощности

Из этого можно сделать вывод, что наша игра почти не нагружает процессор, и мы можем спокойно ставить значение 60 FPS.



Далее мы должны произвести выгрузку наших файлов. Как это реализуется, мы уже описали в пунктах «Интерфейс» и «Работа со звуковыми файлами».

Далее идет настройка начального положения карты. Здесь у нас есть вледующие задачи:

1. Установить фон
2. Сгенерировать платформы
3. Сгенерировать ложные платформы
4. Сгенерировать главного героя

Монстры и реактивные ранцы генерируются после набора определенной высоты. Так же до набора определенной высоты Дудл не может умереть, разбившись о землю.

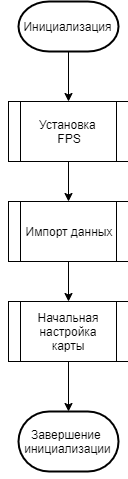


Рисунок 2.8 – Блок-схема инициализации

* + 1. Игровой цикл

Подпрограмма игрового цикла вызывается из главного модуля main после инициализации и имеет название «run».

Цикл так же разделен на несколько частей (его блок-схема будет вынесена на главный лист A1.

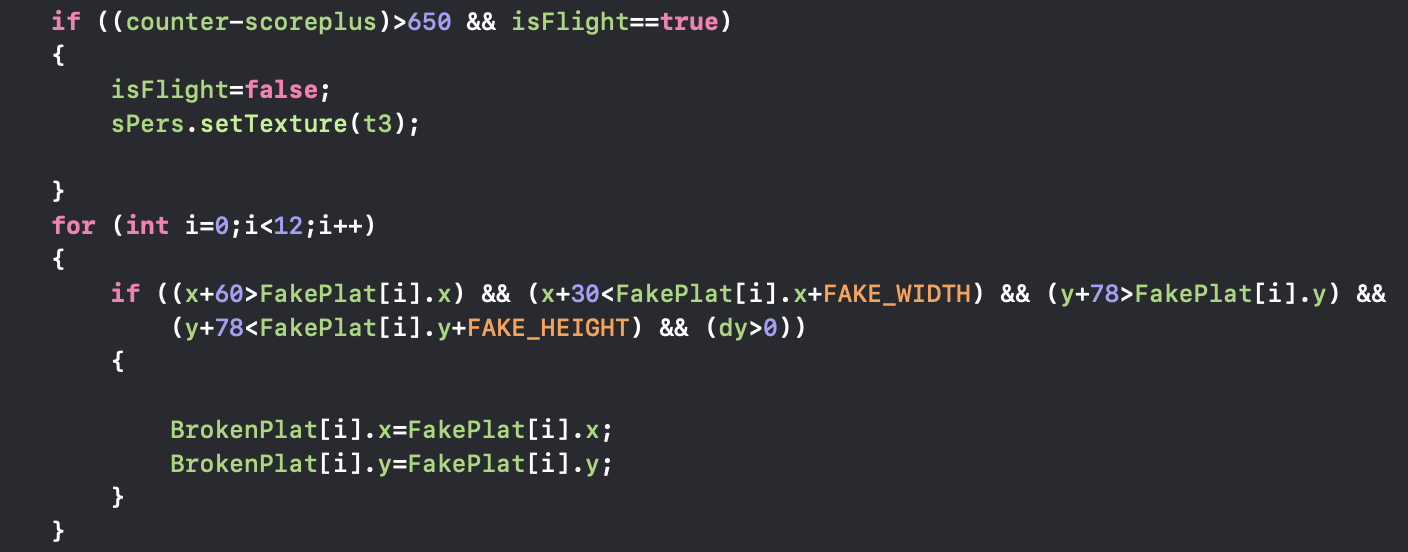
1. Управление
2. Проверка контакта с объектами
3. Генерирование новых текстур
4. Отрисовка
5. В случае смерти персонажа вызывает окно конца игры и передает туда значение нашего счета.

Управление реализовано с помощью встроенного в SFML обработчика нажатий.



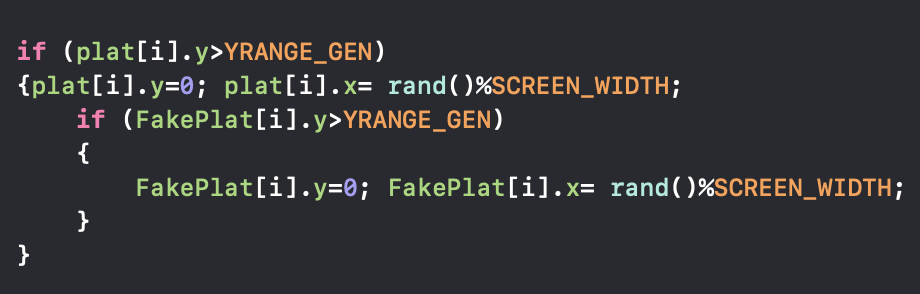
Проверка контакта с объектами вынесена в отдельную подпрограмму

«checktouch». Проверка касания для каждого объекта своя т к например условие правильного контакта с платформами нужно чтобы Дудл приземлился именно на них, для монстра и реактивного ранца это условие не выполняется. Так же важную роль играют флаги, которые показывают состояния некоторых объектов. Например флаг isFlight показывает состояние Дудла в полете, и в этом состоянии Дудл неуязвим и принимает текстуру с ранцем на спине.



Следующая часть игрового цикла - генерирование новых текстур.

Эта часть выделена в отдельную подпрограмму «reRender». Условие ее вызова – выход дудла за пределы рендера. Эта подпрограмма смотрит, вышел ли какой-нибудь уже отрисованный объект за пределы карты, и если да, то генерирует новые координаты для этой текстуры.



Генерирование объектов это частичный рандом. Причем для некоторых объектов методом рандома выбираются не только координаты а и еще непосредственно то, будут ли они генерироваться.

Касаемо платформ, карта условно разбита на 4 зоны по оси Y. Это сделано для того, чтобы Дудл всегда имел возможность подняться выше.

У некоторых текстур есть ошибочные значения координат, при которых они отрисовываться не будут, для этого в подпрограмме, отвечающей за отрисовку предусмотрена проверка.

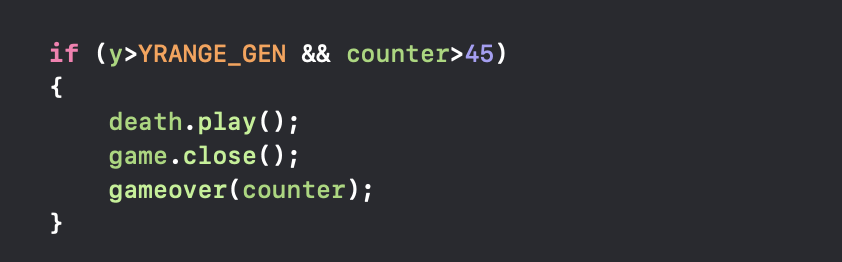
Следующий этап – отрисовка.

Подпрограмма –«drawing»

На этом этапе подпрограмма непосредственно берет значения координат для каждого объекта, берет сами текстуры и перерисовывает их на карту. Для тех текстур, которые на карте представлены не в одном экземпляре, предусмотрены циклы, которые проходят по целому массиву координат.

Сама реализация отрисовки уже описана в пункте «Интерфейс»

Заключительная часть игрового цикла – окно конца игры. (подпрограмма gameover) Оно вызывается при смерти персонажа. Условия смерти – персонаж столкнулся с монстром, персонаж упал и достиг конца карты. При это у нас в цикле фигурирует переменная counter. Она и отвечает за наш конечный счет, соответственно она передается в подпрограмму окна конца для последующего отображения на экран.



Само окно конца игры может иметь два состояния – обычное отображение счета и новый рекорд.

Стоит отметить что рекорды хранятся в отдельном файле «scores.txt», поэтому процедура gameover вызывает отдельную подпрограмму для перевода рекорда из типа Int в тип String.

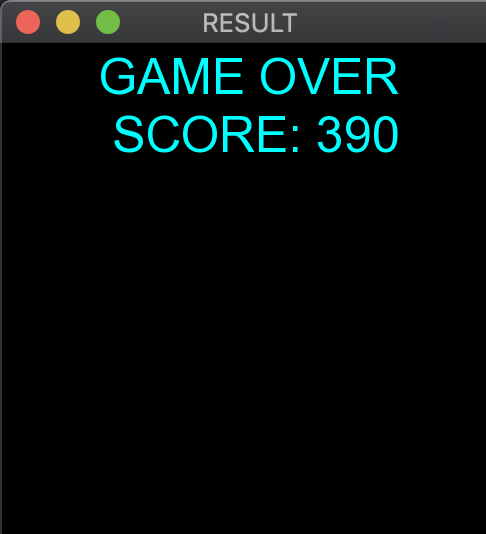


Рисунок 2.9 - Экран конца игры

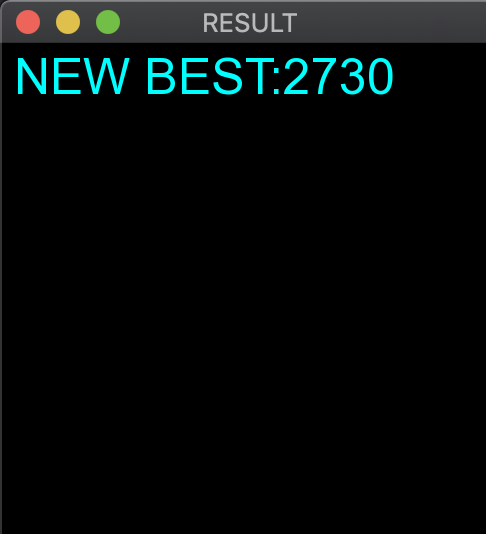


Рисунок 2.10 – Экран конца игры с новым рекордом

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

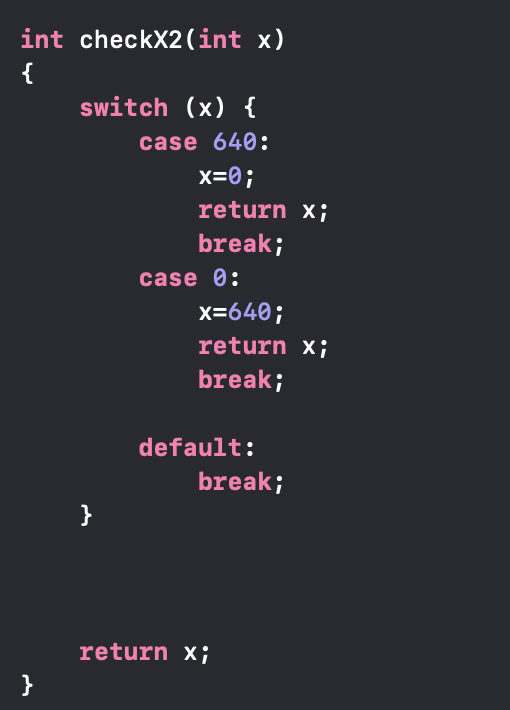
В ходе тестирования приложения никаких недостатков или ошибок работы программного игрового средства выявлено не было. Однако стоит отметить что данное программное игровое средство достаточно сильно отличается от оригинальной версии игры doodle jump, но это лишь вопрос концептуальных доработок и многообразия карт и объектов.

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Прохождение текстуры панели при имеющимся ускорении | Персонаж не оттолкнется от панели | Персонаж не оттолкнулся от панели |
| 2. | Персонаж достиг конца карты(упал), но еще не доходил до границы рендера | Персонаж не погиб | Персонаж не погиб |
| 3. | Контакт с монстром в состоянии полета на реактивном ранце | Персонаж не погиб | Персонаж не погиб |
| 4. | Контакт с реактивным ранцев в состоянии полета | Продление действия реактивного ранца | Продление действия реактивного ранца |
| 5. | Контакт с ложной панелью при имеющимся ускорении | Панель не сломается | Панель не сломалась |
| 6. | Контакт с ложной панелью при падении | Персонаж продолжит падать, панель сломается | Персонаж продолжил падать, панель не сломалась |
| 7. | Достижение левого конца карты | Персонаж телепортируется в правый конец карты | Персонаж телепортировался в правый конец карты |
| 8. | Достижение правого конца карты | Аналогично с предыдущим пунктом | Аналогично с предыдущим пунктом |
| 9. | Контакт с монстром | Смерть персонажа | Смерть персонажа |
| 10. | Достижение конца карты персонажем после пересечения границы рендера | Смерть персонажа | Смерть персонажа |

Разработка программного игрового средства «Doodle Jump» велась с использованием системы контроля версий GitHub, позволившая сохранять состояние программы на каждом отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений. Также можно добавлять к каждому этапу новые файлы, или удалять устаревшие варианты. После накопления определённого количества групп изменений, их следует отправить на удалённый репозиторий, где видна вся история приложения и разница между каждым новым «коммитом».

Была обнаружена проблема при телепортации с правого края карты на левый, при этом при достижении левого конца карты телепортация проходила успешно. Ошибка была справлена путем добавления отдельной подпрограммы котоаря использовала конструкцию «Switch» и возвращала обновленные координаты персонажа.



Второй сложностью стала внезапная отрисовка монстра, из-за чего игрок не мог подготовиться к его появлению и умирал. Эта проблема была решена простым уменьшением границы рендера. После этого монстр стал прорисовываться в определенной дальности от персонажа и игрок стал успевать предпринимать определенные меры для избегания контакта с монстром.

Ещё одной проблемой стало невозможное взаимодействие программы с .mp3 файлами, т к в интернете довольно сложно найти подходящие файлы в формате .wav и .ogg. Однако после многочисленных попыток найти бесплатные ресурсы, я скачал саундпак из игры minecraft и уже из него выбрал подходящие звуковые файлы.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Интерфейс программы

Интерфейс программного средства будет знаком всем, кто в детстве играл в оригинальную версию игры «Doodle Jump». Только многим, кто играл в эту игру уже на современных телефонах будет непривычно управление кнопками, тк в версиях этой игры для iOS и Android управление базировалось на акселерометре и гироскопе (путем наклона устройства влево и вправо). Так же будет непривычно то, что персонаж не получает ускорения при верхнем контакте с монстром, а просто умирает.

Управление персонажем происходит с помощью нажатия кнопок «Left» и «Right»



Рисунок 4.1 – Размещение управляющих кнопок на клавиатуре MacBook

На данном рисунке (4.2) мы видим на карте монстра, контакта с ним нужно избегать. Для этого выбираем оптимальный путь наверх.

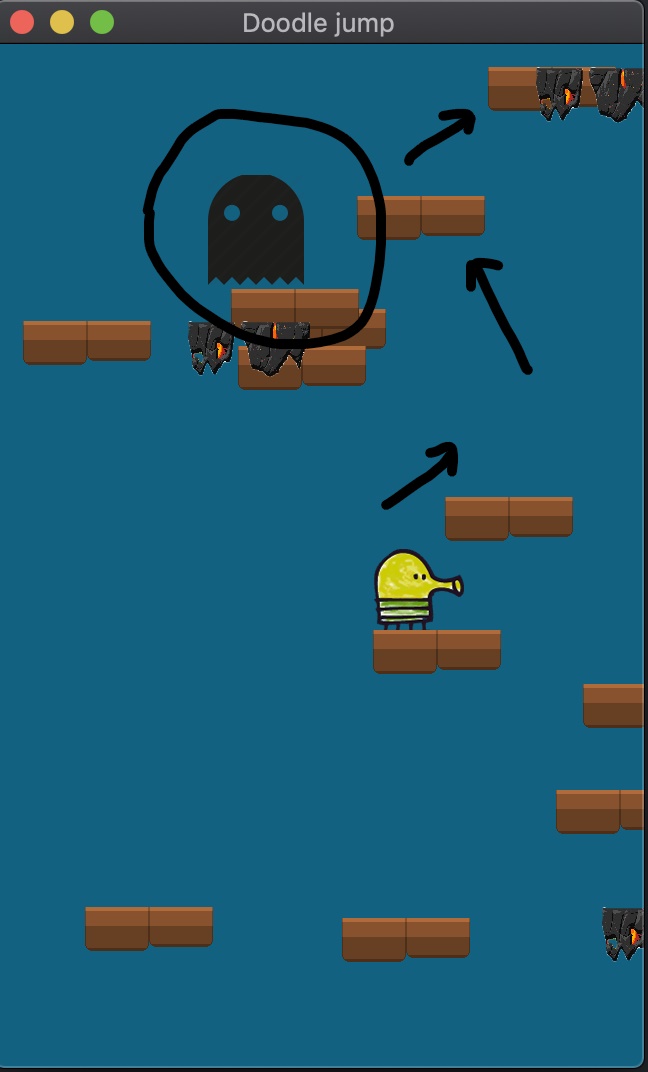


Рисунок 4.2 – Оптимальный путь для избегания контакта с монстром

На данном рисунке (4.3) мы видим реактивный ранец. Добраться к нему мы можем с помощью телепортации. Если мы коснемся ранца раньше монстра, то получим неуязвимость, и монстр нам будет не страшен.

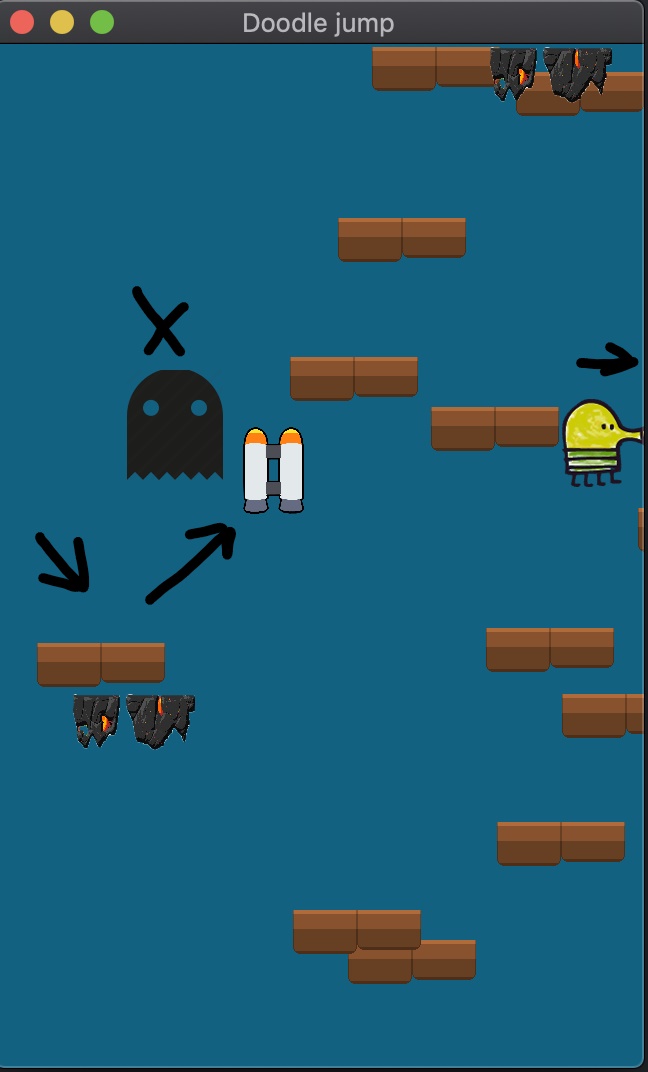


Рисунок 4.3 – Способ добраться до реактивного ранца

На этом рисунке(4.4) мы видим две опасные ситуации. Реактивный ранец сгенерировался прям рядом с монстром, брать его достаточно рискованно, однако нужно помнить про неуязвимость во время полета. Ложная панель находится прям над пропастью. Перед началом игры нужно знать какая платформа дает ускорение, а какая просто ломается. Прыгать на эту платформу нельзя, тк она сломается и мы полетим вниз.

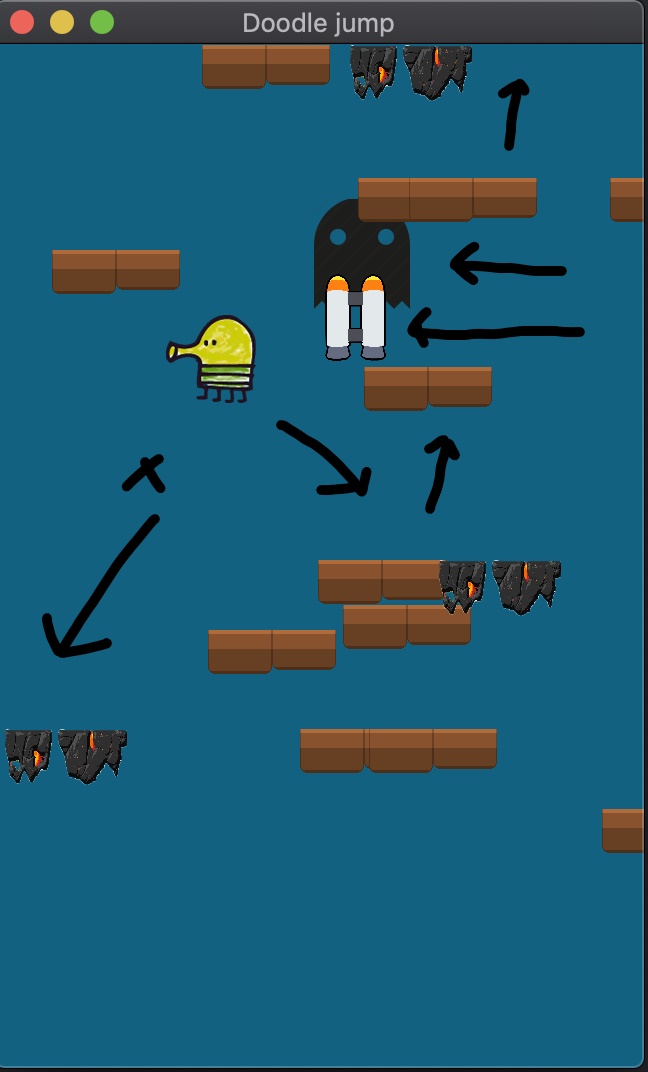


Рисунок 4.4 – Рискованная ситуация в игре

Подведем итоги. Каких правил стоит придерживаться для успешной игры :

1. Избегать контакта с монстром;
2. Брать реактивный ранец исключительно в случае когда нам не угрожает опасность на пути к нему;
3. Не становиться на ложные платформы;
4. Грамотно использовать телепортацию;
5. Во время полета стараться задеть новые реактивные ранцы для продления полета;
6. Наслаждаться игрой, ставить рекорды;
7. При приступе эпилепсии прекращать игру;
   1. Инструкция к сбросу рекордного счета

Серьезным недостатком игры является отсутствие возможности сброса рекордного счета без изменения файлов игры. Однако это может понадобиться, когда мы хотим дать поиграть кому-нибудь другому.

Инструкция:

1. Заходим в папку с игрой;
2. Заходим в папку textures;
3. Находим файл scores.txt;
4. Стираем значение;
5. Пишем новое(какое хотим) либо ничего не пишем;
6. Сохраняем файл

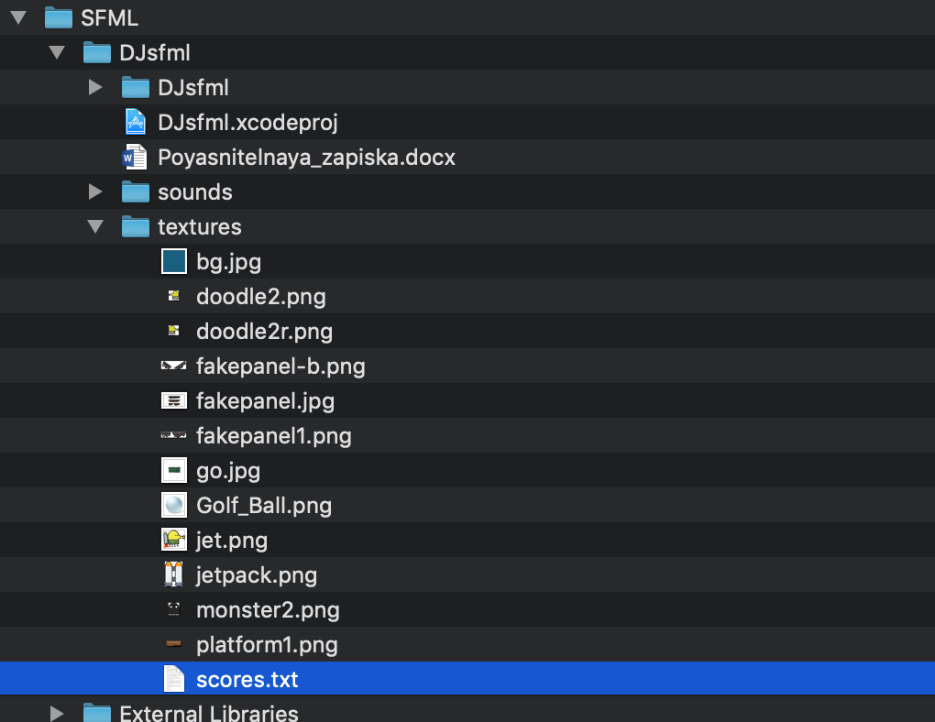


Рисунок 4.5 – Путь к файлу

ЗАКЛЮЧЕНИЕ

Развитие игровой индустрии набирает обороты. Однако такой жанр игр как «Времяубивалки» никогда не выйдет из трендов.

В рамках данного курсового проекта было разработано программное игровое средство «Doodle Jump», которое поможет вам отвлечься от важных дел и поностальгировать. Ведь сейчас в этой индустрии новые короли. Однако такую легендарную игру как Doodle Jump еще не скоро забудут. И более того, фанаты этой серии игр всегда будут отдавать предпочтение именно ей.

Для успешного выполнения всех поставленных целей потребовалось изучить библиотеку SFML, основы компьютерных систем и сетей, объектно-ориентированные возможности языка C++, изучить основные принципы данной парадигмы, а также освоить взаимодействия существующих компонентов.

Существует много возможностей для дальнейшего улучшения приложения. Одним из самых простых направлений является добавление мультиплеера. Интерфейс может быть дополнен новыми фонами, монстрами, ускорителями или новым внешним видом главного героя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке C++ / Гирберд Шилдт. Учебный курс: / Гирберд Шилдт. – СПб: изд. С. В. Малгачёва, 2001. – 231 с;
2. Documentation of the SFML 2.5.1;
3. Apple Developer Documentation;
4. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
5. 2009 год в компьютерных играх;
6. Культин, Н. А. Основы программирования в C++ 2-е издание – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 182с;
7. Культин, Н. А. Основы программирования в C++ – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 167с;

ПРИЛОЖЕНИЕ А

Исходный код программы

**Main.c**

|  |
| --- |
| // |
|  | // main.cpp |
|  | // DJsfml |
|  | // |
|  | // Created by Евгений on 10/28/19. |
|  | // Copyright © 2019 Евгений. All rights reserved. |
|  | // |
|  | #include <SFML/Graphics.hpp> |
|  | #include <SFML/Audio.hpp> |
|  | #include <time.h> |
|  | #include <iostream> |
|  | #include <fstream> |
|  | #include <sstream> |
|  |  |
|  |  |
|  | #define SCREEN\_WIDTH 650 |
|  | #define SCREEN\_HEIGHT 1024 |
|  | #define OUTGEN 1100 |
|  | #define GAMEOVER\_SCREEN 500 |
|  | #define DOODLE\_WIDTH 90 |
|  | #define DOODLE\_HEIGHT 88 |
|  | #define MONSTER\_WIDTH 96 |
|  | #define MONSTER\_HEIGHT 110 |
|  | #define PLATE\_WIDTH 128 |
|  | #define PLATE\_HEIGHT 46 |
|  | #define YRANGE\_GEN 900 |
|  | #define FAKE\_WIDTH 122 |
|  | #define FAKE\_HEIGHT 56 |
|  | #define JETPACK\_WIDTH 61 |
|  | #define JETPACK\_HEIGHT 87 |
|  | #define BORDER1 256 |
|  | #define BORDER2 512 |
|  | #define BORDER3 768 |
|  |  |
|  | using namespace sf; |
|  |  |
|  | class point |
|  | { |
|  | public: |
|  | int x,y; |
|  | }; |
|  |  |
|  | RenderWindow game(sf::VideoMode(SCREEN\_WIDTH,SCREEN\_HEIGHT),"Doodle jump"); |
|  |  |
|  |  |
|  | bool isMonsterDrawn = false; |
|  | bool isFlight = false; |
|  | int scoreplus; |
|  | int i; |
|  | Texture t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11; |
|  |  |
|  |  |
|  | Sprite sBackground, sPlat, sPers, sMonster,sGO, sBullet, sFakePanel,sBrokenPlat,sJetpack; |
|  |  |
|  | SoundBuffer deathSoundBuffer; |
|  | SoundBuffer jumpSoundBuffer; |
|  | SoundBuffer monsterSoundBuffer; |
|  | SoundBuffer jetSoundBuffer; |
|  |  |
|  |  |
|  | Sound death; |
|  | Sound jump; |
|  | Sound monsterhit; |
|  | Sound jet; |
|  |  |
|  | point plat[100]; |
|  | point monster[1]; |
|  | point FakePlat[12]; |
|  | point JetPack[1]; |
|  | point BrokenPlat[12]; |
|  | point bullet; |
|  |  |
|  |  |
|  | int x=400,y=1020,h=250; |
|  | float dx=0,dy=0; |
|  | bool isShoot = false; |
|  | int counter=0; |
|  |  |
|  | void setSeed() { |
|  | srand(time(0)); |
|  | } |
|  |  |
|  | void imports() |
|  | { |
|  | t1.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/platform1.png"); |
|  | t2.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/bg.jpg"); |
|  | t3.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/doodle2.png"); |
|  | t4.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/monster2.png"); |
|  | t5.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/go.jpg"); |
|  | t6.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/doodle2r.png"); |
|  | t7.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/Golf\_Ball.png"); |
|  | t8.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/fakepanel1.png"); |
|  | t9.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/fakepanel-b.png"); |
|  | t10.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/jetpack.png"); |
|  | t11.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/jet.png"); |
|  |  |
|  | deathSoundBuffer.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/sounds/fall.ogg"); |
|  | jumpSoundBuffer.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/sounds/jump.ogg"); |
|  | monsterSoundBuffer.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/sounds/monster.ogg"); |
|  | jetSoundBuffer.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/sounds/jet.wav"); |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | void clearScreen() |
|  | { |
|  | printf("\e[1;1H\e[2J"); |
|  | //flush(std::cout); |
|  | } |
|  |  |
|  | void setUpApp() |
|  | { |
|  | game.setFramerateLimit(60); |
|  |  |
|  | imports(); |
|  |  |
|  | sPers = Sprite(t3); |
|  | sPlat = Sprite(t1); |
|  | sBackground = Sprite(t2); |
|  | sMonster = Sprite(t4); |
|  | sBullet = Sprite(t7); |
|  | sFakePanel = Sprite(t8); |
|  | sBrokenPlat = Sprite(t9); |
|  | sJetpack = Sprite(t10); |
|  |  |
|  |  |
|  | sFakePanel.scale(0.25f, 0.5f); |
|  | sBrokenPlat.scale(0.25f,0.5f); |
|  | sJetpack.scale(0.17f,0.17f); |
|  |  |
|  | death.setBuffer(deathSoundBuffer); |
|  | jump.setBuffer(jumpSoundBuffer); |
|  | monsterhit.setBuffer(monsterSoundBuffer); |
|  | jet.setBuffer(jetSoundBuffer); |
|  |  |
|  |  |
|  |  |
|  | for (int i=0;i<=3;i++) |
|  | { |
|  | plat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | plat[i].y=BORDER3 +(rand()%BORDER1); |
|  | if (i==3) |
|  | { |
|  | FakePlat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | FakePlat[i].y=BORDER3 +(rand()%BORDER1); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | for (int i=4;i<=6;i++) |
|  | { |
|  | plat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | plat[i].y=BORDER2+(rand()%BORDER1); |
|  | if (i==6) |
|  | { |
|  | FakePlat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | FakePlat[i].y=BORDER3 +(rand()%BORDER1); |
|  | } |
|  | } |
|  |  |
|  | for (int i=7;i<=9;i++) |
|  | { |
|  | plat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | plat[i].y=BORDER1+(rand()%BORDER1); |
|  | if (i==9) |
|  | { |
|  | FakePlat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | FakePlat[i].y=BORDER3 +(rand()%BORDER1); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | for (int i=10;i<=12;i++) |
|  | { |
|  | plat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | plat[i].y=rand()%BORDER1; |
|  | if (i==12) |
|  | { |
|  | FakePlat[i].x=rand()%(SCREEN\_WIDTH-PLATE\_WIDTH); |
|  | FakePlat[i].y=BORDER3 +(rand()%BORDER1); |
|  | } |
|  | } |
|  |  |
|  | JetPack[1].x=0; |
|  | JetPack[1].y=0; |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | int checkX2(int x) |
|  | { |
|  | switch (x) { |
|  | case 640: |
|  | x=0; |
|  | return x; |
|  | break; |
|  | case 0: |
|  | x=640; |
|  | return x; |
|  | break; |
|  |  |
|  | default: |
|  | break; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | return x; |
|  | } |
|  |  |
|  | std::string convertToString(char\* a,int size) |
|  | { |
|  | int i; |
|  | std::string s = ""; |
|  | for(i=0; i<size;i++) |
|  | { |
|  |  |
|  | s=s+a[i]; |
|  | } |
|  | return s; |
|  |  |
|  | } |
|  |  |
|  | void gameover(int counter){ |
|  |  |
|  | RenderWindow result(sf::VideoMode(GAMEOVER\_SCREEN,GAMEOVER\_SCREEN),"RESULT"); |
|  | result.setFramerateLimit(60); |
|  | Texture r1; |
|  | r1.loadFromFile("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/go.jpg"); |
|  | Sprite sRes(r1); |
|  | sf::Text score; |
|  | sf::Font Font; |
|  | char str[5]; |
|  | std::ifstream fout("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/scores.txt"); |
|  | fout.getline(str, 50); |
|  | fout.close(); |
|  | int b=0; |
|  | std::string best = convertToString(str, 5); |
|  | std::stringstream bestf(best); |
|  | bestf >> b; |
|  |  |
|  | bool isNewScore=false; |
|  | if (counter>b) |
|  | { |
|  | std::ofstream fout("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/scores.txt",std::ios::trunc); |
|  | fout.close(); |
|  | std::ofstream fscore("/Users/zhenya\_rs6/Desktop/SFML/DJsfml/textures/scores.txt",std::ios::out); |
|  | std::string s = std::to\_string(counter); |
|  | fscore << s; |
|  | fout.close(); |
|  | isNewScore = true; |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | // sf::SoundBuffer buffer; |
|  | // buffer.loadFromFile();// тут загружаем в буфер что то |
|  | // sf::Sound sound; |
|  | // sound.setBuffer(buffer); |
|  | // sound.play(); |
|  | if (!Font.loadFromFile("/Users/zhenya\_rs6/Downloads/Arial/Arial.ttf")) |
|  | { |
|  | //handle error |
|  | } |
|  |  |
|  | score.setCharacterSize(50); |
|  | std::string s = std::to\_string(counter); |
|  |  |
|  | if (isNewScore==true) |
|  | { |
|  | score.setString(" NEW BEST:"+s); |
|  | } |
|  | else |
|  | { |
|  | score.setString(" GAME OVER\n SCORE: "+s); |
|  | } |
|  | score.setFont(Font); |
|  | score.setFillColor(sf::Color::Cyan); |
|  | while (result.isOpen()) { |
|  | result.clear(); |
|  |  |
|  | Event e; |
|  | while (result.pollEvent(e)) |
|  | { |
|  | if (e.type == Event::Closed) |
|  | result.close(); |
|  | } |
|  |  |
|  |  |
|  | result.draw(score); |
|  | result.display(); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | void drawing() |
|  | { |
|  | sPers.setPosition(x, y); |
|  |  |
|  | game.draw(sBackground); |
|  | game.draw(sPers); |
|  | i=1; |
|  | if (counter>100) |
|  | { |
|  | sMonster.setPosition(monster[i].x, monster[i].y); |
|  | game.draw(sMonster); |
|  | isMonsterDrawn = true; |
|  | } |
|  |  |
|  | for (int i=0;i<12;i++) |
|  | { |
|  | sPlat.setPosition(plat[i].x, plat[i].y); |
|  | game.draw(sPlat); |
|  |  |
|  | } |
|  | for (int i=0;i<=12;i++) |
|  | { |
|  | if (i==3 || i==6 || i==9 ) |
|  | { |
|  | if (FakePlat[i].x!=BrokenPlat[3].x && FakePlat[i].x!=BrokenPlat[6].x && FakePlat[i].x!=BrokenPlat[9].x && FakePlat[i].x!=BrokenPlat[12].x) |
|  | { |
|  | sFakePanel.setPosition(FakePlat[i].x, FakePlat[i].y); |
|  | game.draw(sFakePanel); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | for (int i=0;i<=12;i++) |
|  | { |
|  | if (i==3 || i==6 || i==9 ) |
|  | { |
|  | if (BrokenPlat[i].x!=0 && BrokenPlat[i].y!=0) |
|  | { |
|  | sBrokenPlat.setPosition(BrokenPlat[i].x, BrokenPlat[i].y); |
|  | game.draw(sBrokenPlat); |
|  | } |
|  |  |
|  |  |
|  | } |
|  | } |
|  |  |
|  | if (JetPack[1].x!=0 && JetPack[1].y!=0 && counter>=120) |
|  | { |
|  | sJetpack.setPosition(JetPack[1].x,JetPack[1].y); |
|  | game.draw(sJetpack); |
|  |  |
|  | } |
|  | } |
|  |  |
|  |  |
|  | void reRender() |
|  | { |
|  | counter+=5; |
|  | for (int i=0;i<=12;i++) |
|  | { |
|  | y=h; |
|  | plat[i].y=plat[i].y-dy; |
|  | FakePlat[i].y=FakePlat[i].y-dy; |
|  | BrokenPlat[i].y=BrokenPlat[i].y-dy+50; |
|  |  |
|  |  |
|  | if (plat[i].y>YRANGE\_GEN) |
|  | {plat[i].y=0; plat[i].x= rand()%SCREEN\_WIDTH; |
|  | if (FakePlat[i].y>YRANGE\_GEN) |
|  | { |
|  | FakePlat[i].y=0; FakePlat[i].x= rand()%SCREEN\_WIDTH; |
|  | } |
|  | } |
|  |  |
|  | } |
|  | monster[1].y-=dy; |
|  | int i=1; |
|  |  |
|  | if (monster[1].y>OUTGEN) |
|  | { |
|  | int isMonsterSpawn = rand()%15; |
|  | if ((isMonsterSpawn==1 || isMonsterSpawn==2 || isMonsterSpawn==3) && counter>100){ |
|  |  |
|  | monster[i].x=40+rand()%400; |
|  | monster[i].y=0; |
|  | isMonsterDrawn=true; |
|  | } |
|  | else |
|  | { |
|  | isMonsterDrawn=false; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | JetPack[1].y-=dy; |
|  | if (JetPack[1].y>OUTGEN) |
|  | { |
|  | i=rand()%15; |
|  | if (i==1 || i==2 || i==3 || i==4 || i == 5 || i==6) { |
|  |  |
|  | JetPack[1].y=0; |
|  | JetPack[1].x=40+rand()%400; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | JetPack[1].x=0; |
|  | JetPack[1].y=0; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void checktouch() |
|  | { |
|  |  |
|  | for (int i=0;i<12;i++) |
|  | { |
|  | if ((x+60>plat[i].x) && (x+30<plat[i].x+PLATE\_WIDTH) && (y+78>plat[i].y) && (y+78<plat[i].y+PLATE\_HEIGHT) && (dy>0)) |
|  | { |
|  | jump.play(); |
|  | dy=-10; |
|  |  |
|  |  |
|  | } |
|  | } |
|  |  |
|  | if ((x>JetPack[1].x-DOODLE\_WIDTH) && (x<JetPack[1].x+JETPACK\_WIDTH) && (y>JetPack[1].y-DOODLE\_HEIGHT) && (y<JetPack[1].y+JETPACK\_HEIGHT) && JetPack[1].x!=0 && JetPack[1].y!=0) |
|  | { |
|  | isFlight=true; |
|  | jet.play(); |
|  | sPers.setTexture(t11); |
|  | dy =-30; |
|  | scoreplus = counter; |
|  | } |
|  |  |
|  | if ((counter-scoreplus)>650 && isFlight==true) |
|  | { |
|  | isFlight=false; |
|  | sPers.setTexture(t3); |
|  |  |
|  | } |
|  | for (int i=0;i<12;i++) |
|  | { |
|  | if ((x+60>FakePlat[i].x) && (x+30<FakePlat[i].x+FAKE\_WIDTH) && (y+78>FakePlat[i].y) && (y+78<FakePlat[i].y+FAKE\_HEIGHT) && (dy>0)) |
|  | { |
|  |  |
|  | BrokenPlat[i].x=FakePlat[i].x; |
|  | BrokenPlat[i].y=FakePlat[i].y; |
|  | } |
|  | } |
|  | // doodle 90x88 plate 128x46 monster |
|  | int i=1; |
|  |  |
|  | if ((x>monster[i].x-DOODLE\_WIDTH) && (x<monster[i].x+MONSTER\_WIDTH) && (y>monster[i].y-DOODLE\_HEIGHT) && (y<monster[i].y+MONSTER\_HEIGHT) && isMonsterDrawn == true && isFlight==false) |
|  | { |
|  | monsterhit.play(); |
|  | game.close(); |
|  |  |
|  | gameover(counter); |
|  |  |
|  | } |
|  | } |
|  |  |
|  | void run() |
|  | { |
|  | x=checkX2(x); |
|  |  |
|  | Event e; |
|  | while (game.pollEvent(e)) |
|  | { |
|  | if (e.type == Event::Closed) |
|  | game.close(); |
|  | } |
|  | if (Keyboard::isKeyPressed(Keyboard::Right)) |
|  | { |
|  | x+=4; |
|  | if (isFlight==false) |
|  | { |
|  | sPers.setTexture(t6); |
|  | } |
|  |  |
|  | } |
|  | if (Keyboard::isKeyPressed(Keyboard::Space)) |
|  | { |
|  | isShoot=true; |
|  |  |
|  | } |
|  |  |
|  | if (Keyboard::isKeyPressed(Keyboard::Left)) |
|  | { |
|  | x-=4; |
|  | if (isFlight==false) |
|  | { |
|  | sPers.setTexture(t3); |
|  | } |
|  | } |
|  |  |
|  | dy+=0.2; |
|  |  |
|  | y+=dy; |
|  |  |
|  |  |
|  | if (y>YRANGE\_GEN) dy=-10; |
|  |  |
|  | if (y<h) |
|  | { |
|  | reRender(); |
|  | } |
|  |  |
|  | checktouch(); |
|  |  |
|  | if (y>YRANGE\_GEN && counter>45) |
|  | { |
|  | death.play(); |
|  | game.close(); |
|  | gameover(counter); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | drawing(); |
|  |  |
|  | game.display(); |
|  | } |
|  |  |
|  | void init() |
|  | { |
|  | setSeed(); |
|  |  |
|  | setUpApp(); |
|  | } |
|  |  |
|  |  |
|  | int main( ) |
|  | { |
|  |  |
|  |  |
|  | init(); |
|  |  |
|  |  |
|  | while (game.isOpen()) |
|  | { |
|  | run(); |
|  | } |
|  | return 0; |
|  |  |
|  |  |
|  | } |