Министерство образования и науки РФ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования «НИТУ «МИСиС»
Институт ИТКН
Кафедра ИК

Отчёт по курсовой работе
по дисциплине
«Технологии программирования»

Игра «Звездная пустошь»

Выполнил:
Студент группы БПМ-19-2
Ивершин В.С.
Проверил:
Полевой Д.В.

Москва 2020

ОГЛАВЛЕНИЕ

ЗАДАЧА

Написать пользовательское приложение-игру.

Игрок управляет космическим кораблём в космосе. Его врагами являются астероиды, появляющиеся из случайных точек вне карты и летящие строго на игрока.

Игра не имеет конца, главная цель игрока – получить как можно больше очков за уничтожение астероидов с помощью бластера. Чтобы выжить, игрок может разрушать астероиды двумя способами: с помощью бластера или прячась за планетами, появляющимися на карте с определенными очками прочности (Рисунок 1). Когда игрок прячется за планетой, она становится полупрозрачная, чтобы можно было видеть свой корабль.

Наличие меню игры.

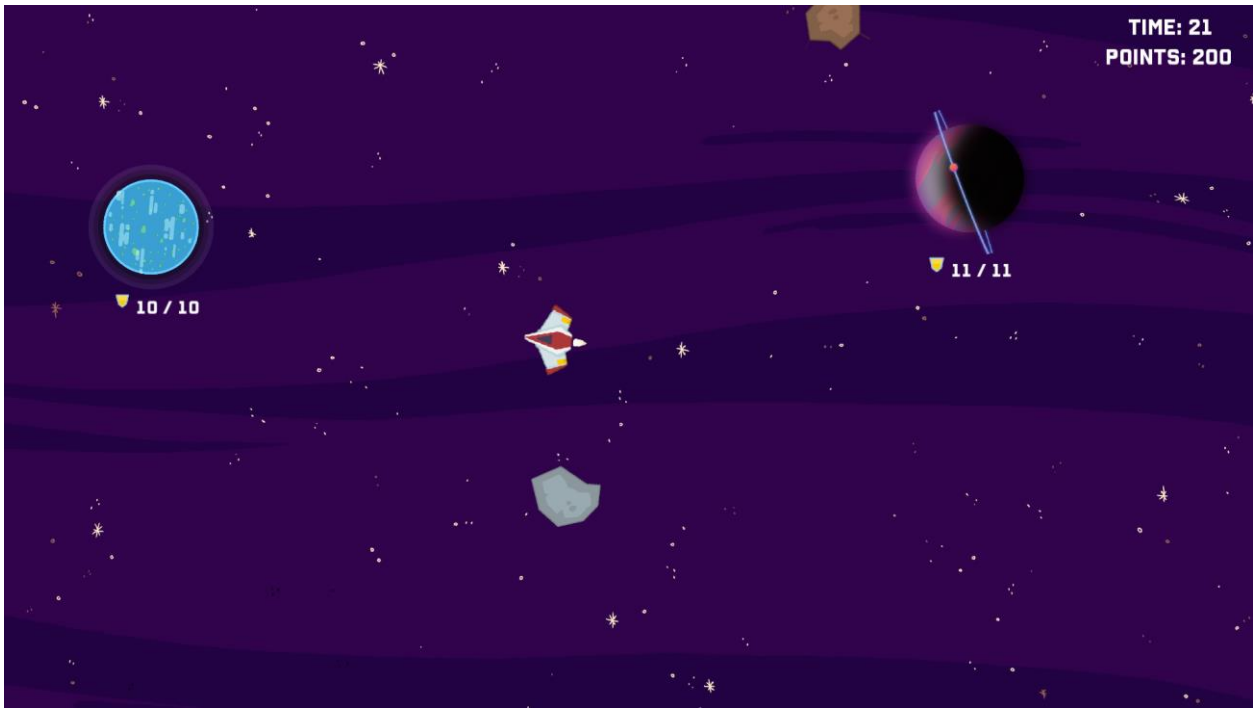

Рисунок 1 – Общий вид игры

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Реализовать графическое приложение на языке C++17 версии с использованием библиотеки SFML 4.2. Игра находится в полноэкранном режиме. Сначала игрок попадает в главное меню игры, в котором есть кнопки «PLAY», «RULES», «RECORDS», «SETTINGS», «EXIT».

При взаимодействии с кнопками происходят визуальные и звуковые эффекты. Во время игрового процесса идет подсчет прошедшего времени и количества набранных очков.

Со случайными параметрами на карте появляются планеты, которые имею очки прочности. Когда игрок подлетает к планете она становится полупрозрачной. При появлении планеты есть анимация ее «рождения» со звуком. Из случайных точек вне карты на игрока летят астероиды со случайными параметрами. При столкновении объектов (астероид – планета / астероид - игрок) воспроизводятся визуальные и звуковые эффекты. Игрок может выпускать лазер, который может разрушить 1 астероид. Перезарядка бластера зависит от его скорости. При выстреле проигрывается звук выстрела.

Каждые 30 секунд игра усложняется (увеличивается скорость астероидов и время появления планеты после ее смерти). Если снаряд попадает в игрока, то наступает конец игры и открывается меню со статистикой по текущей игре и тремя кнопками: «EXIT», «MENU», «RESTART».

Настройки и рекорды сохраняются при их изменении.

ПОЛЬЗОВАТЕЛЬСКАЯ ДОКУМЕНТАЦИЯ

После запуска приложения игрок попадает в главное меню (Рисунок 2), в котором у вас есть на выбор 5 кнопок с различными функциями. «RULES» показывает вам правила игры (Рисунок 3). Для настройки музыки и звуков, а также установки игрового имени, следует нажать на кнопку «SETTINGS» (Рисунок 4). Следует помнить, что длина игрового имени не должна быть

меньше 2 и больше 15 символов. Для сохранения настроек нужно нажать на кнопку «SAVE». Если желаете посмотреть таблицу рекордов – нажмите «RECORDS» (Рисунок 5).

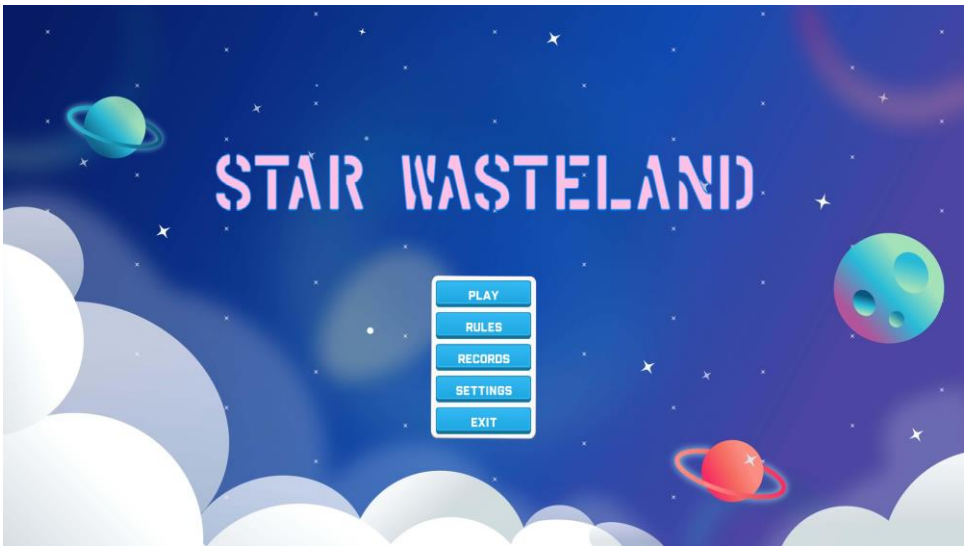

Рисунок 2 - Главное меню

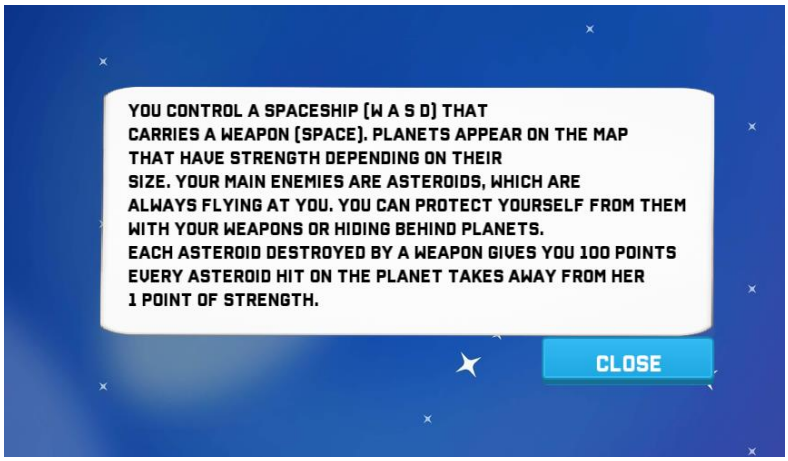

Рисунок 3 - Правила

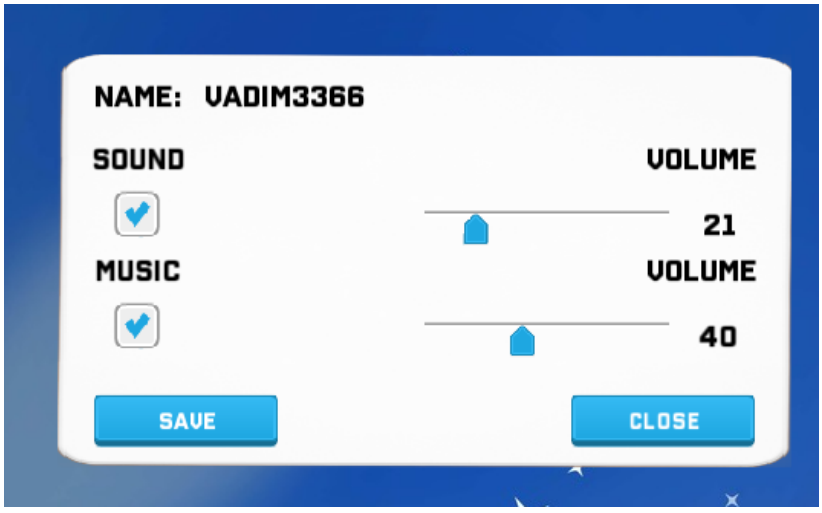

Рисунок 4 - Настройки

Рисунок 5 - Рекорды

Для начала игры нажмите кнопку «PLAY». После этого вы окажетесь в игровой зоне, управляя космическим кораблем (Рисунок 6) при помощи клавиш WASD. В верхнем правом углу экрана ведется статистика вашей игры (Рисунок 7).

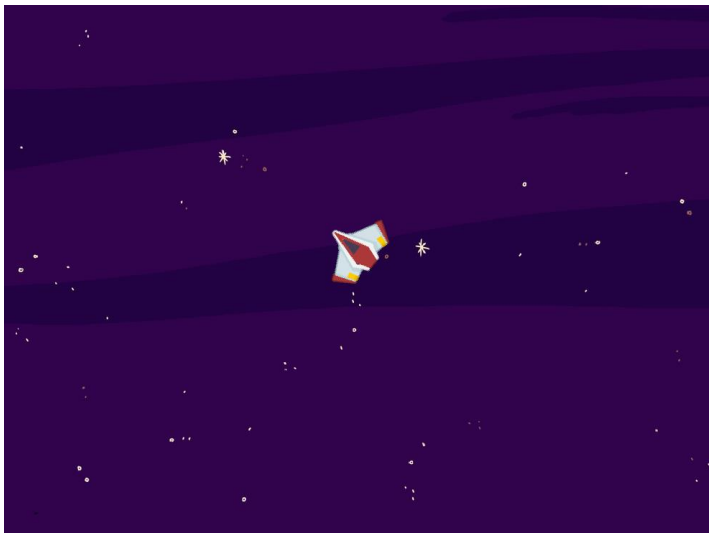

Рисунок 6 - Корабль игрока

Рисунок 7 - Статистика игры

Главными врагами игрока являются астероиды (Рисунок 8), летящие из случайной точки вне игровой области на игрока, которые при столкновении с игроком приводят к концу игры.

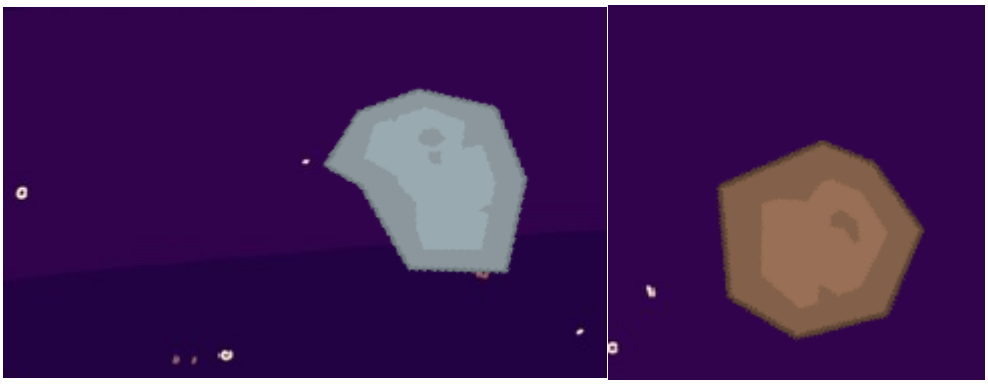

Рисунок 8 - Астероиды

В игровой зоне появляются планеты (Рисунок 9) с разными случайными параметрами, основным из которых для игрока является количество очков прочности. Планеты защищают игрока от астероидов, если они сталкиваются. Очки прочности – количество астероидов, столкновение с которыми планета может выдержать. Когда игрок находится за планетой, она становится полупрозрачной, чтобы он мог контролировать ситуацию.

Рисунок 9 - Планеты

Когда астероид сталкивается с игроком, наступает конец игры. В этом случае перед игроком появляется меню (Рисунок 10), в котором он может посмотреть результаты своей игры, выйти в главное меню, выйти из игры или начать заново.

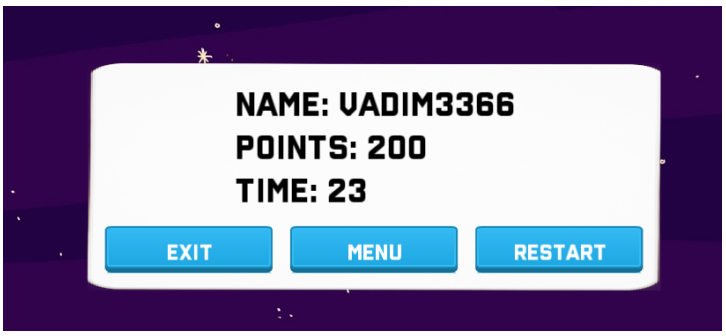

Рисунок 10 - Конец игры

ОСНОВНЫЕ ФУНКЦИИ

Для создания графического интерфейса используется каркас SFML 2.4.2

Классы

Классы с их кратким описанием.

Asteroid (класс астероидов)
CheckBox (класс кнопки с состояниями true / false)
Menu (класс вызова всех меню игры)
MenuButton (классичесая кнопка)
Minimizer (класс для сворачивания окна)
Planet (класс планеты)
Player (класс игрока)
Result (рекорд)
Rollover (регулятор уровня громкости)
settings (настройки игры)
Spawner (генератор параметров для объектов)
textureLoader (хранилище текстур)
TopResults (топ рекордов)
Weapon (оружие игрока)

Все текстуры игры хранятся в классе textureLoader в виде vector или map, хранящих указатели на текстуры:

- std::vector< Texture * > **players**
- std::vector< Texture * > **planets**
- std::vector< Texture * > **asteroids**
- std::vector< Texture * > **weapons**
- std::vector< Texture * > **dest_effect**
- std::vector< Texture * > **ui**
- std::vector< Texture * > **backgrounds**
- std::map< std::string, sf::Texture * > **menu**

Для их использования в main функции создается экземпляр этого класса и передаются текстуры из нужной категории.

Настройки игры хранятся в файле settings.ini. Для их сохранения, записи и чтения в программе используется класс settings, экземпляр которого хранит значения настроек:

- bool **soundIsOn**
- bool **musicIsOn**
- int **soundVolume**
- int **musicVolume**
- std::string **nickname**
- int **maxLengthNick**

Для записи, чтения и сохранения используются следующие методы:

bool settings::read () - чтение из файла в экземпляр класса

Возвращает

true при успешном открытии файла

bool settings::write () - запись в файл настроек из экземпляра класса

Возвращает

true при успешном открытии файла

Таблица рекордов хранится в файле records.ini и для работы с ними используется класс TopResults, в одном заголовочном файле с которым есть структура Result, хранящая значения для записи одного рекорда:

- std::string **name**
- int **points** = 0
- int **time** = 0

Класс TopResult содержит методы по записи, чтению, сохранению рекордов, проверки на рекорд и создания таблицы:

- bool **read** () - *чтение из файла с рекордами и запись в экземпляр класса*
- bool **write** ()
- bool **write** (std::string name, int point, int time) - *запись нового рекорда в экземпляр класса с сортировкой и перезаписью в файл*
- bool **isRecord** (int point) - *проверка входит ли введенное количество очков в таблицу рекордов*
- void **makeView** (int charSize, sf::Font &font, sf::Color color) - *создание макета таблицы рекордов*

Главное меню игры, его подменю, а также меню конца игры обрабатываются в классе Menu. Для работы с ними создается экземпляр этого класса при помощи конструктора:

Menu::Menu (settings & ***setting*****, TopResults &** ***records*****, textureLoader &** ***textures*****, Vector2u &** ***ScreenSize*****, Font &** ***font*****, Font &** ***recordFont*****, std::string** ***Prefix*****) -** Конструктор

Аргументы

setting	настройки игры
records	рекорды игры
textures	все загруженные текстуры
ScreenSize	размер окна
font	шрифт для меню
recordFont	шрифт для таблицы рекордов

Prefix	префикс пути к файлам

После создания экземпляра для вывода на экран нужного меню вызывается метод menu, который в свою очередь вызывает свои подменю вызовом приватных методов.

- bool **menu** (RenderWindow &window) - *вызов главного меню*
- int **endGameMenu** (RenderWindow &window, int pointsCount, int gameTime, RectangleShape &Background) - *вызов меню меню после конца игры*

Кнопки, а также остальные манипуляторы с меню являются экземплярами одного из классов: MenuButton (классическая кнопка), CheckBox (кнопка с состоянием true/false) и Rollover (регулятор уровня громкости). Каждая из кнопок для обработки нажатия на нее использует метод listen.

Космический корабль, которым управляет игрок, является экземпляром класса Player. Для задания ему начальных параметров существует конструктор:

Player::Player (Texture * *texture*, const double & *speed*, const double & *angularSpeed*, Vector2u *screen*) - Конструктор

Аргументы

texture	текстура игрока, содержащая все его состояния
speed	скорость движения
angularSpeed	скорость поворота
screen	размер окна

Для управления им с помощью клавиатуры в обработчике нажатия вызываются методы:

- void **flyForward** (float time, float &currentFrame) - *смена анимации при движении вперед*
- void **flyBack** (const float &time) - *остановка при движении назад*
- void **flyLeft** (const float &time) - *поворот влево на месте*
- void **flyRight** (const float &time) - *поворот вправо на месте*
- void **flyForwardAndBack** () - *одновременное нажатие вперед и назад*

Траектория движения считается в методе void **Move** (bool forward)

Оружие, которым управляет игрок, является экземпляром класса Weapon, создаваемым конструктором:

Weapon::Weapon (Texture * *texture*, double *Speed*, float *kd*, float *screenR*) - Конструктор

Аргументы

texture	текстура лазера
Speed	скорость полета

kd	время перезарядки
screenR	соотношение ширины и высоты экрана

Оно имеет аналогичный классу Player метод Move для движения в сторону, в которую был повернут корабль игрока.

Астероиды, летящие на игрока, являются объектами класса Asteroids. В конструкторе астероиду присваиваются начальные параметры:

Asteroid::Asteroid (float *diametre*, float *Speed*, Vector2u *screen*, Vector2f *SpawnPosition*, int *SpawnTime*, Texture * *texture*) - Конструктор

Аргументы

diametre	диаметр астероида
Speed	скорость полета
screen	размер окна
SpawnPosition	позиция спавна
SpawnTime	время, через которое астероид начнет полет
texture	текстура астероида

Если астероид уничтожен или его время перезарядки прошло, то вызывается метод void **update** (Vector2f spawn, Texture *texture) - *обновление позиции астероида*.

Перемещение астероида на игрока производится в методе Move:

void Asteroid::Move (Player & *player* - перемещение астероида на игрока

При уничтожении астероида вызывается метод Destroy:

void **Destroy** (std::vector< Texture * > &textures) - *анимация уничтожения астероида*

Класс Planet хранит логику планет. Они создаются конструктором:

Planet::Planet (float *Radius*, float *AngularSpeed*, Texture * *texture*, Vector2f *Position*, int *hp_*, float *spawnKD_*, int *Area*, Font & *font*, Texture * *icon_*, Texture * *dest_effect*) - Конструктор

Аргументы

Radius	радиус планеты
AngularSpeed	скорость вращения
texture	текстура планеты
Position	позиция спавна
*hp_*	количество очков прочности
*spawnKD_*	время следующего появления
Area	номер занимаемой области на экране
font	шрифт текста количества очков прочности

*icon_*	иконка щита слева от очков прочности
dest_effect	текстура, содержащая состояния взрыва планеты

Важнейшие функции планет:

- bool **Intersects** (RectangleShape shape) - *столкновение объекта с планетой*
- void **Rotation** () - *поворот планеты вокруг своей оси*
- void **Spawn** () - *анимация рождения планеты*
- void **Die** (float Radius, float AngularSpeed, Texture *texture, Vector2f Position, int hp_, float spawnKD_, int area) - *обновление параметров планеты после ее смерти*
- void **Destroy** (int width, int height, Vector2f beginPoint, int sizeRect) - *анимация уничтожения планеты на одной текстуре*
- void **Transparency** (RectangleShape &object) - *установление полупрозрачности при столкновении с объектом*

Для задания планетам и астероидам случайных параметров используются методы класса Spawner. Простейшими его методами являются:

- int **generator** (int min, int max) - *генератор случайного int числа*
- float **generator** (float min, float max) - *генератор случайного float числа*

Для выбора случайной позиции появления астероида с предотвращением его появления в недопустимых зонах используется метод:

Vector2f **generatorAsteroids** (Vector2u screen) - *создает точку спавна астероида в допустимых зонах*

Для корректного появления нескольких планет (без наложения одной на другую, выхода ее за пределы экрана) в main функции создается vector, в котором хранятся позиции левых верхних углов областей спавна. Эти области представляют собой 6 прямоугольников, которые делят экран в отношении 3:2. Для хранения уже используемых областей используется map.

При создании планеты ей выбирается допустимая область появления при помощи метода:

int **chooseArea** (std::vector< Vector2f > &spawnPoints, std::map< int, Vector2f > &useSpawnPoints) - *выбор области спавна*

После выбора области происходит выбор точной позиции появления планеты методом:

Vector2f **generatorPlanets** (Vector2u screen, std::vector< Vector2f > &spawnPoints, int radius, int &area) - *создает точку спавна планеты в заданной области*

После этих манипуляций в map, содержащую занятые области добавится занятая новой планетой область. При смерти планеты эта область освобождается.

РУКОВОДСТВО ПО СБОРКЕ

1) Скачать исходные файлы по адресу:
https://github.com/Dartanum/Course_Task
2) Скачать SFML 2.4.2 Visual C++14 (2015) 64-bit по ссылке
https://www.sfml-dev.org/download/sfml/2.4.2/
3) В CMake создать переменную (Add Entry) с названием SFML_DIR, типом PATH и со значением абсолютного пути к библиотеке SFML.
4) Собрать проект под Visual Studio версии 14 или выше, 64-bit.
5) В структуре решения собрать подпроект “INSTALL”
6) Файл с расширением .exe находится в пути, указанном в CMAKE_INSTALL_PREFIX:

и документация по адресу:
<адрес директории сборки>/doc/rtf/refman.rtf