МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибернетики

Кафедра высшей математики

Утверждаю
Заведующий кафедрой _____ Ю.И.Худак
«10» 10 2019г.

ЗАДАНИЕ
на выполнение курсовой работы
по дисциплине «Языки и методы программирования»

Студент Валяев *Н.А.* Группа *КМБО-04-18*

1. **Тема: «Создание Баллистической игры на языке С++»**

2. **Исходные данные:**
В разных концах экрана расположены две пушки, принадлежащие двум игрокам. Игроки ходят поочерёдно. Ход заключается в выборе массы камня для пушки, энергии (в процентах) приложенной к камню и угла между вектором скорости и горизонтом. По этим данным программа рассчитывает траекторию полета и моделирует полет на экране. Игроки ходят по очереди, пока один из них не разрушит пушку другого.

3. **Перечень вопросов, подлежащих разработке, и обязательного графического материала:**
   1) Реализовать игровое меню.
   2) Реализовать визуализацию полета

4. **Срок представления к защите курсовой работы: до «30» 12 2019 г.**

Задание на курсовую работу выдал «10» 10 2019 г. _____ (Шрамков В.К.)
Задание на курсовую работу получил «10» 10 2019 г. _____ (_____)

Содержание

1.Введение……2
2.Идея разработки……..3
3.Основные идеи……4
4.Классы и методы SFML……6
5.Мои Классы и их методы и данные……7
6.Особенности……..……..11
7.Защита от дурака……..15
8.Тесты……..……..16
9.Заключение…….21
10.Список Литературы……..……...21

Введение

Целью данной курсовой является реализация “Баллистической игры”
Два игрока стреляют из пушек расположенных в разных концах экрана
Возможна как и одновременная так и поочередная стрельба, последнее реализуется по средствам устного договора.
Игроку известны масса ядра, угол базовой скорости и процент от максимальной энергии. Также есть возможность теоретического расчета части траектории до выстрела. и реализована анимация для изображения взрывов. Есть требуемое условием меню и возможность менять размер игрового поля.
Для реализации выбран язык C++

Система автоматизации сборки программного обеспечения: CMake
Среда разработки: CLion
Целевая ОС: Linux/Windows
ОС разработки: Linux(Ubuntu)

Идея разработки

Так как это игра, первое что нам нужно - класс игрока; как показывает мой малый опыт разработки за игрока может сойти набор из нескольких функций, но поскольку данный проект подразумевает двух игроков, разумнее сделать их объектами класса.

Игроки стреляют друг в друга, значит нам нужен класс пули/ядра.
Здесь обязательность класса возрастает, ведь ядер может быть сильно больше, чем два игрока.

Ядра обычно взрываются, суть реализации анимации описана ниже. Пока очевидно лишь то, что это еще один класс. Так как взрывов, как и ядер много, и все они схожи, применим ООП.

Также реализован класс игры с целью инкапсуляции методов отрисовки и контейнеров с ядрами и взрывами, а также их сокрытия, что менее важно, но косвенно следует из желаемой инкапсуляции.

Для отрисовки графики использована сторонняя библиотека SFML(Simple and Fast Multimedia Library).
Реализована данная библиотека в виде набора классов, использование которых описано ниже.
Все подробности ее “подкапотной” работы для меня тайна за семью печатями, все что мне известно, это то, что она работает с OpenGL.

Основные идеи

1) Тут нет атмосферы

Поскольку виртуальное ядро вещь тяжело осязаемая, а виртуальная атмосфера еще проблемнее, в этом аспекте мною было принято решение работать исходя из установки, что атмосферы, а как следствие и трения об нее ядра, не существует.
Но гравитация все еще подкладывает нам свинью своим существованием, да здравствует физика 7-го класса и ее задачки на равноускоренное движение.
"Равноускоренное движение - это движение, при котором вектор ускорения не меняется по модулю и направлению."
Формулы для решения самых примитивных из этих задач выглядят как:

$$\boldsymbol{x = x_0 + v_x * t} \qquad \text{и} \qquad \boldsymbol{y = y_0 + v_y * t + \frac{-gt^2}{2}}$$

В данном случае ускорение свободного падения(g) взято со знаком минус так как предполагается что положительным по оси Y будет выстрел вверх

2)"Живая сила" Лейбница

Ядро умеет летать, но как заставить его отправится в правильный полет? У нас есть максимальные энергия и масса и их коэффициенты, также есть угол поворота пушки. Нам же нужны две составляющие начальной скорости ядра по осям X и Y, как их найти? Что ж "Живая сила" Лейбница говорит нам что, кинетическая энергия это именно то что получит ядро при выстреле. Значит искать будем ее.

$E = \frac{m*v^2}{2}$ скорость ядра будем считать как $V = \sqrt{2E/m}$

А скорости по X и по Y как: $v_x = V * cos(\alpha)$ и $v_y = V * sin(\alpha)$ соответственно

3) Зачем считать время внутри игры

Если следовать формулам равноускоренного движения, нам нужно время прошедшее от старта полета, поэтому в игре есть общие внутриигровые часы, реализованные благодаря классу Clock в SFML.

4) Как сделать красивое окно?

Ключевой класс SFML это RenderWindow - окно наподобие окон любой крупной программы с графическим пользовательским интерфейсом, но так как это класс, объекты этого класса при выходе из функции, где они были созданы будут уничтожены, поэтому пользовательский класс Game содержит указатель на объект этого класса, что позволяет держать его в живых столько, сколько нужно, но требует не забыть про него в деструкторе; тоже касается и игроков.

5) Один спрайт для всех

И вправду, в начале было решено, что ядер и взрывов много, и встают два вопроса как их хранить, и как рисовать?

С первым в моем случае разобрался шаблонный класс из стандартной библиотеки шаблонов STL vector вполне подходит для моих целей: добавлять новое, “гулять” по-старому, вырезать взорвавшееся. С проблемой хранения кучи спрайтов справляется SFML, позволяющий рисовать один и тот же спрайт в множестве мест окна.

6) Считывание нажатий

Управление реализовано через считывание нажатий клавиш SFML и описано ниже. Одной из проблем, возникшей в процессе написания, оказалось количество проходов по “вечному” циклу (см. ниже) за время нажатия на кнопку; в итоге одно нажатие засчитывалось за значительно большее количество. Решение оказалось достаточно простым. Уже имея внутриигровые часы, я реализовал кулдаун на нажатия.

“кулдаун от cooldown – время, в течение которого вы не сможете снова использовать способность”

Это были основные вопросы\проблемы, с которыми я столкнулся. Теперь рассмотрим всю картину в целом.

Описание классов, переменных и методов

1)Классы и методы SFML

1)sf::RenderWindow - Системное OpenGL окно в котором в дальнейшем рисуются объекты других sfml классов.

Методы:

1) draw - Добавление “Рисовабельного” объекта на окно
2) display - отображение всего ранее добавленного с draw
3) clear очистка окна (удаление всего отрисованого)
4) setPosition - установка позиции левого верхнего угла окна в координатах экрана

2)sf::Texture - изображение которое накладывается на объекты классов: Sprite, Shape итд.

Методы:

1) LoadFromFile- Загрузка картинки в текстуру

3)sf::Sprite - объект который можно рисовать в определенных координатах на окне.

Методы:

1) SetTexture - Выбор текстуры для спрайта
2) SetTextureRect - выбор куска текстуры для спрайта
3) SetScale - увеличение.уменьшение спрайта
4) SetPosition - координаты верхнего левого угла спрайта в окне
5) IntRect - класс “Куска” для вырезания спрайта с текстуры

4)sf::Clock - часы более близкие к таймеру

Методы:

1) getElapsedTime - возвращает отсчитанное время
2) restart - обнуление часов

5)sf::Time - объект хранящий время

Методы:

1) as… - возвращает числовое значение времени в мили\микро\просто секундах

6)sf::Keyboard - класс для работы с клавиатурой

Методы:

1) Key – объект хранящий данные о кнопке
2) Проверяет нажата ли кнопка bool

2) Мои Классы и их методы и данные (Сверху hpp снизу cpp файлы)

```
class Cannonball {
public:
  //Конструктор
  Cannonball(int x, int y, double speedx, double speedy, sf::Time StartTime,
double masscoof,bool LeftorRigth);
  ~Cannonball()= default;//В ядре у всего есть дефолтные деструкторы
  void Move(sf::Time T);//Функция движения
  void Draw(sf::RenderWindow&W,sf::Sprite&S);//Рисование ядра
координаты же в привате
  bool operator == (const Cannonball&canonball);//В игре сравниваем ядра
для столкновения
  bool deliteqm;//Удалять ли ядро

private:
  std::pair<double ,double >coords;//Координаты
  std::pair<double,double> speed;//Скорость по X и по Y
  std::pair<int,int>coordsbase;//Начальные координаты нужны для расчета
текущих смотри Move
  sf::Time StartTime;//Для того же расчета текущих координат
  double  masscoof;//коофицент массы для размера спрайта и урона

  //Даем другим классам право лазить куда неможно
  friend class Explosion;//Чтобы делать взрывы
  friend class Player;//Чтобы проверять получение урона
```

```
  bool Player;//Во избежание самоурона(попаданий в самого себя)
};
-------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------

class Player
{
public:
  Player(float FeeldL, float FeeldH, sf::Keyboard::Key anglePlus,
sf::Keyboard::Key angleMinus, sf::Keyboard::Key energyPlus,
        sf::Keyboard::Key energyMinus, sf::Keyboard::Key massPlus,
sf::Keyboard::Key massMinus, std::vector<Cannonball> &Cannonslink,
sf::Clock &ClockLink,
        sf::Sprite &CSP, sf::Sprite &WheelSPL, bool LeftPlayer, sf::Font
&textFont, sf::Keyboard::Key shootkey);
  void Control();// Метод управления
  void Draw(sf::RenderWindow&W,bool T);//Метод отрисовки
  ~Player();// Деструктор(Пустой, так как все переменные имеют свои
деструктооры срабатывающие автомотически
  void TrytoGetDamage();// Проверка на попадание
  bool is_deadqm();// Проверка на смерть
private:
  std::pair<double ,double >coords;// Координаты пушки
  int FeeldWidth;// Длинна игрового поля
  double  mas;// Масса будущего снаряда
  double angle;// Угол приложения энергии к снаряду
  double energymax;// Максимально возможная энергия
  double  procent;// Процент от макс. энергии приложенный к след снаряду
  void Shoot();// Метод стрельбы
  void MakeTheory();// Метод расчета теоретической траектории полета
  // Кнопки
  sf::Keyboard::Key
MassPlusskey,MassMinuskey,AnglePlusskey,AngleMinuskey,EnergyPluskey,E
nergyMinuskey,ShootKey;
  std::vector<Cannonball>&Cannonballslink;// Ссылка на вектор ядер из
класса игры
  sf::Clock&Clocklink;// Ссылка на игровые часы
```

```
  sf::Time Tcooldown;// Время последнего нажатия на кнопку нужно для ограничения скорости изменения параметров
  sf::Vertex Trace[5000];// Вектор точек теоретической кривой
  sf::Sprite CannonSP,WheelSP;// Спрайты Пушки и колес
  bool LeftPosition;// Слева игрок или справа
  int HP;// Здоровье
  sf::Text hptext,parametrs;// Текст для вывода здоровья и параметров выстрела
};
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------

class Explosion {
private:
  sf::Time LastSpriteChange;// Последнее время смены спрайта анимации
  std::pair<int,int>Coords;// Координаты взрыва
  int number_of_Sprite;// Номер спрайта
public:
  ~Explosion();//В деструкторе чистить нечего
  bool deliteqm;//Удалять ли его из вектора взрывов
  Explosion(Cannonball&canonball, sf::Clock&GameClock);//Конструктор
  void DrawNextSprite(sf::RenderWindow&Window,sf::Clock&GameClock,std::vector<sf::Sprite>&Anime);//Рисуем тот же спрайт или следующий
};
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
class Game {
  public:
     Game();// Конструктор по умолчанию
     void  Menu();// Меню
     ~Game();// Деструктор
  private:
     void MakeReadme();//Защита от дурака удаляет и создает файл с инструкцией
     std::vector<Cannonball>Cannonballs;// Вектор ядер ибо их много
     std::vector<Explosion>Explosions;// Вектор взрывов их тоже много
```

```
    std::vector<sf::Sprite>ExplosionAnime;// Анимация взрыва, здесь чтобы не хранить кучу в куче взрывов, а передовать по ссылке
    sf::RenderWindow *Gwindow;// Указатель на окно чтобы его ререндорить + так его легче хранить
    float FeeldWidth,FeeldHegth;// Размеры поля
    Player *P1,*P2;// Указатели на игроков
    sf::Clock GameClock;// Внутреигровые часы
    sf::Sprite cannonSP,cannonballSP,backgroundSP,wheelsSP;// Спрайты пушки,ядра,задник и колеса
    sf::Texture TextureCanBallWheel;// Текстура для спрайтов выше
    sf::Texture ExplosionT;// Текстура со взрывами
    sf::Texture BackGround;// Текстура задника
    bool Theory;// Рисует или нет теоретическую кривую
    sf::Font font;// Шрифт
    std::ostringstream WinStream;// Строка поздравления с победой
    sf::Text Wintext;// Текст этой строки для вывода
    void BUILD();// Настройка перед началом игры
    void Play();// Циклическая функция игры
    void Settings();// Настройки(изменение разрешения == размер поля)
};
```

Особенности

1) Анимация взрыва

В классе игры есть вектор хранящий все существующие в игре взрывы
std::vector<Explosion>Explosions;
у взрывов есть своя текстура с 9 спрайтами
все спрайты загружены в вектор в классе игры
std::vector<sf::Sprite>ExplosionAnime;// Анимация взрыва, здесь чтобы не хранить кучу в куче взрывов, а передавать по ссылке
для каждого из взрывов вызываем метод

```
DrawNextSprite(sf::RenderWindow&Window,sf::Clock&GameClock,std::vector<sf::Sprite>&Anime)
{
  if(number_of_Sprite==Anime.size()) {//Если мы все прошли то удаляем взрыв
      deliteqm = 1;
  }
  else if((GameClock.getElapsedTime()-LastSpriteChange).asMilliseconds()>50)
  {//Если прошло более 50мс то меняем кадр
      Anime[number_of_Sprite].setPosition(Coords.first,Coords.second);
      Window.draw(Anime[number_of_Sprite]);
      LastSpriteChange=GameClock.getElapsedTime();
      number_of_Sprite++;
  }
  else
  {//Иначе рисуем старый кадр
      Anime[number_of_Sprite].setPosition(Coords.first,Coords.second);
      Window.draw(Anime[number_of_Sprite]);
  }
}
```

У каждого взрыва есть время с последней смены спрайта и номер текущего из 9 спрайтов при помощи часов и времени последней смены добиваемся задержки чтобы анимация не пролетела слишком быстро.

2) Управление

В классе игрока есть метод

```
void Player::Control() {
  // Если с прошлого нажатия прошло более 300 милисекунд
  // Формально с прошлого принятия действия так как проверяется удерживается ли кнопка
  if (((Clocklink.getElapsedTime() - Tcooldown).asMilliseconds() ) > 300) {

      if ((sf::Keyboard::isKeyPressed(ShootKey))){//Если кнопка такая-то нажата/зажата то делай то
          Shoot();// Выстрел
      }
      if (sf::Keyboard::isKeyPressed(AnglePlusskey)) {
          if (angle < 90) {  // Угол не может быть более 90 градусов
              angle += 1;
          }
      }
   ……….
      Tcooldown=Clocklink.getElapsedTime();// После действия обновляем кулдаун
      MakeTheory();// Перерасчет теоретической кривой
      std::ostringstream parametrostrem;// Изменение строки параметров
      parametrostrem << "Mas: " << mas << std::endl << "Angle: " << angle << std::endl << "Energy %:" << procent * 100;
      parametrs.setString(parametrostrem.str());
  }
}
```

Проверяем, нажата ли какая-то кнопка, если да, то делаем что-то: изменяем угол\массу\энергию или стреляем;
после чего записываем время последнего принятия действия для кулдауна и меняем выводимую на экран строку с текущими параметрами и пересчитываем теоретическую кривую (см ниже).

3) Стрельба

```
void Player::Shoot() {
  double v;//Скорость
     v=(sqrt(energymax*procent* 2 / mas));//E=(mv^2)/2
  sf::Time shootTime=Clocklink.getElapsedTime();// Время выстрела нужно для расчета полета см.Cannonball
  if(LeftPosition) {
     // Если слева, то скорость считается как указано ниже; составляющие скорости по OX и OY
     Cannonball C(coords.first, coords.second, v * cos(angle * M_PI / 180), v * sin(angle * M_PI / 180), shootTime, mas / 50,LeftPosition);
     Cannonballslink.push_back(C);// Добавление ядра в вектор ядер из класса игры
  }
  else
  {
     // То же, что выше, но по Y скорость отрицательная, так как из большего Y надо попасть в 0
     Cannonball C(coords.first, coords.second, -v * cos(angle * M_PI / 180), v * sin(angle * M_PI / 180), shootTime, mas / 50,LeftPosition);
     Cannonballslink.push_back(C);
  }
}
```

Скорость считает по формуле выше (стр.2) оттуда же разделение скорости на X и Y.

Создаем ядро и добавляем его в вектор всех ядер в игре.

4) Полет ядра после выстрела и в теории

```
void Cannonball::Move(sf::Time T) {
  //Физика X=x0+Vx*t    Y=y0+Vy*t+(gt^2)/2
  coords.first=coordsbase.first+(speed.first*4*(T.asSeconds() -
StartTime.asSeconds()));
  coords.second= coordsbase.second -speed.second*4*(T.asSeconds() -
StartTime.asSeconds()) + (9.8 * pow(4*((T.asSeconds() -
StartTime.asSeconds())), 2)) / 2;
}
```

Считаем текущие координаты из координат начала и приращения координат за время со старта полета по формулам из начала стр.1

Аналогично по этим формулам считаем теоретическую кривую полета до выстрела

```
void Player::MakeTheory() {
  double v;//Скорость
  double size;//Длинна кривой
  v=(sqrt(energymax*procent* 2 / mas));//E=(mv^2)/2
  size= ((FeeldWidth / 3) - coords.first) / 200;//Методом подгона
  if(!LeftPosition)
     size=size/2;//Подгона ручками методом тыка
  if(LeftPosition)
 for(int i=0;i<5000;i++)
  { //Координата точки графика
     Trace[i].position=sf::Vector2f(coords.first + i * size / 5000 * v * cos(angle
* M_PI / 180), coords.second - i * size / 5000 * v * sin(angle * M_PI / 180) +
(9.8 * pow(i * size / 5000, 2)) / 2+15);
     Trace[i].color=sf::Color::Blue;//Цвет
  }
  if(!LeftPosition)
  for(int i=0;i<5000;i++)
  { //Скорость в разные стороны для левого и правого
     Trace[i].position=sf::Vector2f(coords.first + i * size / 5000 * v * cos(angle
* M_PI / 180), coords.second + i * size / 5000 * v * sin(angle * M_PI / 180) +
(9.8 * pow(i * size / 5000, 2)) / 2+5);
     Trace[i].color=sf::Color::Red;}}
```

У игрока есть массив точек sfml Vertex который рисуется как кривая из точек с заданными координатами.
в цикле считаем координаты Y в каждом X.

5) **Отрисовка**

Все что рисуется, рисуем с помощью Sprite::setPosition
RenderWindow::draw и RenderWindow::display

Устанавливаем позицию спрайта с помощью setPosition, потом забиваем местечко под солнцем (на экране) при помощи draw и рисуем все, у чего есть такие местечки при помощи display. Замечу, что draw может использоваться много раз для одного спрайта для получения нескольких копий, что позволяет экономить место в памяти, как например, с вектором из спрайтов взрывов использующийся для разных взрывов по ссылке.

Защита от дурака

Основным аспектом этой защиты является инкапсуляция (объединение данных и функций для работы с ними в один объект)
и следующее из нее сокрытие (данные и методы в “private:” доступны только объектам самого класса и друзьям (friend class), дружественные классы используются для перевода ядер в взрывы, для этого класс взрыва заявлен другом класса ядер.
Еще одним примером подобной защиты является эта часть строчек, где рендерится окно

```
//Style::Close == Запрет на изменение размера окна т.к тогда меняется и разрешение
 this->Gwindow=new sf::RenderWindow(sf::VideoMode(FeeldWidth, FeeldHegth), "Balistick Game", sf::Style::Close);
```

Также смотри ниже о файле ReadMe.

Тесты

1) Меню

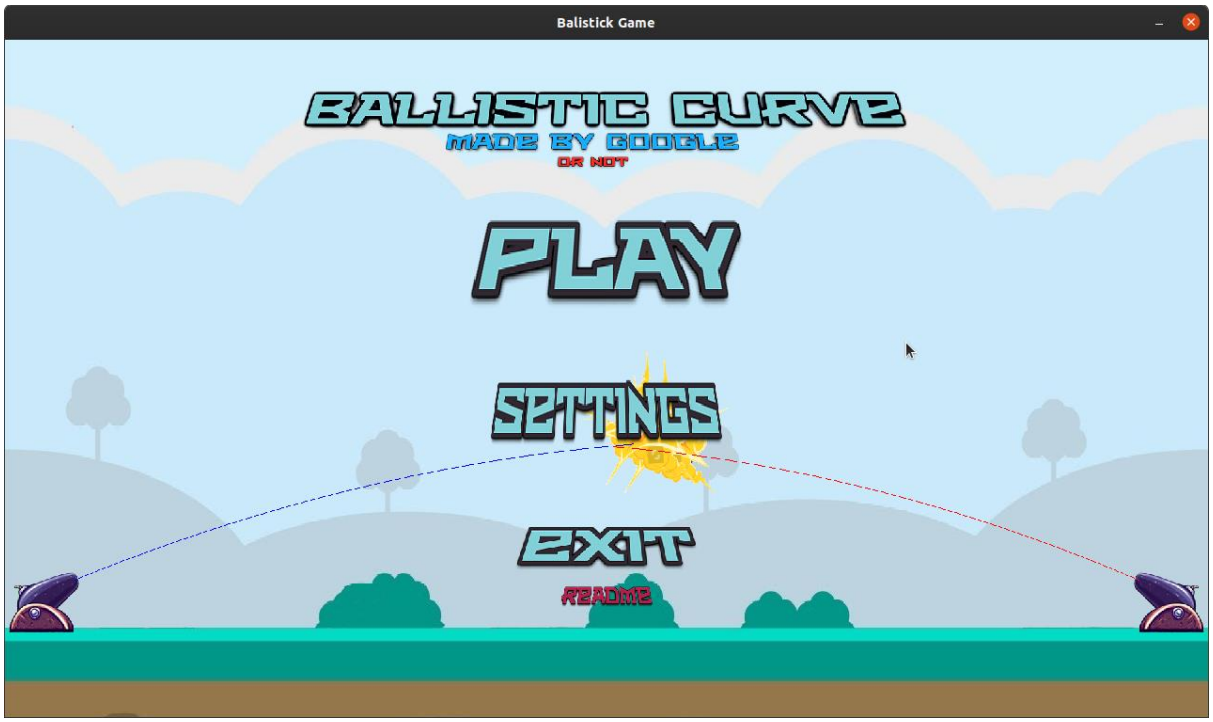


рис.1

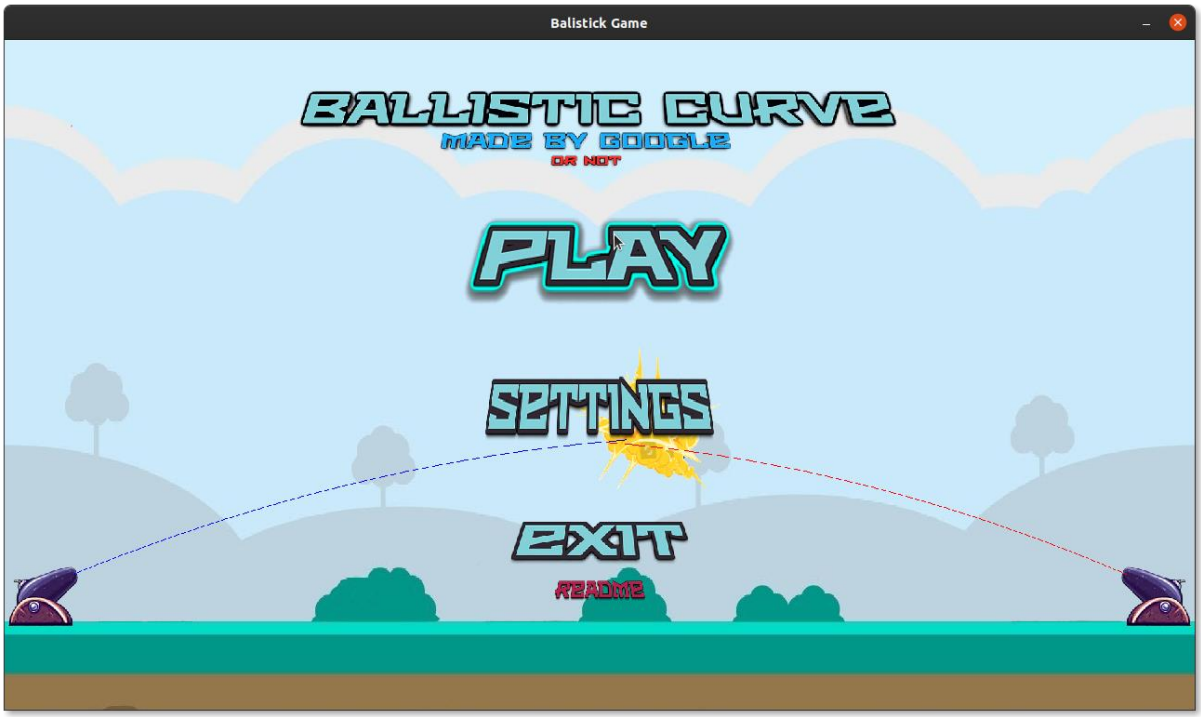


рис.2

Кнопки: Играть, Настройки, Выход и вызов файла ReadMe

На данных скриншотах видно изменение кнопок при наведении мышки.

Рассмотрим пункт настроек.

2) Настройки

Как видно ниже при выборе разрешения 720p окно меньше, чем весь мой экран (1080p)

При выборе 1080p окно игры занимает весь экран, что логично, к тому же на всех скриншотах видно отсутствие кнопки “во весь экран”, так как защита от дурака см. выше про sf::Style

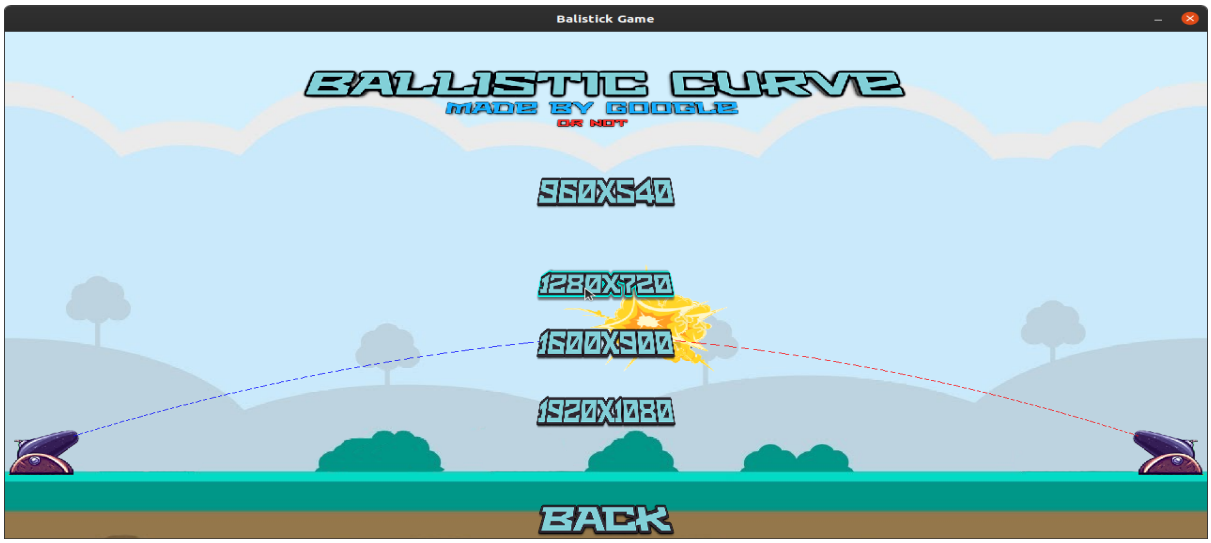


рис.3

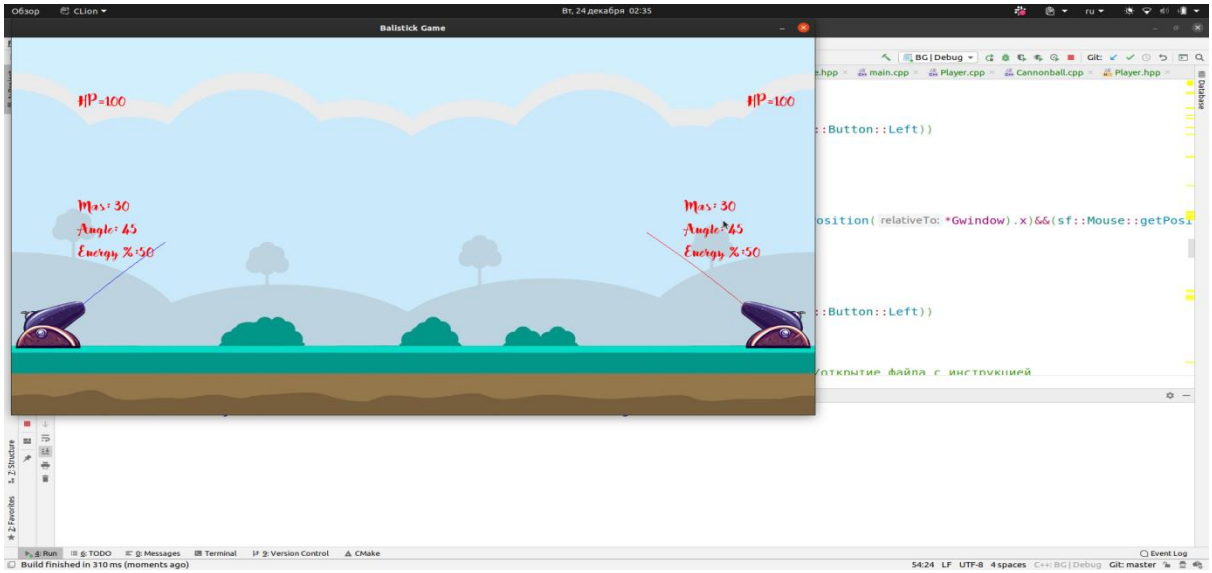


рис.4

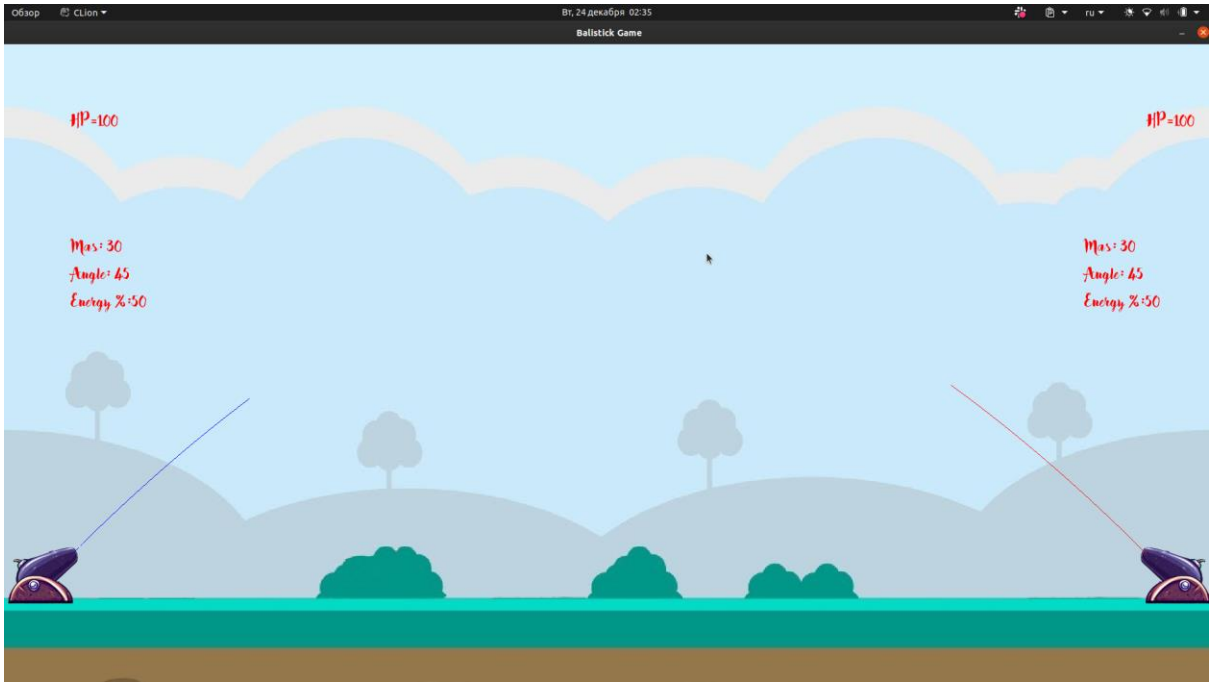


рис.5

3) ReadMe

В меню при нажатии на кнопку

```
if(sf::Mouse::isButtonPressed(sf::Mouse::Button::Left))
{
MakeReadme(); //метод создания файла
  #if defined(linux) || defined(__linux)
  std::system("gedit README.txt");//открытие файла с инструкцией
  #endif        //Попытка в мультиплатформенность
  #if defined(_WIN32) || defined (_WIN64)
  std::system ("start notepad.exe README.txt");
  #endif
}
```

В теории данная команда должна открыть файл в notepad в windows при помощи препроцессорных if ограничиваем действия по системе.

```
void Game::MakeReadme() {
  remove("README.txt");
  std::ofstream outfile ("README.txt");
  std::stringstream ss;
  ss<<"Control:\n Left:\n 1)Mas+ E\n 2)Mas- Q\n 3)Angle+ W\n
4)Angle- S\n 5)Energy+ D\n 6)Energy- A\n 7)Shoot LShift\n"
```

```
<<"Rigth:\n 1)Mas+ ]\n 2)Mas- [\n 3)Angle+ UP\n 4)Angle- Down\n 5)Energy+ Rigth\n 6)Energy- Left\n 7)Shoot RShift\nTheoryLines T";
  outfile << ss.str();
  outfile.close();
```

Здесь я просто удаляю файл, если такой есть, и потом создаю его и пишу в него именно то, что мне нужно. Защита от дурака, ибо вдруг он удалит или перезапишет файл.

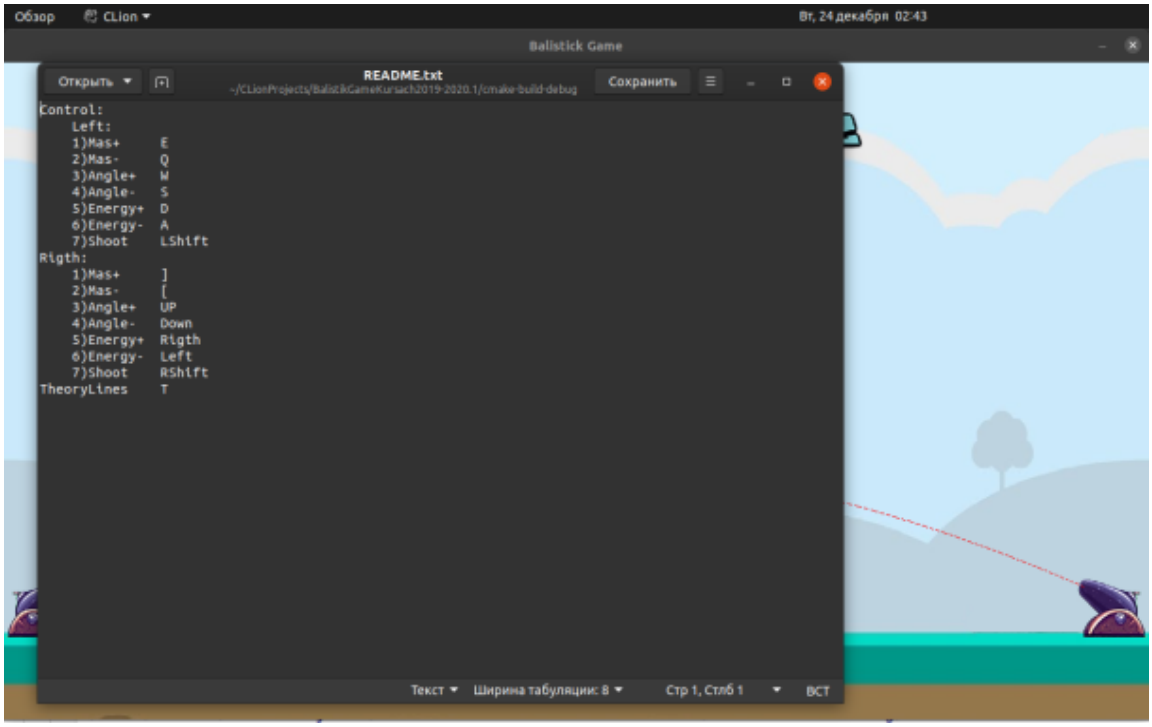
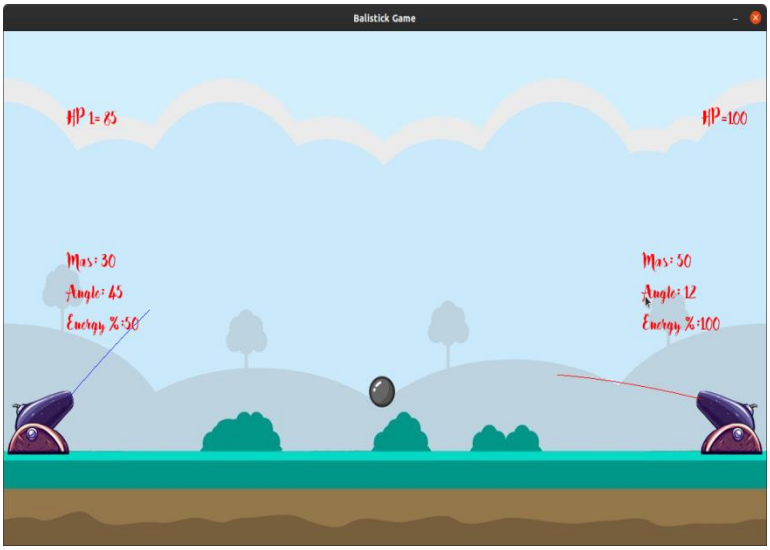


рис.6

4) Играем

Все на примере правого игрока для левого аналогично



Изменение угла, массы получение левом игроком урона, экран победы Управление см. ReadMe. Урон рассчитывается из коэффициента массы. Проблематично поймать соответственные скриншоты, но это можно проверить

рис. 7

скомпилировав проект или запустив приложенный файл через терминал линукс($ ./Linux) или запустив start.exe из папки windows (В силу нехватки некоторых dll в системе файл может не запустится, но я попытался приложить все необходимое)

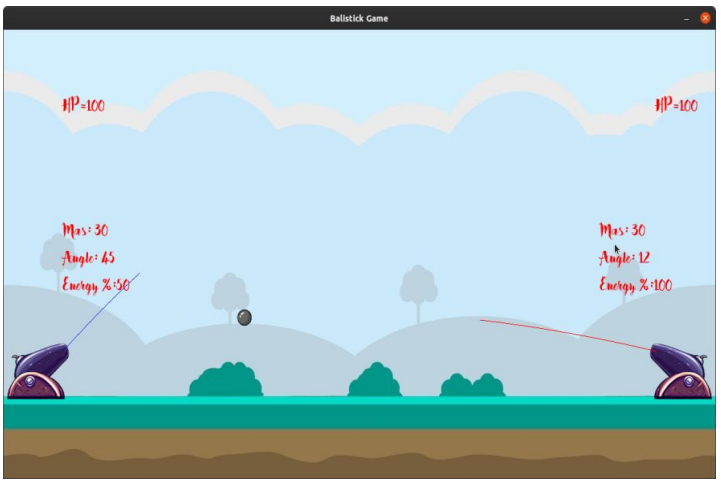


рис. 8

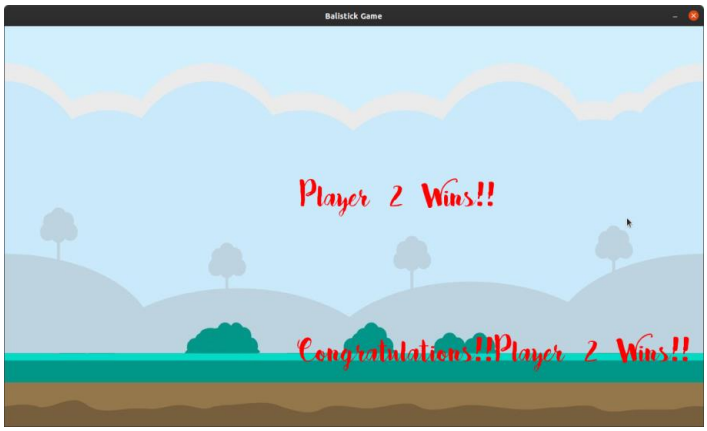


рис. 9

Заключение

В ходе выполнения данной курсовой работы была реализована “Баллистическая игра”. В ходе ее написания были на практике применены знания об основных парадигмах ООП таких как Абстракция и Инкапсуляция. Так же была на базовом уровне изучена библиотека SFML.

Выполнение курсовой работы помогло закрепить изученный материал и навыки, полученные в ходе лекций и лабораторных работах, систематизировать знание языка и научиться применять их в поставленных задачам.

Список Литературы

1) Брайан Керниган, Деннис Ритчи «Язык программирования С» - М.: ООО ―И.Д. Вильяме‖, 2017.

2) Адитья Бхаргава Грокаем Алгоритмы Илюстрированое пособие для програмистов и любопытствующих. - СПБ.: Питер: Прогресс книга, 2019.

3) Tutorials for SFML 2.5 // sfml-dev.org URL: https://www.sfml-dev.org/tutorials/2.5/ (дата обращения: 12.10.19)

4) Clion и SFML // Habr URL: https://habr.com/ru/sandbox/130510/ (дата обращения: 12.10.19).