Оглавление

[ВВЕДЕНИЕ 3](#_Toc9594698)

[1.АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc9594699)

[1.1 Карточная игра «Digger» 4](#_Toc9594700)

[1.2 Выбор языка и среды разработки 5](#_Toc9594701)

[1.3 Постановка задачи 6](#_Toc9594702)

[2.РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 7](#_Toc9594703)

[2.1 Разработка основного алгоритма программы 7](#_Toc9594704)

[2.2 Реализация основных блоков кода 8](#_Toc9594705)

[2.2.1 Класс Bag 8](#_Toc9594706)

[2.2.2 Класс Bullet 9](#_Toc9594707)

[2.2.3 Класс Cube 9](#_Toc9594708)

[2.2.4 Класс EasyEnemy 9](#_Toc9594709)

[2.2.5 Класс Emerald 10](#_Toc9594710)

[2.2.6 Класс Entity 10](#_Toc9594711)

[2.2.7 Класс Scoreboard 11](#_Toc9594712)

[2.3 Контроль программы во время её выполнения 12](#_Toc9594714)

[2.4 Визуальная составляющая приложения 20](#_Toc9594715)

[ЗАКЛЮЧЕНИЕ 26](#_Toc9594716)

# ВВЕДЕНИЕ

Компьютерные игры - чудо двадцатого века. Частью поп-культуры они стали в конце 1970-х. История компьютерных игр охватывает пять десятилетий. На их появление никто не рассчитывал. Для абсолютного большинства они оказались полной неожиданностью. Атомная бомба или полеты в космос в сознании обычного человека тоже не имели предыстории, но о них, по крайней мере, мечтали фантасты, хотя и относили время их появления на сотни лет вперед. У компьютерных игр в массовом сознании нет даже такой предыстории.

В современном мире, компьютерные игры стали не только развлечением, но и носителем культуры. Они мало похожи на балетную сцену, галерейную стену или книжную страницу, но точно так же фиксируют современную мораль, этику, иллюзии, надежды и представления о прошлом и будущем большинства людей.

Одним из способов самореализации личности и являются компьютерные игры. История компьютерных игр начинается в 1947 году с появлением ракетного стимулятора — развлекательного средства, прототипа современных компьютерных игр. И за чуть более чем 50 лет игровая индустрия пришла к невероятным успехам. Сейчас суммированные доходы компаний, занимающихся разработкой и продвижением индустрии составляют многие миллиарды в год. Для сравнения, компания Blizzard Entertainment, Inc (один из мировых лидеров в производстве компьютерных игр) заработала в 2014 году 1,72 млрд долл.

Аналогично в России, рынок игр оставил за собой звание самого значимого и перспективно развивающегося направления российского рынка цифрового контента в 2014–2015 гг. В 2014 году самым значительным сегментом игрового рынка в России были многопользовательские онлайн продукты, которые составили около 40 % всего рынка (в денежном выражении). Самым быстрорастущим сегментом оказались мобильные игры, совокупный среднегодовой темп роста которых составил 23 %. Этому способствует распространение смартфонов и планшетов, улучшение их функционала, а также расширение покрытия 3G и 4G (LTE) [2].

Как мы видим, эта огромнейшая индустрия получает прибыль не меньшую, чем в любой другой сфере. При этом она так же развивается и прогрессирует: создаются различного рода технологии (например, очки виртуальной реальности), позволяющие изучать границы человеческого сознания; создаются программы по обучению через игровой процесс людей с отклонениями в развитии и т. д.

# 1.АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Игра «Digger»

***Digger*** (рус. *Ди́ггер* — «землекоп») — аркадная компьютерная игра, выпущенная канадской компанией Windmill Software в 1983 году для компьютеров IBM PC. Является спин-оффом и клоном другой игры — *Dig Dug*

Задачей игрока является сбор под землёй золота и драгоценных изумрудов. Для этого игрок управляет моторизированной машиной «Diggermobile», которая перемещается в заброшенной шахте по добыче золота, может рыть туннели и собирать найденное. В то же время под землёй машину игрока преследуют ноббины ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Nobbins*), встреча с которыми приводит к смерти игрового персонажа. Игровой мир представлен множеством уровней, каждый из которых занимает один экран. Для прохождения уровня игроку нужно либо собрать все изумруды, либо уничтожить всех врагов. Игра идёт до тех пор, пока у игрока не закончатся все жизни[[1]](https://ru.wikipedia.org/wiki/Digger#cite_note-pcmag-1).

Каждый уровень — это прямоугольная сетка 15×10, и по данным ячейкам происходит горизонтальное или вертикальное перемещение «Diggermobile» и монстров. В начале уровня каждая ячейка либо является пустой (тоннель), либо не пустой (земля). В последних могут находиться изумруды или мешки золота. Если диггер перемещается на занятую землёй клетку, то он очищает её и тем самым прорывает туннель. Изумруды землекоп собирает автоматически, а при горизонтальном движении на мешок золота диггер пытается его толкнуть. При возможности мешок перемещается и таким образом может быть сброшен с высоты, и, если мешок пролетает более одной клетки, то в результате падения он разбивается и образуется кучка золота, которая может быть подобрана землекопом или монстрами. В то же время, падающий мешок опасен для диггера и монстров — если он задевает кого-то, то захватывает в падении с собой и разбивает о землю[[1]](https://ru.wikipedia.org/wiki/Digger#cite_note-pcmag-1).

В начале уровня диггер появляется внизу в центре экрана, а ноббины по одному появляются с правом верхнем углу. Они не могут рыть тоннели, а способны только перемещаться по существующим. Однако, данные существа могут со временем превратиться в хоббинов ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Hobbins*), которые могут рыть туннели, съедать изумруды и мешки золота, и тем самым более агрессивно преследуют землекопа. Для борьбы с монстрами «Diggermobile» может стрелять огненным шариком ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *fireball*), который летит по горизонтали или вертикали и, если встречается с монстром, то взрывается и уничтожает всех монстров в клетке (как правило, одного). После выстрела землекопу нужна перезарядка, которая происходит через некоторое время.

Новые монстры с некоторым промежутком времени появляются на уровне, и так происходит до тех пор, пока их количество не достигнет максимума для данного уровня, после чего в углу появляется «вишенка». Если её «съесть», то на некоторое время включается бонусный режим, в котором монстры и «Diggermobile» меняются ролями — до него первые преследовали землекопа, а во время его диггер может преследовать врагов и поедать их[[1]](https://ru.wikipedia.org/wiki/Digger#cite_note-pcmag-1).

От уровня к уровню игра усложняется: растёт количество монстров на уровне, как общее, так и одновременно присутствующих на экране; растёт их скорость; увеличивается время на перезарядку; уменьшается длительность бонусного режима

## 1.2 Выбор языка и среды разработки

Для написания этой курсовой работы выбран язык С++.

C++ — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

Интегрированная Среда Разработки (ИСР) − это среда, в ко- торой есть все необходимое для проектирования, запуска и тестирования приложений, где все нацелено на облегчение процесса создания программ. ИСР интегрирует в себе редактор кодов, отладчик, инструментальные панели, редактор изображений, инструментарий баз данных, т.е. все, с чем приходится работать. Результатом является быстрая разработка сложных прикладных программ. Таким образом, IDE дает возможность получить ЕХЕ файл, не используя другие программы.

Для разработки программ на языке С++ будет использована среда программирования **Qt Creator**, которая содержит в себе средства создания программы, ее компиляции, отладки и за- пуска на выполнение. В этой связи рассмотрим кратко структуру этой среды, а точнее ее интерфейс. Интерфейс − это аппарат, который позволяет удобно взаимодействовать пользователю со средой.

## 

## 1.3 Постановка задачи

В задачу курсового проекта входит разработать программное средство на языке С++, которое:

* Будет реализовывать правила игры «Digger»
* Позволяет пользователю играть с различными вариантами концовки игры

# 2.РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## 2.1 Разработка основного алгоритма программы

Основной алгоритм программы написан на языке программирования С++, который поддерживает объектно-ориентированное программирование.

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

* абстрагирование для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса;
* инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;
* наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтённое на предыдущих шагах;
* полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». Прогрессирование, в частности, на последнем этапе даёт возможность перехода на следующий уровень детализации, что замыкает общий процесс.

Обычный человеческий язык в целом отражает идеологию ООП, начиная с инкапсуляции представления о предмете в виде его имени и заканчивая полиморфизмом использования слова в переносном смысле, что в итоге развивает[3] выражение представления через имя предмета до полноценного понятия-класса.

Поэтому программа разделена на классы:

* Bag
* Bullet
* Cube
* EasyEnemy
* Emerald
* Entity
* Scoreboard

## 2.2 Реализация основных блоков кода

Так как код программы достаточно большой, то все он разделён на файлы. Каждый класс находится в отдельном юните, который состоит из h-файла и cpp-файла.

В программировании заголовочный файл (англ. header file) или подключаемый файл — файл, содержимое которого автоматически добавляется препроцессором в исходный текст в том месте, где располагается некоторая директива

В языках программирования Си и C++ заголовочные файлы — основной способ подключить к программе типы данных, структуры, прототипы функций, перечислимые типы и макросы, используемые в другом модуле. По умолчанию используется расширение .h; иногда для заголовочных файлов языка C++ используют расширение .hpp.

Заголовочный файл в общем случае может содержать любые конструкции языка программирования, но на практике исполняемый код (за исключением inline-функций в C++) в заголовочные файлы не помещают. Например, идентификаторы, которые должны быть объявлены более чем в одном файле, удобно описать в заголовочном файле, а затем его подключать по мере надобности. Подобным же образом работает модульность и в большинстве ассемблеров.

В cpp-файле находятся определения всех функций.

Будут приведены только заголовочные файлы классов.

### 2.2.1 Класс Bag

### class Bag : public virtual Entity {

### protected:

### int f\_x;

### int f\_y;

### int distance;

### public:

### Bag(MAP& s\_map, int x, int y, int lifes, std::string path, int dir);

### Bag();

### void Move(MAP& map, float time, std::vector<Bag>& bag, std::vector<EasyEnemy>& ens, Digger& player);

### bool CanMoveLeft(MAP& s\_map, std::vector<Bag>& bag, std::vector<EasyEnemy>& ens, Digger& player) ;

### bool CanMoveRight(MAP& s\_map, std::vector<Bag>& bag, std::vector<EasyEnemy>& ens, Digger& player) ;

### int GetPosFX();

### int GetPosFY();

### };

### 2.2.2 Класс Bullet

class Bullet :public virtual Entity {

public:

Bullet();

void Move(MAP& s\_map);

};

### 2.2.3 Класс Cube

class Cube {

public:

sf::Texture texture;

sf::Sprite sprite;

Cube() {}

Cube(const std::string& path);

void setPosition(float X, float Y);

};

### 2.2.4 Класс EasyEnemy

class EasyEnemy : public virtual Entity {

protected:

int Choose(MAP &map,std::vector<Bag>& bag);

public:

EasyEnemy();

EasyEnemy(MAP &s\_map,int x, int y, int lifes, std::string path, int dir);

~EasyEnemy();

void Move(MAP& map, float time, std::vector<Bag>& bag, bool difficulty);

void Respawn(int x, int y, float time);

};

### 2.2.5 Класс Emerald

class Emerald :public virtual Entity {

public:

Emerald();

Emerald(MAP& s\_map, int x, int y, int lifes, std::string path, int dir);

bool is\_alive();

};

### 2.2.6 Класс Entity

class Entity {

protected:

int m\_x, m\_y;

int m\_prevdir;

char m\_dir;

int m\_lifes;

float el\_time;

bool is\_moving;

std::stack<int> way;

Cube icon;

public:

Entity(MAP &s\_map,int x, int y,int lifes,std::string path,int dir);

Entity();

~Entity();

int GetPosX() const;

int GetPosY() const;

void SetPosX(const int x);

void SetPosY(const int y);

void SetDir(const int dir);

void SetLifes(int lifes);

void SetTime(const float time);

void SetInfo(int x, int y, int lifes, std::string path, int dir);

void setPosition(float X, float Y);

char GetDir() const;

int GetLifes();

float GetTime();

sf::Sprite sprite();

int GetBlock(char dir, MAP &s\_map,int hbx) const;

bool is\_identical(int y1, int x1, int y2, int x2);

char toChar(char& dir);

void SetMoving(bool flag);

bool Moving();

void StepAdd(const int dir, const int steps);

bool EmptyStack()const;

};

### 2.2.7 Класс Scoreboard

### class FocusObject {

### public:

### virtual ~FocusObject() = default;

### virtual void Event(const sf::Event&, int& choose) = 0;

### };

### class TextBox : public sf::Drawable, public FocusObject {

### private:

### mutable sf::Text m\_Text;

### mutable std::string m\_NewText;

### mutable bool m\_TextChanged = false;

### public:

### explicit TextBox(sf::Text Text);

### void draw(sf::RenderTarget& Render, sf::RenderStates States) const override;

### void Event(const sf::Event& event,int &choose) override;

### void SetText(const sf::String& String) const;

### std::string GetText() const;

### };

### class FocusController {

### private:

### FocusObject\* m\_Object;

### public:

### explicit FocusController(FocusObject\* Obj = nullptr);

### void SetFocusObject(FocusObject\* NewObj);

### FocusObject\* GetFocusObject() const;

### };

## 2.3 Контроль программы во время её выполнения

Все алгоритмы контроля запущенной программы находятся в файле формы и описаны в событиях.

Форма является одним из важнейших элементов приложения C++. Процесс редактирования формы происходит при добавлении к форме компонентов, изменении их свойств, создании обработчиков событий.

При добавлении компонента к форме заголовочный файл и файл формы модифицируются. При редактировании свойств компонента в Инспекторе объектов эти изменения сохраняются в файле формы.

События. Страница событий (Events) Инспектора объектов показывает список событий, распознаваемых компонентом и возникающих при изменении состояния компонента. Каждый экземпляр компонента имеет свой собственный набор функций - обработчиков событий. Создавая обработчик события, вы поручаете программе выполнить указанную функцию, если это событие произойдет. Чтобы добавить обработчик события, нужно выбрать компонент, затем открыть страницу событий Инспектора объектов и дважды щелкнуть левой клавишей мыши рядом с событием. Это заставит C++ Builder сгенерировать текст пустой функции с курсором в том месте, где следует вводить код. Далее нужно ввести код, который должен выполняться при наступлении данного события.

#include "Bag.h"

#include "EasyEnemy.h"

#include "Entity.h"

#include "Bullet.h"

#include "Emerald.h"

class Bag;

class EasyEnemy;

class Digger : public virtual Entity {

protected:

int m\_emerald\_count;

public:

Digger();

Digger(MAP& s\_map, int x, int y, int lifes, std::string path, int dir);

~Digger();

void Move(char dir, MAP& s\_map, std::vector<Bag>& bag, std::vector<EasyEnemy>& ens);

int GetEmCount();

void collision(Bullet& bullet, std::vector<EasyEnemy>& ens, std::vector<Emerald>& ems, std::vector<Bag>& bag, std::string type1, std::string type2, int& Score);

void Entity\_Collision(Bullet& bullet, std::vector<EasyEnemy>& ens, std::vector<Emerald>& ems, std::vector<Bag>& bag, int& Score);

void Respawn(int x, int y, float time);

void SetEmCount(const int count);

};

void Digger::Move(char dir, MAP& s\_map, std::vector<Bag>& bag, std::vector<EasyEnemy>& ens) {

bool border\_ahead = false;

if (dir == 0) return;

toChar(dir);

//Use recursion if player coordinates is not % 4

if (el\_time > 0.05) {

int block = GetBlock(dir, s\_map, 4);

if (block == 0)

border\_ahead = true;

switch (dir) {

case 1:

if (m\_x % 4 != 0 || border\_ahead) {

Move(m\_x % 4 == 1 ? dirs::left : m\_x % 4 == 3 ? dirs::right : m\_x % 4 == 2 ? m\_prevdir : 0, s\_map, bag, ens);

m\_x % 4 == 1 ? m\_prevdir = dirs::up : m\_x % 4 == 3 ? m\_prevdir = dirs::up : m\_prevdir = m\_prevdir;

return;

}

//Can not move into the bag

for (int i = 0; i < bag.size(); i++) {

if (is\_identical(m\_x, m\_y - 4, bag.at(i).GetPosX(), bag.at(i).GetPosY())) {

return;

}

}

///////////////////////////////////////

m\_y--;

for (int i = 0; i < 4; i++)

s\_map.map[m\_y][m\_x + i] = 17;

break;

case 2:

if (m\_y % 4 != 0 || border\_ahead) {

Move(m\_y % 4 == 1 ? dirs::up : m\_y % 4 == 3 ? dirs::down : m\_y % 4 == 2 ? m\_prevdir : 0, s\_map, bag, ens);

m\_y % 4 == 1 ? m\_prevdir = dirs::left : m\_y % 4 == 3 ? m\_prevdir = dirs::left : m\_prevdir = m\_prevdir;

return;

}

//Can not move bag & our player, if after bag is another bag or easy\_enemy

for (int i = 0; i < bag.size(); i++) {

for (int j = -3; j < 4; j++) {

if (is\_identical(m\_x - 4, m\_y + j, bag.at(i).GetPosX(), bag.at(i).GetPosY()) && bag.at(i).Moving()) {

return;

}

}

if (is\_identical(m\_x - 4, m\_y, bag.at(i).GetPosX(), bag.at(i).GetPosY()) && !bag.at(i).Moving()) {

if (bag.at(i).EmptyStack()) {

bag.at(i).StepAdd(2, 4);

}

return;

}

}

/////////////////////////////////////////////

icon.sprite.setTextureRect(sf::IntRect(32, 0, -32, 32));

m\_x--;

for (int i = 0; i < 4; i++)

s\_map.map[m\_y + i][m\_x] = 17;

break;

case 3:

if (m\_y % 4 != 0 || border\_ahead) {

Move(m\_y % 4 == 1 ? dirs::up : m\_y % 4 == 3 ? dirs::down : m\_y % 4 == 2 ? m\_prevdir : 0, s\_map, bag, ens);

m\_y % 4 == 1 ? m\_prevdir = dirs::right : m\_y % 4 == 3 ? m\_prevdir = dirs::right : m\_prevdir = m\_prevdir;

return;

}

//Can not move bag & our player, if after bag is another bag or easy\_enemy

for (int i = 0; i < bag.size(); i++) {

for (int j = -3; j < 4; j++) {

if (is\_identical(m\_x + 4, m\_y + j, bag.at(i).GetPosX(), bag.at(i).GetPosY()) && bag.at(i).Moving()) {

return;

}

}

if (is\_identical(m\_x + 4, m\_y, bag.at(i).GetPosX(), bag.at(i).GetPosY()) && !bag.at(i).Moving()) {

if (bag.at(i).EmptyStack()) {

bag.at(i).StepAdd(3, 4);

}

return;

}

}

///////////////////////////////////////////////////////

icon.sprite.setTextureRect(sf::IntRect(0, 0, 32, 32));

m\_x++;

for (int i = 0; i < 4; i++)

s\_map.map[m\_y + i][m\_x + 3] = 17;

break;

case 4:

if (m\_x % 4 != 0 || border\_ahead) {

Move(m\_x % 4 == 1 ? dirs::left : m\_x % 4 == 3 ? dirs::right : m\_x % 4 == 2 ? m\_prevdir : 0, s\_map, bag, ens);

m\_x % 4 == 1 ? m\_prevdir = dirs::down : m\_x % 4 == 3 ? m\_prevdir = dirs::down : m\_prevdir = m\_prevdir;

return;

}

//Can not move into the bag

for (int i = 0; i < bag.size(); i++) {

if (is\_identical(m\_x, m\_y + 4, bag.at(i).GetPosX(), bag.at(i).GetPosY())) {

return;

}

}

////////////////////////////////

m\_y++;

for (int i = 0; i < 4; i++)

s\_map.map[m\_y + 3][m\_x + i] = 17;

break;

}

//Settint our previous direction

switch (dir) {

case 1:

m\_prevdir = m\_dir = dirs::up;

break;

case 2:

m\_prevdir = m\_dir = dirs::left;

break;

case 3:

m\_prevdir = m\_dir = dirs::right;

break;

case 4:

m\_prevdir = m\_dir = dirs::down;

break;

}

el\_time = 0;

setPosition(static\_cast<float>(m\_x) \* 8.f, static\_cast<float>(m\_y) \* 8.f + 32.f);

}

}

// ---------------------------------------------------------------------------

void Digger::collision(Bullet& bullet, std::vector<EasyEnemy>& ens, std::vector<Emerald>& ems, std::vector<Bag>& bag, std::string type1, std::string type2, int& Score) {

//Collisions & killing our entities

if (type1 == "player" && type2 == "emerald") {

for (int j = 0; j < ems.size(); j++) {

if (ems.at(j).GetLifes() != 0) {

if (is\_identical(m\_y - 2, m\_x, ems.at(j).GetPosY(), ems.at(j).GetPosX()) ||

is\_identical(m\_y, m\_x - 2, ems.at(j).GetPosY(), ems.at(j).GetPosX()) ||

is\_identical(m\_y, m\_x + 2, ems.at(j).GetPosY(), ems.at(j).GetPosX()) ||

is\_identical(m\_y + 2, m\_x, ems.at(j).GetPosY(), ems.at(j).GetPosX())) {

Score += 25;

m\_emerald\_count++;

ems.at(j).SetLifes(0);

return;

}

}

}

}

if (type1 == "player" && type2 == "enemy") {

for (int j = 0; j < ens.size(); j++) {

if(ens.at(j).GetLifes()){

for (int i = -2; i < 3; i++) {

for (int f = -2; f < 3; f++) {

if (is\_identical(m\_y + i, m\_x + f, ens.at(j).GetPosY(), ens.at(j).GetPosX()) && ens.at(j).GetLifes()) {

m\_lifes--;

for (int h = 0; h < ens.size(); h++) {

ens.at(h).SetLifes(0);

ens.at(h).SetTime(0);

}

for (int h = 0; h < bag.size(); h++) {

if (bag.at(h).Moving()) {

bag.at(h).SetMoving(0);

bag.at(h).SetLifes(0);

}

}

return;

}

}

}

}

}

}

if (type1 == "player" && type2 == "bag") {

for (int j = 0; j < bag.size(); j++) {

if (bag.at(j).GetLifes()>0) {

for (int f = -3; f < 4; f++) {

if (is\_identical(m\_y - 2, m\_x + f, bag.at(j).GetPosY(), bag.at(j).GetPosX())) {

m\_lifes--;

for (int h = 0; h < ens.size(); h++) {

ens.at(h).SetLifes(0);

}

for (int h = 0; h < bag.size(); h++) {

if (bag.at(h).Moving()) {

bag.at(h).SetMoving(false);

bag.at(h).SetLifes(0);

}

}

return;

}

}

}

}

}

if (type1 == "enemy" && type2 == "bag") {

for (int j = 0; j < bag.size(); j++) {

for (int k = 0; k < ens.size(); k++) {

for (int f = -(ens.at(k).GetPosX() % 4); f < 2; f++) {

if (bag.at(j).GetLifes() > 0) {

if (is\_identical(ens.at(k).GetPosY() - 3, ens.at(k).GetPosX() + f, bag.at(j).GetPosY(), bag.at(j).GetPosX())) {

if (ens.at(k).GetLifes()) {

Score += 100;

}

ens.at(k).SetLifes(0);

break;

}

}

}

}

}

}

if (type1 == "enemy" && type2 == "bullet") {

for (int j = 0; j < ens.size(); j++) {

if (ens.at(j).GetLifes()) {

for (int i = -2; i < 3; i++) {

for (int f = -2; f < 3; f++) {

if (is\_identical(bullet.GetPosY() + i, bullet.GetPosX() + f, ens.at(j).GetPosY(), ens.at(j).GetPosX()) && bullet.GetLifes()) {

if (ens.at(j).GetLifes()) {

Score += 50;

}

bullet.SetLifes(0);

ens.at(j).SetLifes(0);

return;

}

}

}

}

}

}

if (type1 == "bag" && type2 == "bullet") {

for (int j = 0; j < bag.size(); j++) {

if (bullet.GetLifes()) {

for (int i = -2; i < 3; i++) {

for (int f = -2; f < 3; f++) {

if (is\_identical(bullet.GetPosY() + i, bullet.GetPosX() + f, bag.at(j).GetPosY(), bag.at(j).GetPosX()) && bag.at(j).GetLifes()) {

bullet.SetLifes(0);

return;

}

}

}

}

}

}

}

// ---------------------------------------------------------------------------

void Digger::Entity\_Collision(Bullet& bullet, std::vector<EasyEnemy>& ens, std::vector<Emerald>& ems, std::vector<Bag>& bag, int& Score) {

collision(bullet, ens, ems, bag, "player", "emerald", Score);

collision(bullet, ens, ems, bag, "player", "bag", Score);

collision(bullet, ens, ems, bag, "player", "enemy", Score);

collision(bullet, ens, ems, bag, "enemy", "bag", Score);

collision(bullet, ens, ems, bag, "enemy", "bullet", Score);

collision(bullet, ens, ems, bag, "bag", "bullet", Score);

}

// ---------------------------------------------------------------------------

void Digger::Respawn(int x, int y, float time) {

//Delay our player after his dead

el\_time = time; m\_x = y; m\_y = x;

setPosition(static\_cast<float>(m\_x) \* 8.f, static\_cast<float>(m\_y) \* 8.f + 32.f);

}// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::Protection(TObject \*Sender) {

TImage\*tmp = (TImage\*)Sender;

tmp->Width = WIDTH;

tmp->Height = HEIGHT;

Card card = Hum.Search(tmp);

if (!Buf.ProtectionCard(card)) {

return;

}

Hum.Delete(card);

Hum.Show();

if (Hum.GetSize() != 0) {

Robot.Attack(Buf);

Robot.Show();

}

}

// ---------------------------------------------------------------------------

void \_\_fastcall TForm1::Timer1Timer(TObject \*Sender) {

Buf.SetPosition(turn);

if (turn) {

if (int a = Buf.GetSizeA() != 0) {

Button2->Enabled = true;

}

Button1->Enabled = false;

}

else {

Button2->Enabled = false;

if (Buf.GetSizeA() != 0) {

Button1->Enabled = true;

}

}

if (D.GetCount() == 0 && Hum.GetSize() == 0) {

Timer1->Enabled = false;

ShowMessage("Вы победили");

exit(1);

}

if (D.GetCount() == 0 && Robot.GetSize() == 0) {

Timer1->Enabled = false;

ShowMessage("Вы проиграли");

exit(1);

}

}

// ---------------------------------------------------------------------------

void Digger::SetEmCount(const int count) { m\_emerald\_count = count; }//

## 2.4 Визуальная составляющая приложения

Интерфейс приложения создаётся с помощью компонентов

Компоненты C++. Создание пользовательского интерфейса приложения заключается в добавлении в окно формы объектов, называемых компонентами. C++ позволяет разработчику создавать собственные компоненты и настраивать Палитру компонентов.

Компоненты разделяются на видимые (визуальные) и невидимые (невизуальные). Визуальные компоненты появляются как во время выполнения, так и во время проектирования. Невизуальные компоненты появляются во время проектирования как пиктограммы на форме. Они не видны во время выполнения, но обладают функциональностью. Для добавления компонента в форму можно выбрать мышью нужный компонент в Палитре компонентов и щелкнуть левой клавишей мыши в нужном месте проектируемой формы. Компонент появится на форме, и далее его можно перемещать и изменять. Каждый компонент C++ имеет три характеристики: свойства, события и методы.

Инспектор объектов автоматически показывает свойства и события, которые могут быть использованы с компонентом. Свойства являются атрибутами ком- понента, определяющими его внешний вид и поведение. Инспектор объ- ектов отображает опубликованные (published) свойства компонентов на странице свойств (properties) и используется для установки published- свойств во время проектирования. Для изменения свойств компонента во время выполнения приложения нужно добавить соответствующий код. Помимо published-свойств, компоненты могут иметь открытые (public) свойства, которые доступны только во время выполнения приложения.

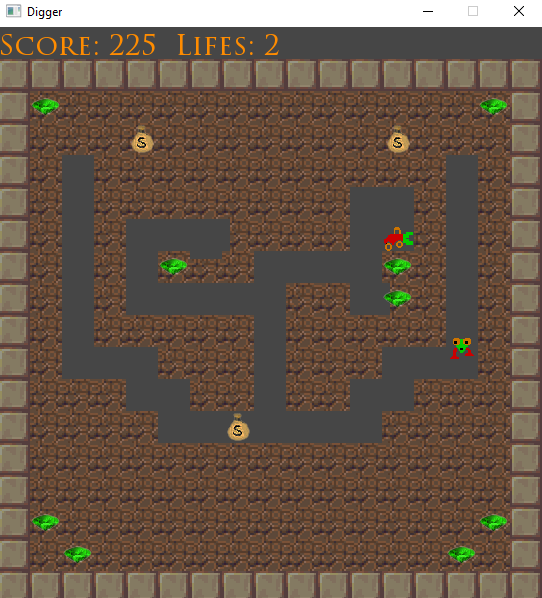
На форму добавлен набор компонентов:

* Панели(Panel)
* Кнопки(Button)
* Многострочный редактор(Memo)
* Картинки(Image)
* Текст(Edit)

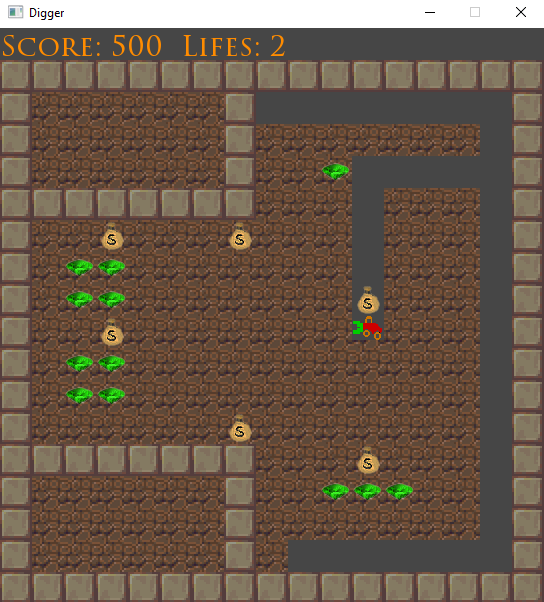
Главная форма



Процесс игры



Форма с первым вариантом концовки игры



Форма со вторым вариантом концовки игры



# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта была разработана игра «Digger». В ходе его разработки были получены теоретические и практические знания по языку программирования С++, а также по принципам объектно-ориентированного программирование.