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 “GEOMETRY DASH”

|  |  |  |
| --- | --- | --- |
| Студент |  | Гладкий М.Г. |
| Руководитель |  | Шулицкий Д.С. |

Минск 2019

СОДЕРЖАНИЕ

Введение...................................................................................................................3

1. Анализ предметной области...............................................................................6

1.1 Обзор аналогов..............................................................................................6

1.2 Постановка задачи.......................................................................................10

2. Разработка программного средства.................................................................11

2.1 Структура программы.................................................................................11

2.2 Интерфейс программного средства...........................................................11

2.3 Звуковые ресурсы........................................................................................13

2.4 Игровая логика.............................................................................................14

2.5 Внешний вид................................................................................................22

3. Тестирование программного средства............................................................23

4. Руководство пользователя................................................................................26

4.1 Правила игры...............................................................................................26

4.2 Интерфейс программы................................................................................27

Заключение.............................................................................................................29

Список использованных источников...................................................................30

Приложение А. Исходный код программы.........................................................31

ВВЕДЕНИЕ

21-й век считается временем развития IT-индустрии. Данные технологии всё больше внедряются в отрасли современной жизни. Также они не обошли сферу развлечений, в которую добавили нечто новое – видеоигры.

История компьютерных игр начинается ещё в далёких 1950-х с разработки простейших игр и симуляций. Данный вид времяпрепровождения не сразу стал популярным, потребовалось несколько десятков лет, чтобы он вышел в свет и стал частью жизни молодёжи.

Первая игра для компьютера Nim и игровой автомат Nimatron были созданы в 1940-м году. Игрок, делая ход, мог погасить одну или более ламп в одном из рядов; далее наступал такой же ход компьютера. Тот, кто выключал последнюю лампу, выигрывал. Изобретение компьютерных игр обычно приписывают кому-то из троих людей: Ральфу Баэру, инженеру, выдвинувшему в 1951-м году идею интерактивного телевидения, А. С. Дугласу, написавшему в 1952-м году OXO – компьютерную реализацию крестиков-ноликов, или Уильяму Хигинботаму, создавшему в 1958-м году игру Tennis for Two.

Spacewar! – одна из первых известных цифровых видеоигр. Она была выпущена в 1962-м. В данной игре два самолета на экране обстреливали друг друга, что задало такой жанр как аркада. Данный жанр долгое время оставался популярным и задал направление будущим компьютерным играм, однако даже сейчас существует много аркадных игр, которые до сих пор притягивают внимание игроков по всему миру.

В сентябре 1971-го года Биллом Питтсом был создан первый аркадный автомат Galaxy Game на базе PDP-11. В ноябре фирма Nutting Associates выпустила около 1500 аркадных автоматов Computer Space, разработанных Ноланом Бушнеллом и Тедом Дабни, из которых были проданы от 500 до 1000 штук. Таким образом Computer Space стала первой видеоигрой, изданной для широкой публики.

24-го мая 1972-го была впервые представлена и продемонстрирована публике Magnavox Odyssey – первая игровая приставка. Она подключалась к телевизору, вместе с ней использовались два игровых контроллера. Приставка способна выводить на экран три квадратные точки, поведение которых меняется в зависимости от установленной игры. Игроки могут размещать на экране телевизора цветные пластиковые накладки для создания визуальных эффектов. В каждой игре один или два игрока управляют своими точками с помощью контроллеров.

В дальнейшем компьютерные игры получили большое развитие. Компании по всему миру соревновались в выпуске передовых игр, приставок и консолей. Наблюдались как упадки, так и резкие возрастания популярности видеоигр. Однако индустрия никогда не стояла на месте. Разработчики всё время пытались привнести что-то новое, создать нечто уникальное. По сей день существует конкуренция между ведущими компаниями в данной сфере, однако с быстрым развитием технологий видеоигры появились и на мобильных устройствах, что привело к их бурному росту на данной платформе. В настоящее время не все могут позволить себе мощный компьютер, который позволял бы комфортно играть в современные игры. На помощь этому и пришли разработчики игр на мобильных устройствах. Почти каждый человек имеет смартфон, который поддерживает множество приложений, в том числе находятся мобильные игры. Для многих поиграть на телефоне намного проще, нежели поиграть на компьютере, ввиду нехватки времени, материальных средств и многих других причин. Большинство людей используют игры по пути на работу, в долгой дороге или когда им абсолютно нечем заняться.

Первая игра на мобильном устройстве появилась в 1993-м году, на первом в мире смартфоне Simon Personal Communicator, разработанном компаниями IBM и BellSouth. Этой игрой являлась Scramble – разновидность пазла, в котором было необходимо передвигать части изображения, чтобы составить цельную картинку.

Второй смартфон с предустановленной игрой появился в 1994-м году. Это была версия Тетриса на устройстве датской разработки под названием Hagenuk MT-2000. Суть игры заключалась в том, что сверху в прямоугольный стакан падали случайные фигурки тетрамино. В полёте игрок мог поворачивать фигурку, двигать её по горизонтали, а также ускорять её падение, когда решено, куда фигурка должна упасть. Игра заканчивается, когда фигурка не может поместиться в стакан. Игрок получает очки за каждый заполненный ряд, который пропадает после его горизонтального заполнения, а остальные блоки сдвигаются вниз. Поэтому задача пользователя – набрать как можно больше очков, не заполняя стакан по вертикали как можно дольше.

Одной из известнейших и узнаваемых игр на телефоне является Змейка или Snake. В ней игрок управляет тонким существом, напоминающим змею, которое ползает по плоскости, собирая еду, избегая столкновения с собственным хвостом и краями игрового поля. В некоторых вариантах на поле присутствуют дополнительные препятствия. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. Первым мобильным устройством с этой игрой стала Nokia 6110, выпущенная в 1997 году. С тех пор долгое время каждый телефон компании Nokia поставлялся с данным приложением.

Помимо этого существует не менее известная и популярная игра под названием Bounce, вышедшая в 2001-м году на телефонах Nokia. В ней игрок управляет красным мячиком, проходя различные уровни, в которых необходимо избегать попадания на шипы и собирать монеты.

С того момента мобильные устройства продвинулись намного вперед. Мощности некоторых телефонов сравнимы с мощностями ноутбуков, а возможно и старых компьютеров. Были созданы новые мощные процессоры, которые подходят для портативных устройств, каждый день выпускаются планшеты, которые имеют больший экран, следовательно более удобны для чтения книг, прохождения игр и много другого. Появились операционные системы Android, iOS, в которых пользователю даётся возможность устанавливать любые приложения на свой телефон или планшет, начиная от будильника, заканчивая программами для обработки фотографий. Это подтолкнуло множество разработчиков внести свой вклад в создание игровых приложений, которые помогут пользователю скрасить своё время препровождение. Появилось множество жанров игр, каждый из которых уникален.

На данный момент существует большое количество компаний, создающих свои приложения на портативные устройства. В их числе: Ketchapp, Voodoo, Supercell, Rovio, Nekki и др. Программисты создают нечто новое, каждый день работая над совершенными проектами. Это даёт пользователям огромный выбор игр, позволяя найти что-то, что подойдёт каждому в отдельности. Некоторые компьютерные игры портируются на мобильные устройства, что позволяет геймерам проводить в них время не только дома за компьютером, но и в дороге, на перерывах на работе, учёбе.

Шведский разработчик Роберт Топала, выступающий под псевдонимом RobTop, не стал исключением, написав видеоигру в жанре 2D-платформера, выпущенную изначально для мобильных устройств, а позже и для настольных компьютеров, под названием “Geometry Dash”. Суть этой игры заключается в том, чтобы проходить предложенные уровни, открывать новых персонажей и режимы прохождения. С каждым последующим уровнем сложность повышается и игроку требуется всё больше внимательности и реакции на их прохождение.

Целью данного проекта является разработка игрового приложения “Geometry Dash”.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

На 2019-й год существует множество вариаций “Geometry Dash”, отличающихся от оригинала графической составляющей, режимами игры, а также своими уникальными особенностями.

Первая версия игры появилась 13-го августа 2013-го года на мобильных устройствах на платформе Android / iOS. С течением времени она претерпела большие изменения: были добавлены новые уровни, улучшен редактор пользовательских карт, исправлены ошибки, приводящие к досрочному закрытию. 23-го декабря 2014-го года видеоигра вышла в свет на настольных компьютерах в магазине Steam.

Одним из аналогов является игра “BIT.TRIP RUNNER”, которая представляет собой 2D раннер с бесконечным генерирующимся уровнем, а также бонусами, помогающими игроку в продвижении. Внешний вид игры показан ниже:



Рисунок 1.1 – Внешний вид “BIT.TRIP RUNNER”.

Существуют и другие подобия выбранной игры, к примеру “Canabalt”. В ней игрок управляет персонажем, прыгающим по крышам и уворачивающимся от препятствий и помех, возникающих на его пути при прохождении уровней. Скриншот игрового процесса прилагается на рисунке 1.2.

Одним из клонов стала “The Impossible Game”. Игра отличается от оригинала лишь графикой, однако на уровне располагается множество ловушек и потайных ходов, что требует от игрока запоминания их расположения и совершения всё большего числа попыток. Пример геймплея на рисунке 1.3.



Рисунок 1.2 – Игровой процесс “Canabalt”

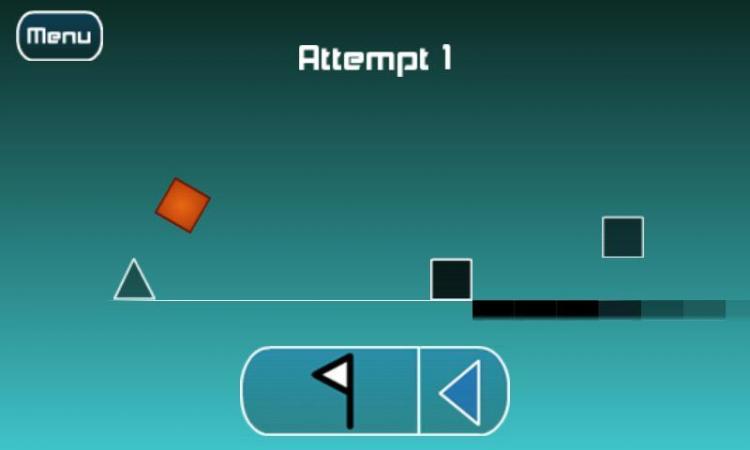


Рисунок 1.3 – Геймплей “The Impossible Game”

Не менее популярным аналогом является игра под названием “Hill Climb Racing”. В ней игроку предстоит проходить различные карты на различных средствах передвижения, начиная от старого пикапа, заканчивая танком. Во время прохождения пользователь зарабатывает монеты, которые может потратить на покупку улучшений для своего транспорта, покупку другой модели или открытие новой карты для игры. Скриншот из приложения на рисунке 1.4:



Рисунок 1.4 – Скриншот из “Hill Climb Racing”

Одной из похожих игр является классическая “Mario”. Игры основной серии выполнены в жанре платформера, в которых необходимо проходить уровни один за другом, преодолевая препятствия, собирая монеты и побеждая врагов. В конце игрока ждёт сражение с боссом, после которого игра считается пройденной. Игровой процесс показан ниже:

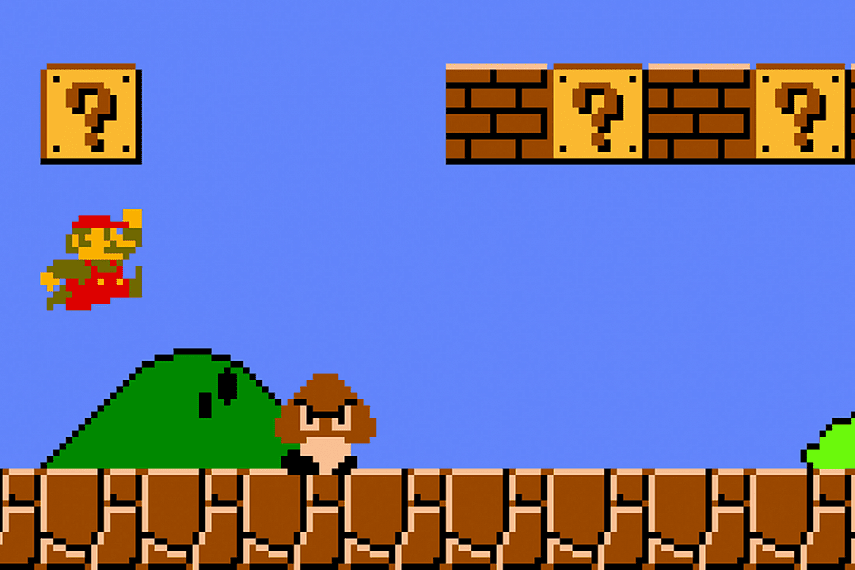


Рисунок 1.5 – Игровой процесс “Mario”

Подобной игрой в жанре платформер является популярная классическая “Super Meat Boy”. Геймплей и графика игры выполнены в стилистике старых консольных платформеров, где от игрока требуется высокая точность управления персонажем. Суть игры же заключается в том, чтобы провести персонажа к его подруге, минуя множество разнообразных ловушек, таких как циркулярные пилы, иглы. Внешний вид игры показан на рисунке 1.6:

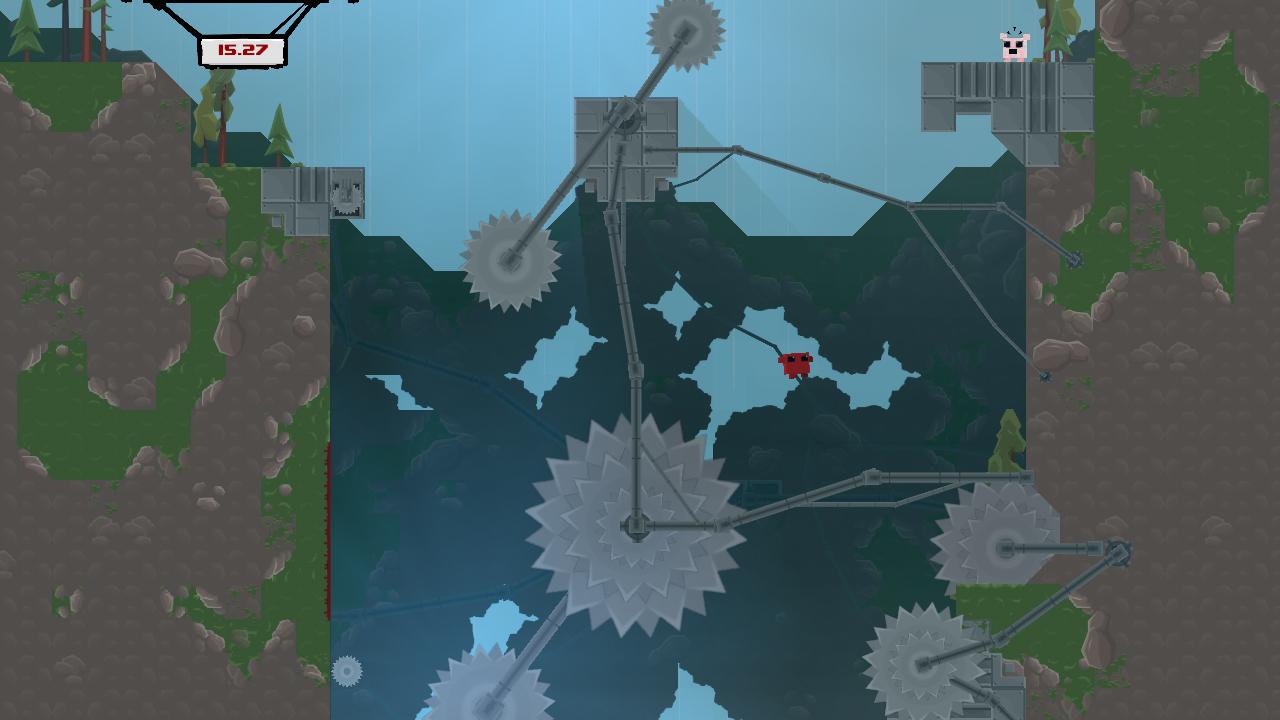


Рисунок 1.6 – Внешний вид “Super Meat Boy”

Все вышеперечисленные игры и по сей день привлекают множество геймеров, проводящих в них большое количество времени. Жанр платформер всегда был и будет популярным, а разработчики никогда не стоят на месте и постоянно привносят в него нечто новое.

1.2 Постановка задачи

В рамках данного курсового проекта планируется разработка игрового программного средства “Geometry Dash”.

В процессе реализации программы будет разработано главное меню, музыкальное сопровождение.

Планируются следующие функции:

– выбор игрового уровня;

– изменение внешнего вида игрока;

– возможность отключения музыки;

– отображение пройденных уровней;

– сохранение игрового процесса;

– приостановка игры.

Логическая часть приложения будет представлена в виде:

– взаимодействия структурных блоков;

– построения игровых уровней;

– алгоритмов столкновения и прыжка.

Для разработки программного средства будет использоваться язык программирования C++ и среда разработки Embarcadero C++ Builder 10.2.

2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Структура программы

В приложении используются 3 структурных блока:

– Main – модуль основной логики программы;

– Settings – модуль настроек игры;

– Info – модуль с правилами для ознакомления.

2.2 Интерфейс программного средства

Оформление программного средства и удобство использования являются главными качествами, которые необходимы для работы в приложении. В связи с этим, взаимодействие с пользователем нужно организовать интуитивно понятным и удобным образом.

В качестве навигации был выбран простой компонент TButton. Для взаимодействия с пользователем установлены следующие опции:

– “Play”

– “Settings”

– “Info”

На рисунке 2.1 представлено главное окно программы.

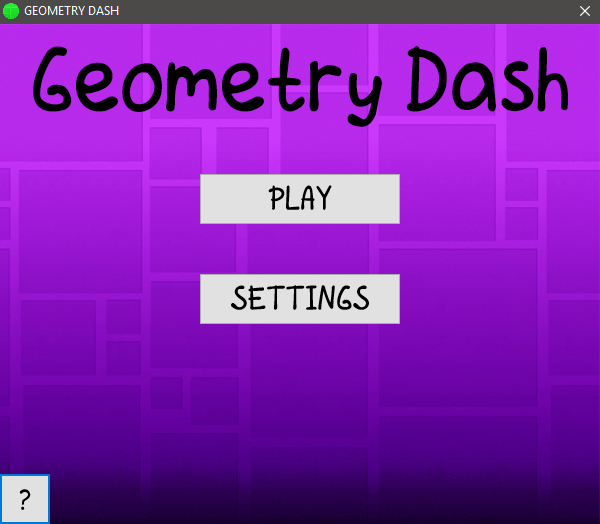


Рисунок 2.1 – Главное окно программы

После главного меню игрок попадает в окно выбора карты, на котором представлена картинка предпросмотра карты, а также кнопки для выбора желаемого уровня, возврата в главное меню и запуска игрового процесса. Данное окно отображено на рисунке 2.2:



Рисунок 2.2 – Окно выбора карты

В главном меню имеется кнопка перехода к настройкам игры, в которых пользователь может заглушить музыку и выбрать персонажа для прохождения. Внешний вид меню “Settings” указан далее:

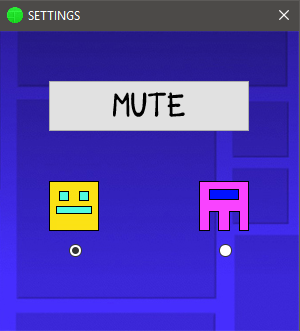


Рисунок 2.3 – Внешний вид меню “Settings”

Для поддержания цветовой гаммы и стилистики игры был использован шрифт Asinastra, являющийся стандартным в среде разработки Embarcadero С++ Builder 10.2 Tokyo, 30-го размера и чёрного цвета.

Эти настройки сделали интерфейс программного средства понятным и простым, но в то же время красивым и уместным.

2.3 Звуковые ресурсы

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

Управление звуком реализовано при помощи компонента TMediaPlayer. При этом используются такие свойства, как: FileName, Volume, CurrentTime, Play, Duration и др.

Различные случаи использования данных свойств указаны ниже.

Music->FileName = **"Resourse/Music.wav"**;  
Music->Volume = 50;  
Music->Play();

В FileName передаётся файл в виде строки, который будет проигрываться во время работы программы. Volume отвечает за громкость проигрывания музыки, принимая значения от “0” до “100”. Play начинает воспроизведение аудиозаписи.

**void \_\_fastcall** TMainF::CycleMusicTimer(TObject \*Sender)  
{  
 **if** (Music->CurrentTime == Music->Duration ) {  
 Music->CurrentTime = 0;  
 Music->Play();  
 }  
}

Данный пример показывает принцип зацикливания аудио в программном средстве. Реализуется данный процесс при помощи компонента TTimer, свойств компонента TMediaPlayer, таких как: CurrentTime, Duration. Duration показывает продолжительность аудио в секундах, а CurrentTime - текущую проигрываемую секунду песни.

В окне “Settings” пользователь имеет возможность отключить музыку.

**if** (MainF->Music->Volume != 0) {  
 MainF->Music->Volume = 0;  
 btnMute->Text = **"UNMUTE"**;  
} **else** {  
 MainF->Music->Volume = 50;  
 btnMute->Text = **"MUTE"**;  
}

Указанная выше процедура отвечает за приглушение музыки во время игры. Здесь используется свойство Volume, описанное ранее. Активируется данная часть посредством нажатия на кнопку “Mute”.

2.4 Игровая логика

Работу данной части программы обеспечивает модуль “Main”.

Для отрисовки игровых уровней было принято решение использовать матрицы, содержащие в себе 0 или 1, в зависимости от того, необходимо ли создавать блок в указанной точке. При их объявлении также задаётся постоянное начальное значение.

**int** Map[SizeY][SizeX];  
**const int** Map1[SizeY][SizeX];  
**const int** Map2[SizeY][SizeX];  
**const int** Map3[SizeY][SizeX];

Константные переменные SizeX, SizeY являются максимальным размером матрицы игрового поля.

Установка препятствий происходит каждый раз при начале игры, что исключает возможность появления лишних блоков, а также ускоряет работу приложения. Участок кода, отвечающий за данную функцию, прилагается ниже:

**int** count = 0;  
counter = 0;  
Blck = 0;  
**for** (**int** i = 0; i < SizeY - 1; i++) {  
 **for** (**int** j = 0; j < SizeX; j++) {  
 **if** (Map[i][j] == 1) {  
 TImage \*Block = new TImage(this);  
 Block->Parent = MainF;  
 Block->Size->Width = 50;  
 Block->Size->Height = 50;  
 Block->Position->X = 100 + j \* 50;  
 Block->Position->Y = 300 - (SizeY - 2 - i) \* 50;  
 Block->Bitmap->LoadFromFile(**"Resourse/block.jpg"**);  
 Blocks[count] = Block;  
 count++;  
 Blck++;  
 }  
 }  
}

Препятствия представляют собой компоненты TImage, которым задаётся размер 50х50 пикселей, что соответствует размеру одного блока. Позиция высчитывается исходя из расчёта их размещения в заготовленной матрице. 1 клетка матрицы – 50 пикселей на игровом экране. Ввиду того, что отсчёт координат начинается слева сверху экрана, позиция по Y задаётся иной формулой, представленной выше. Вместе с заданием позиций, устанавливается изображение препятствия в Bitmap компонентов TImage. Массив Blocks, который состоит из элементов типа TImage, необходим для последующего отслеживания позиций объектов во время игрового процесса, а именно проверки на столкновение и прыжок. Переменная count используется для последовательного заполнения массива Blocks. Переменная Blck необходима для подсчёта количества препятствий и используется в процедуре с основной логикой. Текстура блока показана на рисунке ниже:



Рисунок 2.4 – Текстура блока

Выбор уровня происходит в отдельном меню, появляющемся после нажатия кнопки “Play”. Кнопки “Left” и “Right” позволяют пользователю выбрать желаемую карту посредством клика на них или нажатия стрелок на клавиатуре. Нажатие кнопки “Go back” или клавиши “ESC” возвращает игрока в главное меню, сохраняя последнюю выбранную им карту. “Play” запускает игровой процесс, описанный в отдельной процедуре. Выбор уровня возможен благодаря использованию глобальной переменной mp. Часть кода, иллюстрирующая работу боковых кнопок, представлена ниже:

**void \_\_fastcall** TMainF::btnLeftClick(TObject \*Sender)  
{  
 **switch**(mp){  
 **case** 0:  
 ChangeMap(Mapa3);  
 mp = 2;  
 **break**;  
 **case** 1:  
 ChangeMap(Mapa1);  
 mp = 0;  
 **break**;  
 **case** 2:  
 ChangeMap(Mapa2);  
 mp = 1;  
 **break**;  
 }  
}

В зависимости от значения переменной mp, в процедуру ChangeMap передаётся параметр необходимой карты для установки. Процедура ChangeMap описывается далее:

**void** ChangeMap(Mapa Change)  
{  
 MainF->imgLevel->Bitmap->LoadFromFile(Change.bmpLvl);  
 MainF->imgBackgr->Width = Change.WidthMap;  
 MainF->imgBackgr->Bitmap->LoadFromFile(Change.bmpBack);  
 shft = Change.ShiftMap;  
 MainF->BackgrMove->Interval = Change.IntervalMap;  
 **for** (**int** i = 0; i < SizeY; i++) {  
 **for** (**int** j = 0; j < SizeX; j++) {  
 Map[i][j] = Change.MapNew[i][j];  
 }  
 }  
}

Здесь происходит изменение картинки предпросмотра уровня, картинки карты, её ширины, а также смещения, различного для каждой карты, которое позволяет сделать остановку фона в конце уровня и движение персонажа. Помимо этого, изменяется параметр Interval у таймера BackgrMove, отвечающего за всевозможные проверки, что ускоряет или замедляет прохождение карты. Также задаётся новое значение всех элементов основной матрицы для выбранной карты.

При попытке выхода из программы появляется диалог, в котором пользователь сможет отменить или подтвердить выход. Окно диалога указано на рисунке 2.5:

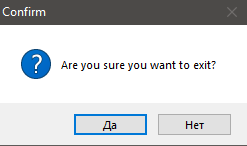


Рисунок 2.5 – Окно диалога выхода из программы

Блок-схема данного участка кода приведена ниже:



Рисунок 2.6 – Блок-схема выхода из программы

Отслеживание нажатий клавиш происходит в отдельной процедуре. Данная возможность была реализована при помощи структуры Switch, в которой рассматриваются отдельные случаи нажатия клавиш. По нажатию определённой клавиши на клавиатуре выполняется необходимая часть кода. Код процедуры указан далее:

**void \_\_fastcall** TMainF::FormKeyDown(TObject \*Sender, WORD &Key,

System::WideChar &KeyChar, TShiftState Shift)  
{  
 **if** ((KeyChar == VK\_SPACE) && !IsUp) {  
 IsUp = true;  
 }  
 **if** ((Key == VK\_LEFT) && (btnLeft->Visible == true)) {  
 btnLeftClick(Sender);  
 }  
 **if** ((Key == VK\_RIGHT) && (btnRight->Visible == true)) {  
 btnRightClick(Sender);  
 }  
 **if** ((Key == VK\_ESCAPE) && (btnReturn->Visible == true)) {  
 btnReturnClick(Sender);  
 }  
 **if** ((Key == VK\_RETURN) && (btnGo->Visible == true)) {  
 btnGoClick(Sender);  
 }  
}

При запуске программы происходит инициализациях переменных и свойств различных компонентов, установка начальной позиции игрока, загрузка картинок элементов, запуск музыкального сопровождения. Блок-схема запуска программы представлена на рисунке 2.7.



Рисунок 2.7 – Блок-схема запуска программы

Основная работа программы построена на компоненте TTimer, и описывается в свойстве OnTimer. Указанный компонент включается при выбранной карте и нажатии кнопки “Play”, а отключается при возврате в меню выбора карты или закрытии программы.

В начале процедуры написана отрисовка уровня по мере его прохождения игроком.

counter += 1;  
**if** (imgBackgr->Position->X > shft) {  
 imgBackgr->Position->X -= 50;  
 **for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Position->X -= 50;  
 }  
} **else** {  
 imgPlayer->Position->X += 50;  
 counter -= 1;  
}

Переменная counter подсчитывает количество ходов, совершённых во время прохождения карты. Она необходима для возвращения препятствий в исходное положение при повторном прохождении пользователем уровня. После этого, программа решает, сдвигать задний фон и блоки, или сдвигать самого персонажа. Движение персонажа по оси Х происходит в конце уровня, когда все препятствия помещаются в одно окно. Это позволяет реализовать продвижение карты в зависимости от пройденного игроком расстояния.

Далее описана проверка на столкновение игрока с препятствием.

**if** (((Map[pos.Y][pos.X + 1] == 1) && !IsUp) ||  
 ((Map[pos.Y + 1][pos.X + 1] == 1) && (Map[pos.Y + 1][pos.X] == 0)))

В ней рассматриваются два варианта: если пользователь не нажал кнопку прыжка и на следующем ходу он окажется в блоке, если при падении персонаж оказывается в блоке. При срабатывании данной части кода игра останавливается, а игрок исчезает с игрового поля и перед ним ставится выбор, повторить попытку или выйти в меню выбора уровня. Окно выбора показано на рисунке 2.8:

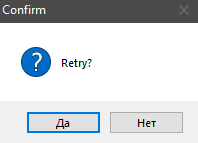


Рисунок 2.8 – Окно выбора

В случае отрицательного ответа, пользователь возвращается в меню выбора карты, скрываются препятствия и другие объекты.

btnPlayClick(Sender);  
imgBackgr->Visible = false;  
**for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Visible = false;  
}  
counter = 0;

Если игрок хочет совершить ещё одну попытку, срабатывает участок кода, приведённый ниже:

IsUp = false;  
pos.X = -1;  
pos.Y = SizeY - 2;  
imgPlayer->Position->Y = 300;  
imgPlayer->Position->X = 100;  
imgPlayer->RotationAngle = 0;  
imgBackgr->Position->X = 0;  
imgPlayer->Visible = true;  
**for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Position->X += counter \* 50;  
}  
counter = 0;  
BackgrMove->Enabled = true;

При его работе задаются следующие параметры: переменная IsUp, отвечающая за прыжок, переменная pos, являющаяся координатами игрока в матрице карты, позиция модели игрока, угол поворота модели, который изменяется при прыжке и падении, позиция заднего фона уровня. Также устанавливаются начальные позиции препятствий, используя вышеупомянутую переменную counter, после ей присваивается нулевое значение и включается таймер BackgrMove.

Далее программа проверяет, выйграл ли пользователь, используя следующий участок кода:

**if** (Map[pos.Y][pos.X + 1] == 3) {  
 BackgrMove->Enabled = false;  
 ShowMessage(**"!!!... YOU WON ^-^ ...!!!"**);  
 btnPlayClick(Sender);  
 imgBackgr->Visible = false;  
 imgPlayer->Visible = false;  
 **for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Visible = false;  
 }  
 counter = 0;  
}

Происходит проверка, находится ли в следующей клетке матрицы значение, необходимое для выигрыша. При успехе останавливается таймер игры, на экран выводится сообщение о том, что пользователь прошёл уровень, происходит возвращение в меню выбора карты. Сообщение о выигрыше показано на рисунке 2.9:

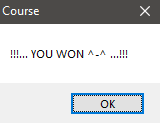


Рисунок 2.9 – Сообщение о выигрыше

Ниже находится код, отвечающий за прыжки персонажа, совершаемые по нажатию клавиши “Space” на клавиатуре, и падение после прыжка.

**if** (IsUp && (Map[pos.Y - 1][pos.X + 1] == 0) &&

(Map[pos.Y + 1][pos.X] == 1)) {  
 imgPlayer->Position->Y -= 50;  
 imgPlayer->RotationAngle += 90;  
 pos.Y -= 1;  
 IsUp = false;  
} **else** {  
 **if** ((pos.Y < (SizeY - 2)) && (Map[pos.Y + 1][pos.X + 1] == 0)) {  
 imgPlayer->Position->Y += 50;  
 imgPlayer->RotationAngle += 90;  
 pos.Y += 1;  
 }  
}

Если игрок нажал клавишу прыжка и в следующей сверху клетке не находится препятствие, а также под игроком есть блок, происходит смещение персонажа по оси Y вверх на один блок, вращение персонажа относительно своей оси на девяноста градусов, изменение переменной pos.Y на единицу, придание переменной IsUp значения false. При невыполнении данного условия совершается проверка на падение, а именно: если игрок находится выше начального уровня и следующая снизу клетка не является препятствием. Когда проверка истинна, персонаж смещается на один блок вниз, вращается на девяноста градусов и к переменной pos.Y прибавляется единица.

После всех проверок происходит увеличение позиции игрока относительно оси Х на единицу.

pos.X += 1;

2.5 Внешний вид

Для возможности кастомизации игрового процесса в приложении предусмотрен выбор модели персонажа. Данная часть программы реализована в модуле “Settings”. Вид всплывающего окна прилагается далее:

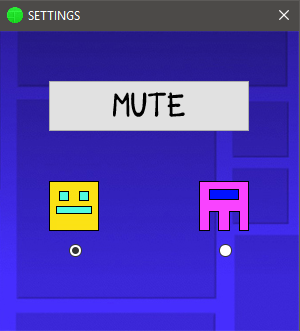


Рисунок 2.10 – Всплывающее окно Settings

При запуске приложения автоматически выбирается первый вариант. Пользователь может изменить модель посредством нажатия на компонент TRadioButton. При этом в программе изменится модель игрока и заблокируется выбранная кнопка. Процедура изменения модели указана ниже:

**void \_\_fastcall** TSettingsF::btnPlayer2Click(TObject \*Sender)  
{  
 MainF->imgPlayer->Bitmap->LoadFromFile(**"Resourse/player2.png"**);  
}

3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения были выявлены некоторые недостатки интерфейса и некорректная работа основной логики программного средства. Была составлена таблица 3.1, показывающая ожидаемые и реальные результаты, полученные при заданных условиях. Таблица прилагается далее:

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Нажатие кнопки btnInfo | Всплывающее окно с информацией | Всплывающее окно с информацией |
| 2. | Нажатие кнопки btnSettings | Всплывающее окно с настройками | Всплывающее окно с настройками |
| 3. | Зацикливание музыкального сопровождения | Повторение песни по её окончании | Остановка музыкального сопровождения |
| 4. | Нажатие кнопки btnPlay | Отображение и скрытие элементов меню | Отображение и скрытие элементов меню |
| 5. | Нажатие кнопок переключения карты | Изменение превью карты | Изменение превью карты |
| 6. | Нажатие кнопки btnReturn | Возвращение в главное меню | Возвращение в главное меню |
| 7. | Нажатие кнопки btnGo | Запуск игрового процесса | Запуск игрового процесса |
| 8. | Нажатие кнопки Space | Прыжок персонажа | Прыжок персонажа |
| 9. | Нажатие кнопки Space во время прыжка | Отсутствие изменений | Прыжок персонажа |
| 10. | Столкновение персонажа с препятствием | Вывод на экран предложения о повторе | Вывод на экран предложения о повторе |
| 11. | Нажатие кнопки Mute | Выключение музыкального сопровождения | Выключение музыкального сопровождения |
| 12. | Закрытие приложения | Вывод на экран сообщения о закрытии | Вывод на экран сообщения о закрытии |
| 13. | Нажатие кнопки под картинкой персонажа | Смена изображения персонажа | Смена изображения персонажа |
| 14. | Падение персонажа с препятствия высотой в 1 блок | Спуск на 1 блок | Спуск на 1 блок |
| 15. | Прохождение уровня до конца | Вывод сообщения о выигрыше | Вывод сообщения о выигрыше |
| 16. | Падение персонажа на нулевой уровень | Персонаж прекращает движение по оси Y | Персонаж продолжает движение по оси Y |
| 17. | Выход из приложения во время игры | Остановка игры и вывод сообщения на экран | Остановка игры и вывод сообщения на экран |
| 18. | Нажатие клавиши ESC в главном меню | Отсутствие изменений | Вывод сообщения о закрытии программы |
| 19. | Нажатие клавиши Enter в главном меню | Отсутствие изменений | Отсутствие изменений |
| 20. | Нажатие клавиши Enter в меню выбора карты | Запуск игрового процесса | Запуск игрового процесса |
| 21. | Нажатие клавиши ESC в меню выбора карты | Выход в главное меню | Выход в главное меню |

Была обнаружена проблема с зацикливанием музыкального сопровождения в приложении. Проигрываемый трек должен быть повторяться после его окончания, однако этого не происходило. В связи с этим в программу был добавлен компонент TTimer, который содержит в себе следующий код:

**void \_\_fastcall** TMainF::CycleMusicTimer(TObject \*Sender)  
{  
 **if** (Music->CurrentTime == Music->Duration ) {  
 Music->CurrentTime = 0;  
 Music->Play();  
 }  
}

Следующей проблемой стала возможность прыжка во время полёта. Это позволяло проходить уровень невзирая на препятствия. Было решено изменить проверку на прыжок в процедуре BackgrMove. Изменённый участок кода находится ниже:

**if** (IsUp && (Map[pos.Y - 1][pos.X + 1] == 0) &&

(Map[pos.Y + 1][pos.X] == 1))

При спуске персонажа на начальный уровень была замечена ошибка, в результате которой модель продолжала движение вниз по уровню. В связи с этим было принято решение добавить в матрицу строку, заполненную единицами и изменить код проверки на падение. Исправления в коде далее:

**if** ((pos.Y < (SizeY - 2)) && (Map[pos.Y + 1][pos.X + 1] == 0))

При написании программы была написана неправильная проверка действия для нажатия клавиши “ESC”. По задумке, в меню выбора карты при нажатии игрок должен был возвращаться в главное меню, однако при нажатии в главном меню происходило закрытие программы. Изменённый код прилагается ниже:

**if** ((Key == VK\_ESCAPE) && (btnReturn->Visible == true)) {  
 btnReturnClick(Sender);

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Правила игры

Игра заключается в том, что игроку необходимо пройти все уровни, предложенные разработчиком, однако каждый следующий уровень сложнее предыдущего, что требует от пользователя всё больших усилий и времени, необходимого для успешного завершения карты.

Геймер управляет кубом, изображение которого может быть изменено в самой игре в меню “Settings”. Ниже указан пример одной из моделей:



Рисунок 4.1 – Одна из моделей куба

Прыжок во время игры осуществляется по нажатии клавиши “Space” на клавиатуре.

При выборе карты пользователь может использовать стрелочки на клавиатуре для изменения уровня, клавишу “ESC” для выхода в главное меню, клавишу “Enter” для начала игры.

При соприкосновении игрока и препятствия на экран выводится предложение о повторении попытки прохождения уровня. В случае отказа пользователь возвращается в меню выбора карты, где сможет выбрать другую и попробовать данный уровень позже.

По окончании прохождения уровня на экран выведется сообщение о выигрыше, а после пользователь вернётся в меню выбора карты, где сможет продолжить игру на другом уровне.

Имеется возможность отключить музыку во время игры путём нажатия кнопки “Mute” в меню “Settings”. При повторном нажатии воспроизведение песни возобновится.

4.2 Интерфейс программы

Главное окно программы, изображённое на рисунке 4.3, содержит следующие элементы:

– “Play”;

– “Settings”;

– “Info”.

Для начала игры следует ознакомиться с её правилами, находящимися в отдельном окне, попасть на которое можно посредством нажатия кнопки “Info”. Внешний вид окна с правилами представлен на рисунке 4.2.

После этого необходимо нажать кнопку “Play”. Игрок попадает в меню выбора карты, в котором определяет уровень, который хочет выбрать для прохождения. Нажатием клавиши “Enter” на клавиатуре или кнопки “Go” на экране запускается игровой процесс, в котором пользователю, нажимая клавишу “Space”, необходимо преодолевать препятствия и дойти до конца карты.

В главном меню программного средства игрок может изменить модель персонажа, за которого играет. Происходит это по открытию соответствующего пункта меню “Settings”. Предоставляется два варианта внешнего вида, что позволяет пользователю разнообразить игровой процесс.

В программном средстве присутствует музыкальное и визуальное сопровождение. Проверка и загрузка ресурсов происходит в момент запуска приложения. При этом поиск файлов осуществляется непосредственно в директории программы.

Скриншот игрового процесса представлен на рисунке 4.4.

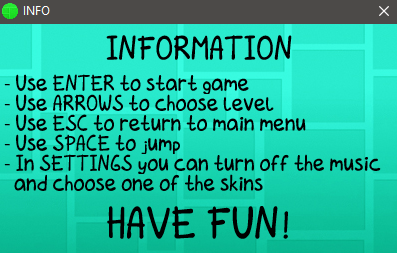
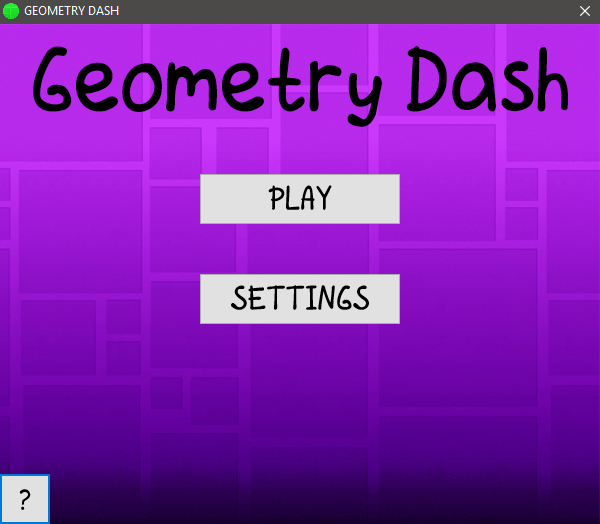
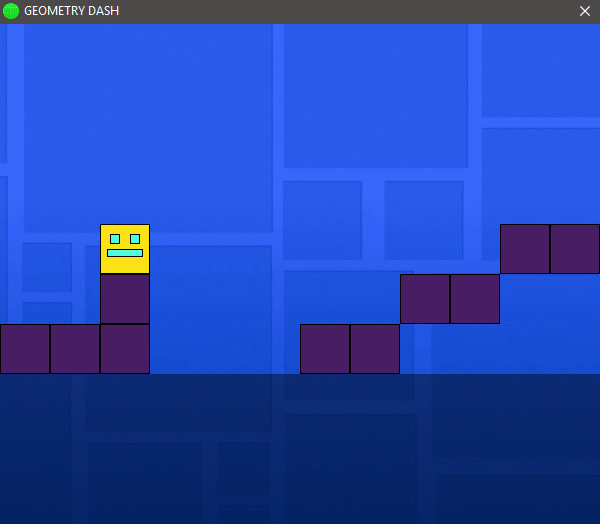


Рисунок 4.2 – Внешний вид окна с правилами



4.3 – Главное окно программы



4.4 – Скриншот игрового процесса

ЗАКЛЮЧЕНИЕ

Видеоигры стали значительной частью жизни множества людей. Большинство игр никак не влияет на развитие человека, однако существуют проекты, благодаря которым пользователи развивают логическое мышление, улучшают реакцию, учатся думать наперёд. Это помогает игрокам в их повседневной жизни, помимо виртуальной.

В рамках данного курсового проекта было разработано программное средство “Geometry Dash”, которое поможет пользователю скрасить время, проведённое за компьютером. Согласно поставленным задачам, в данном приложении были реализованы следующие возможности:

– выбор игрового уровня;

– изменение внешнего вида игрока;

– отображение пройденных уровней;

– возможность отключения музыки;

– приостановка игры.

Для выполнения всех поставленных задач потребовалось тщательно изучить возможности языка C++, освоить взаимодействие между компонентами внутри него, научиться основам объектно-ориентированного программирования.

Существует большое количество возможных улучшений приложения. Одним из них является создание новых уровней, возможности добавления новых типов блоков, добавление новых режимов игры. Другим вариантом развития является адаптация проекта для запуска на устройствах с низкой разрешающей способностью экрана и переход к кроссплатформенной разработке.

Использование данного программного средства позволит не только провести время с удовольствием, но также и развить свою реакцию, логическое мышление и способность просчитывать действия на несколько шагов вперед.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке C++ / Гирберд Шилдт. Учебный курс: / Гирберд Шилдт. – СПб: изд. С. В. Малгачёва, 2001. – 231 с;
2. RAD Studio Product Documentation – Embarcadero Technologies [Электронный ресурс]. – Электронные данные. – Режим доступа: http://docs.embarcadero.com/products/rad\_studio;
3. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003;
4. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
5. Культин, Н. А. Основы программирования в C++ 2-е издание – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 182с;
6. Культин, Н. А. Основы программирования в C++ – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 167с;
7. Хлебостроев, В. С. Программирование графики в Rad Studio – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 243с;
8. Графические возможности Rad Studio (электронный ресурс). – Электронные данные. – Режим доступа: http://embarcadero.gym5cheb.ru.

ПРИЛОЖЕНИЕ А

Исходный код программы

**Main.h**

#ifndef **MainH**#define **MainH***//---------------------------------------------------------------------------*#include **<System.Classes.hpp>**#include **<FMX.Controls.hpp>**#include **<FMX.Forms.hpp>**#include **<FMX.Controls.Presentation.hpp>**#include **<FMX.Menus.hpp>**#include **<FMX.StdCtrls.hpp>**#include **<FMX.Types.hpp>**#include **<FMX.Dialogs.hpp>**#include **<FMX.Objects.hpp>**#include **<FMX.Ani.hpp>**#include **<FMX.Media.hpp>***//---------------------------------------------------------------------------*class TMainF : public TForm  
{  
\_\_published: *// IDE-managed Components*

TButton \*btnPlay;  
 TPopupMenu \*BlankPop;  
 TButton \*btnSettings;  
 TButton \*btnRight;  
 TButton \*btnLeft;  
 TImage \*imgLevel;  
 TButton \*btnReturn;  
 TButton \*btnGo;  
 TTimer \*BackgrMove;  
 TImage \*imgBackgr;  
 TImage \*imgPlayer;  
 TImage \*imgMenu;  
 TMediaPlayer \*Music;  
 TTimer \*CycleMusic;  
 TButton \*btnInfo;  
 **void \_\_fastcall** FormCloseQuery(TObject \*Sender, bool &CanClose);  
 **void \_\_fastcall** btnPlayClick(TObject \*Sender);  
 **void \_\_fastcall** FormCreate(TObject \*Sender);  
 **void \_\_fastcall** btnReturnClick(TObject \*Sender);  
 **void \_\_fastcall** btnGoClick(TObject \*Sender);  
 **void \_\_fastcall** BackgrMoveTimer(TObject \*Sender);  
 **void \_\_fastcall** FormKeyDown(TObject \*Sender, WORD &Key, System::WideChar

&KeyChar, TShiftState Shift);  
 **void \_\_fastcall** btnRightClick(TObject \*Sender);  
 **void \_\_fastcall** btnLeftClick(TObject \*Sender);  
 **void \_\_fastcall** btnSettingsClick(TObject \*Sender);  
 **void \_\_fastcall** CycleMusicTimer(TObject \*Sender);  
 **void \_\_fastcall** btnInfoClick(TObject \*Sender);  
 private: *// User declarations* public: *// User declarations* **\_\_fastcall** TMainF(TComponent\* Owner);

};  
*//---------------------------------------------------------------------------***extern** PACKAGE TMainF \*MainF;  
*//---------------------------------------------------------------------------*#endif

**Main.cpp**

#include **<fmx.h>**#pragma hdrstop  
  
#include **"Main.h"**#include **"Settings.h"**#include **"Info.h"***//---------------------------------------------------------------------------*#pragma package(smart\_init)  
#pragma resource **"\*.fmx"**TMainF \*MainF;  
*//---------------------------------------------------------------------------*TImage \*Blocks[50];  
**int** counter, Blck, shft, mp;  
bool IsUp;  
**struct** position  
{  
 **int** X;  
 **int** Y;  
};  
position pos;  
**const int** SizeY = 6, SizeX = 30;  
**int** Map[SizeY][SizeX];  
**const int** Map1[SizeY][SizeX] =  
 {  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0},  
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0}  
 };  
**const int** Map2[SizeY][SizeX] =  
 {  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 3,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 3,

0, 0, 0, 0, 0, 0, 0},  
 {0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 3,

0, 0, 0, 0, 0, 0, 0},  
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

0, 0, 0, 0, 0, 0, 0}  
 };  
**const int** Map3[SizeY][SizeX] =  
 {  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 3},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 1, 1, 3},  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 1, 0, 0, 3},  
 {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,

0, 0, 1, 0, 0, 0, 3},  
 {0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,

0, 1, 0, 0, 0, 0, 3},  
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1}  
 };  
**struct** Mapa  
{  
 **char** bmpLvl[17];  
 **int** WidthMap;  
 **char** bmpBack[17];  
 **int** ShiftMap;  
 **int** IntervalMap;  
 **int** MapNew[SizeY][SizeX];  
};  
Mapa Mapa1 = { **"Resourse/lvl1.jpg"**, 700, **"Resourse/map1.jpg"**, -50, 700},  
 Mapa2 = { **"Resourse/lvl2.jpg"**, 1250, **"Resourse/map2.jpg"**, -650, 500},  
 Mapa3 = { **"Resourse/lvl3.jpg"**, 1600, **"Resourse/map3.jpg"**, -1000, 370};  
*//---------------------------------------------------------------------------***\_\_fastcall** TMainF::TMainF(TComponent\* Owner)  
: TForm(Owner)  
 {  
 }  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::FormCloseQuery(TObject \*Sender, bool &CanClose)  
{  
 BackgrMove->Enabled = false;  
 **if** (MessageDlg(**"Are you sure you want to exit?"**, TMsgDlgType::mtConfirmation, mbYesNo, 0) == mrNo) {  
 CanClose = false;  
 }  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnPlayClick(TObject \*Sender)  
{  
 btnPlay->Visible = false;  
 btnSettings->Visible = false;  
 btnLeft->Visible = true;  
 btnRight->Visible = true;  
 imgLevel->Visible = true;  
 btnReturn->Visible = true;  
 btnGo->Visible = true;  
 imgMenu->Visible = false;  
 btnInfo->Visible = false;  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::FormCreate(TObject \*Sender)  
{  
 mp = 0;  
 shft = -50;  
 **for** (**int** i = 0; i < SizeY; i++) {  
 **for** (**int** j = 0; j < SizeX; j++) {  
 Map[i][j] = Map1[i][j];  
 Mapa1.MapNew[i][j] = Map1[i][j];  
 Mapa2.MapNew[i][j] = Map2[i][j];  
 Mapa3.MapNew[i][j] = Map3[i][j];  
 }  
 }  
 Music->Volume = 50;  
 Music->Play();  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnReturnClick(TObject \*Sender)  
{  
 btnPlay->Visible = true;  
 btnSettings->Visible = true;  
 btnLeft->Visible = false;  
 btnRight->Visible = false;  
 imgLevel->Visible = false;  
 btnReturn->Visible = false;  
 btnGo->Visible = false;  
 imgMenu->Visible = true;  
 btnInfo->Visible = true;  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnGoClick(TObject \*Sender)  
{  
 btnLeft->Visible = false;  
 btnRight->Visible = false;  
 btnReturn->Visible = false;  
 btnGo->Visible = false;  
 imgLevel->Visible = false;  
 imgBackgr->Visible = true;  
 imgPlayer->Visible = true;  
  
 IsUp = false;  
 imgBackgr->Position->X = 0;  
 pos.X = 0;  
 pos.Y = SizeY - 2;  
 imgPlayer->Position->Y = 300;  
 imgPlayer->Position->X = 100;  
 imgPlayer->RotationAngle = 0;  
  
 **int** count = 0;  
 counter = 0;  
 Blck = 0;  
 **for** (**int** i = 0; i < SizeY - 1; i++) {  
 **for** (**int** j = 0; j < SizeX; j++) {  
 **if** (Map[i][j] == 1) {  
 TImage \*Block = new TImage(this);  
 Block->Parent = MainF;  
 Block->Size->Width = 50;  
 Block->Size->Height = 50;  
 Block->Position->X = 100 + j \* 50;  
 Block->Position->Y = 300 - (SizeY - 2 - i) \* 50;  
 Block->Bitmap->LoadFromFile(**"Resourse/block.jpg"**);  
 Blocks[count] = Block;  
 count++;  
 Blck++;  
 }  
 }  
 }  
 BackgrMove->Enabled = true;  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::BackgrMoveTimer(TObject \*Sender)  
{  
 counter += 1;  
 **if** (imgBackgr->Position->X > shft) {  
 imgBackgr->Position->X -= 50;  
 **for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Position->X -= 50;  
 }  
 } **else** {  
 imgPlayer->Position->X += 50;  
 counter -= 1;  
 }  
  
 **if** (((Map[pos.Y][pos.X + 1] == 1) && !IsUp) ||  
 ((Map[pos.Y + 1][pos.X + 1] == 1) && (Map[pos.Y + 1][pos.X] == 0))) {  
 BackgrMove->Enabled = false;  
 imgPlayer->Visible = false;  
 **if** (MessageDlg(**"Retry?"**, TMsgDlgType::mtConfirmation, mbYesNo, 0) == mrNo) {  
 btnPlayClick(Sender);  
 imgBackgr->Visible = false;  
 **for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Visible = false;  
 }  
 counter = 0;  
 } **else** {  
 IsUp = false;  
 pos.X = -1;  
 pos.Y = SizeY - 2;  
 imgPlayer->Position->Y = 300;  
 imgPlayer->Position->X = 100;  
 imgPlayer->RotationAngle = 0;  
 imgBackgr->Position->X = 0;  
 imgPlayer->Visible = true;  
 **for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Position->X += counter \* 50;  
 }  
 counter = 0;  
 BackgrMove->Enabled = true;  
 }  
 }  
  
 **if** (Map[pos.Y][pos.X + 1] == 3) {  
 BackgrMove->Enabled = false;  
 ShowMessage(**"!!!... YOU WON ^-^ ...!!!"**);  
 btnPlayClick(Sender);  
 imgBackgr->Visible = false;  
 imgPlayer->Visible = false;  
 **for** (**int** i = 0; i < Blck; i++) {  
 Blocks[i]->Visible = false;  
 }  
 counter = 0;  
 }  
  
 **if** (IsUp && (Map[pos.Y - 1][pos.X + 1] == 0) && (Map[pos.Y + 1][pos.X] == 1)) {  
 imgPlayer->Position->Y -= 50;  
 imgPlayer->RotationAngle += 90;  
 pos.Y -= 1;  
 IsUp = false;  
 } **else** {  
 **if** ((pos.Y < (SizeY - 2)) && (Map[pos.Y + 1][pos.X + 1] == 0)) {  
 imgPlayer->Position->Y += 50;  
 imgPlayer->RotationAngle += 90;  
 pos.Y += 1;  
 }  
 }  
  
 pos.X += 1;  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::FormKeyDown(TObject \*Sender, WORD &Key, System::WideChar &KeyChar,  
TShiftState Shift)  
{  
 **if** ((KeyChar == VK\_SPACE) && !IsUp) {  
 IsUp = true;  
 }  
 **if** ((Key == VK\_LEFT) && (btnLeft->Visible == true)) {  
 btnLeftClick(Sender);  
 }  
 **if** ((Key == VK\_RIGHT) && (btnRight->Visible == true)) {  
 btnRightClick(Sender);  
 }  
 **if** ((Key == VK\_ESCAPE) && (btnReturn->Visible == true)) {  
 btnReturnClick(Sender);  
 }  
 **if** ((Key == VK\_RETURN) && (btnGo->Visible == true)) {  
 btnGoClick(Sender);  
 }  
}  
*//---------------------------------------------------------------------------***void** ChangeMap(Mapa Change)  
{  
 MainF->imgLevel->Bitmap->LoadFromFile(Change.bmpLvl);  
 MainF->imgBackgr->Width = Change.WidthMap;  
 MainF->imgBackgr->Bitmap->LoadFromFile(Change.bmpBack);  
 shft = Change.ShiftMap;  
 MainF->BackgrMove->Interval = Change.IntervalMap;  
 **for** (**int** i = 0; i < SizeY; i++) {  
 **for** (**int** j = 0; j < SizeX; j++) {  
 Map[i][j] = Change.MapNew[i][j];  
 }  
 }  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnRightClick(TObject \*Sender)  
{  
 **switch**(mp){  
 **case** 0:  
 ChangeMap(Mapa2);  
 mp = 1;  
 **break**;  
 **case** 1:  
 ChangeMap(Mapa3);  
 mp = 2;  
 **break**;  
 **case** 2:  
 ChangeMap(Mapa1);  
 mp = 0;  
 **break**;  
 }  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnLeftClick(TObject \*Sender)  
{  
 **switch**(mp){  
 **case** 0:  
 ChangeMap(Mapa3);  
 mp = 2;  
 **break**;  
 **case** 1:  
 ChangeMap(Mapa1);  
 mp = 0;  
 **break**;  
 **case** 2:  
 ChangeMap(Mapa2);  
 mp = 1;  
 **break**;  
 }  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnSettingsClick(TObject \*Sender)  
{  
 SettingsF->ShowModal();  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::CycleMusicTimer(TObject \*Sender)  
{  
 **if** (Music->CurrentTime == Music->Duration ) {  
 Music->CurrentTime = 0;  
 Music->Play();  
 }  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TMainF::btnInfoClick(TObject \*Sender)  
{  
 btnPlay->Enabled = true;  
 InfoF->ShowModal();  
}

**Settings.h**

#ifndef **SettingsH**#define **SettingsH***//---------------------------------------------------------------------------*#include **<System.Classes.hpp>**#include **<FMX.Controls.hpp>**#include **<FMX.Forms.hpp>**#include **<FMX.Controls.Presentation.hpp>**#include **<FMX.StdCtrls.hpp>**#include **<FMX.Types.hpp>**#include **<FMX.Objects.hpp>***//---------------------------------------------------------------------------*class TSettingsF : public TForm  
{  
\_\_published: *// IDE-managed Components* TButton \*btnMute;  
 TImage \*imgSettingsBack;  
 TImage \*imgPlayer1;  
 TImage \*imgPlayer2;  
 TRadioButton \*btnPlayer1;  
 TRadioButton \*btnPlayer2;  
 **void \_\_fastcall** btnMuteClick(TObject \*Sender);  
 **void \_\_fastcall** btnPlayer2Click(TObject \*Sender);  
 **void \_\_fastcall** btnPlayer1Click(TObject \*Sender);  
 private: *// User declarations* public: *// User declarations* **\_\_fastcall** TSettingsF(TComponent\* Owner);  
};  
*//---------------------------------------------------------------------------***extern** PACKAGE TSettingsF \*SettingsF;  
*//---------------------------------------------------------------------------*#endif

**Settings.cpp**

#include **<fmx.h>**#pragma hdrstop  
  
#include **"Settings.h"**#include **"Main.h"***//---------------------------------------------------------------------------*#pragma package(smart\_init)  
#pragma resource **"\*.fmx"**TSettingsF \*SettingsF;  
*//---------------------------------------------------------------------------***\_\_fastcall** TSettingsF::TSettingsF(TComponent\* Owner)  
: TForm(Owner)  
 {  
 }  
*//---------------------------------------------------------------------------***void \_\_fastcall** TSettingsF::btnMuteClick(TObject \*Sender)  
{  
 **if** (MainF->Music->Volume != 0) {  
 MainF->Music->Volume = 0;  
 btnMute->Text = **"UNMUTE"**;  
 } **else** {  
 MainF->Music->Volume = 50;  
 btnMute->Text = **"MUTE"**;  
 }  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TSettingsF::btnPlayer2Click(TObject \*Sender)  
{  
 MainF->imgPlayer->Bitmap->LoadFromFile(**"Resourse/player2.png"**);  
}  
*//---------------------------------------------------------------------------***void \_\_fastcall** TSettingsF::btnPlayer1Click(TObject \*Sender)  
{  
 MainF->imgPlayer->Bitmap->LoadFromFile(**"Resourse/player1.jpg"**);  
}

**Info.h**

#ifndef **InfoH**#define **InfoH***//---------------------------------------------------------------------------*#include **<System.Classes.hpp>**#include **<FMX.Controls.hpp>**#include **<FMX.Forms.hpp>**#include **<FMX.Controls.Presentation.hpp>**#include **<FMX.StdCtrls.hpp>**#include **<FMX.Types.hpp>**#include **<FMX.Objects.hpp>***//---------------------------------------------------------------------------*class TInfoF : public TForm  
{  
\_\_published: *// IDE-managed Components* TLabel \*lInfo;  
 TLabel \*lLevel;  
 TLabel \*lJump;  
 TLabel \*lStart;  
 TLabel \*lReturn;  
 TLabel \*lMusic;  
 TLabel \*lSkins;  
 TLabel \*lFun;  
 TImage \*imgBack;  
 private: *// User declarations* public: *// User declarations* **\_\_fastcall** TInfoF(TComponent\* Owner);  
};  
*//---------------------------------------------------------------------------***extern** PACKAGE TInfoF \*InfoF;  
*//---------------------------------------------------------------------------*#endif

**Info.cpp**

#include **<fmx.h>**#pragma hdrstop  
  
#include **"Info.h"***//---------------------------------------------------------------------------*#pragma package(smart\_init)  
#pragma resource **"\*.fmx"**TInfoF \*InfoF;  
*//---------------------------------------------------------------------------***\_\_fastcall** TInfoF::TInfoF(TComponent\* Owner)  
: TForm(Owner)  
 {  
 }