Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**«SPACE SHOOTER»**

БГУИР КП 1-40 01 01 15 ПЗ

|  |  |  |
| --- | --- | --- |
|  |  | Студент: гр. 751001 Крукович П.Н. |
|  |  | Руководитель: асс. Шостак Е.В. |

Минск 2018

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc532382961)

[1 Анализ предметной области 5](#_Toc532382962)

[1.1 Обзор аналогов 5](#_Toc532382963)

[1.2 Постановка задачи 8](#_Toc532382964)

[2 Разработка программного средства 9](#_Toc532382965)

[2.1 Структура программного средства 9](#_Toc532382966)

[2.2 Интерфейс программного средства 13](#_Toc532382996)

[2.3 Игровая логика 19](#_Toc532383001)

[3 Тестирование программного средства 22](#_Toc532383002)

[4 Руководство пользователя 24](#_Toc532383003)

[ЗАКЛЮЧЕНИЕ 26](#_Toc532383004)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc532383005)

[Приложение А. Исходный код программы 28](#_Toc532383006)

ВВЕДЕНИЕ

Язык ассемблера — это машинно-ориентированный язык программирования низкого уровня. Ассемблер — транслятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке. В отличие от языков высокого уровня абстракции, в которых многие проблемы реализации алгоритмов скрыты от разработчиков, язык ассемблера тесно связан с системой команд микропроцессора, обеспечивает доступ к регистрам, указание методов адресации и описание операций в терминах команд процессора.

Оптимальной можно считать программу, которая работает правильно, по возможности быстро и занимает, возможно, малый объем памяти. Кроме того, ее легко читать и понимать; ее легко изменить; ее создание требует мало времени и незначительных расходов. В идеале язык ассемблера должен обладать совокупностью характеристик, которые бы позволяли получать программы, удовлетворяющие как можно большему числу перечисленных качеств.

На языке ассемблера пишут программы или их фрагменты в тех случаях, когда критически важны:

* объем используемой памяти (программы-загрузчики, встраиваемое программное обеспечение, программы для микроконтроллеров и процессоров с ограниченными ресурсами, вирусы);
* быстродействие.

Программы, написанные на языке ассемблера, выполняются гораздо быстрее, чем программы-аналоги, написанные на языках программирования высокого уровня абстракции. В данном случае быстродействие зависит от понимания того, как работает конкретная модель процессора, реальный конвейер на процессоре, размер кэша, тонкостей работы операционной системы, в результате, программа начинает работать быстрее, но теряет переносимость и универсальность.

Знание языка ассемблера облегчает понимание архитектуры компьютера и работы его аппаратной части, то, чего не может дать знание языков высокого уровня абстракции. Вот почему я выбрала язык ассемблера в качестве основного инструмента для разработки курсового проекта.

В настоящее время большинство программистов работает со средами быстрого проектирования, когда все необходимые элементы оформления и управления создаются с помощью готовых визуальных компонентов. Это существенно упрощает процесс программирования. Однако, нередко приходится сталкиваться с такими ситуациями, когда наиболее мощное и эффективное функционирование отдельных программных модулей возможно только в случае написания их на языке ассемблера (ассемблерные вставки). В частности, в любой программе, связанной с выполнением многократно повторяющихся циклических процедур, будь это циклы математических вычислений или вывод графических изображений. Все пакеты современных языков программирования допускают ассемблерные вставки, а результатом всегда является существенное повышение быстродействия программ.

Языки программирования высокого уровня абстракции разрабатывались с целью улучшения читаемости кода, а также, чтобы не учитывать в программах специфические технические особенности отдельных компьютеров. Язык ассемблера разрабатывается с учетом специфики процессора, поэтому для грамотного написания программы требуется, в общем, знать архитектуру процессора используемого компьютера.

Любое оконное приложение должно использовать функции API (по крайней мере, для успешного завершения). Windows API – это набор базовых функций для программирования приложений операционных систем Windows, который предоставляет прямой способ взаимодействия прикладных программ с ОС. Эти функции находятся в диамических библиотеках, тремя исторически первыми являются kernel32.dll (управление памятью, операции ввода-вывода, создание процессов и потоков, функции синхронизации), user32.dll (управление окнами и пользовательским интерфейсом), gdi32.dll (создание графических объектов и передача их на устройства отображения) и других.

WinAPI является основой для создания всех приложений для Windows, поэтому именно эта технология была выбрана мной для изучения.

Так как главной целью проекта является углубленное изучение низкоуровневого программирования и функций Windows API, в качестве темы своей курсовой работы я выбрала игровое приложение в жанре аркады, космический шутер. Подобные игры были очень популярны до 1980-х годов, так как основой игровой логики является конечный автомат, который в то время мог быть легко реализован как для приставок, так и для ПК.

1. анализ предметной области
   1. Обзор аналогов

Игровые приложения космической тематики издавались ещё в начале далеких 1980-ых годов. Рассмотрим некоторые из них.

«Space Invaders» — видеоигра, разработанная Томохиро Нисикадо и выпущенная в 1978 году на аркадных автоматах. Несмотря на то, что по сегодняшним стандартам игра довольно проста, она является основоположницей современных игр и помогла игровой индустрии превратиться в одну из крупнейших в мире. Space Invaders является лучшей аркадной игрой по версии Книги рекордов Гиннесса.

По жанру Space Invaders — shoot 'em up, в котором игрок управляет лазерной пушкой, передвигая её горизонтально, в нижней части экрана, а также отстреливая инопланетян, надвигающихся сверху экрана. Целью игры является уничтожение пяти рядов по одиннадцать инопланетян, которые двигаются горизонтально, а также вертикально, по направлению к низу экрана. Игрок имеет бесконечное количество патронов. Попадая в инопланетянина, игрок уничтожает его, за что получает очки. При уничтожении инопланетян увеличивается скорость движения оставшихся, а также ускоряется темп звуковых эффектов. При уничтожении всех инопланетян появляется новая, ещё более сильная волна, а игрок получает одно дополнительное очко жизни. Количество новых волн инопланетян неограниченно, что делает игру бесконечной.

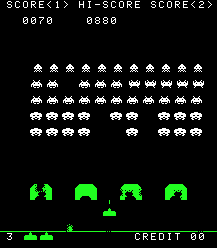


Рисунок 1.1 –Внешний вид игрового поля «Space Invaders»

Позже игра была перевыпущена на многочисленных платформах (Nintendo 64, Game Boy Advance, Atari 2600, Microsoft Windows, DOS и т.д.), было выпущено несколько сиквелов. Версия для Atari 2600 увеличила продажи консоли в четыре раза и стала первой killer app (игрой, которая увеличивает продажи платформы, на которой она выпускается) в истории видеоигр. «Space Invaders» часто пародируется, является частью нескольких видеоигр, а пиксельное изображение инопланетянина стало символом поп-культуры и часто используется как обобщенное обозначение видеоигр в целом.

«Galaxian» — аркадная игра, разработанная Namco в октябре 1979 года. Она была издана Namco в Японии. Игра стала популярной сразу после выпуска. Было выпущено два продолжения: «Galaga» в 1981 году и менее известное «Gaplus»в 1984 году, а также большое число портов и адаптаций. Как и продолжения, игра была одной из самых популярных в течение золотого века аркадных видеоигр.

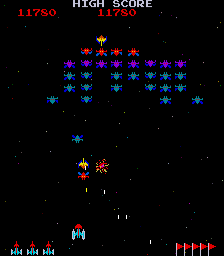


Рисунок 1.2 –Внешний вид игрового поля «Galaxian»

Как и в более ранней игре, в «Galaxian» присутствовала армада атакующих пришельцев, которые обменивались выстрелами с игроком. В отличие от «Space Invaders», в «Galaxian» пришельцы периодически превращались в камикадзе, пикирующих на корабль игрока (который назывался Galaxip). Таким образом, данная игра стала первой, в которой поведение врагов имело некую индивидуальность.

Galaxian был очень успешен и во многих аспектах стал первой игрой в своём роде. Хотя Galaxian и не был первой цветной игрой, в нём использовались многоцветные анимированные спрайты и взрывы, шрифты разного цвета для числа очков и высшего достижения, прокручивающееся игровое поле со звёздами и графические значки, которые показывают число оставшихся кораблей и уровней, которые прошёл игрок. В игре также присутствует грубая музыкальная тема и фоновая музыка. Сочетание этих элементов определило стандарт, которому следовали многие аркадные игры 1980-х, такие как Pac-Man.

В 1986 г. компаниями Sega и Tectoy была выпущена игра «ASTRO WARRIOR» для Sega Master System. Она также является представителем жанра Shoot ’em up. Если учесть, когда была выпущена эта игра, не может не поразить звуковое сопровождение, игровой процесс, динамичность боя.



Рисунок 1.3 –Внешний вид игрового поля «ASTRO WARRIOR»

В большинстве отзывах Astro Warrior получила не слишком высокие оценки. Критике часто подвергались низкое количество уровней, не проработанное управление и низкое разнообразие оружия. Рецензенты интернет портала GameFAQs поставили игре оценки 7/10 и 5/10, коммерческая информационная база данных компьютерных игр для различных платформ — Allgame, оценила её в 3,5 звёздочки из 5, а испаноязычный журнал VideoGame за апрель 1991 года — в 3/5.

Изучив аналоги, можно сделать вывод: существует большое количество ремейков старых популярных видеоигр жанра Shoot ’em up, однако принципиально новых подходов к разработке аркад данной тематики не наблюдается.

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработка игрового программного средства «Space Shooter», адаптированного под запуск на современных операционных системах, в частности, версиях ОС Windows 8 и старше, на компьютерах с Full HD разрешением.

* Для разработки программного средства будет использован язык ассемблера и транслятор flat assembler.
* Проанализировав аналоги программного средства, можно поставить ряд задач:
* изучить язык ассемблера для программирования под ОС Windows;
* изучить основы WinAPI;
* детально изучить интерфейс Graphics Device Interface (GDI) и Windows USER;
* разработать конечный автомат, который будет основой архитектуры программого средства;
* разработать классы объектов для реализации игрового процесса;
* оптимизировать использование программой аппаратных ресурсов;
* оптимизировать программу для работы с большим количеством графических изображений.

В игровом приложении «Space Shooter» планируется реализовать следующие функции:

* контролируемое перемещение космического корабля по пространству игрового поля;
* ведение огня по противнику;
* динамическое создание противников и других объектов;
* организация ведения вражеского огня по кораблю пользователя;
* проверка столкновения противников и игрока;
* увеличение сложности игрового процесса в течение времени игры;
* учет количества очков здоровья всех боевых единиц, отображение очков здоровья, возможность восстанавливать очки здоровья во время ведения боя;
* анимация взрывов и элементов космических кораблей;
* звуковое сопровождение выстрелов игрока;
* подсчет количества очков и сохранение рекорда;
* возможность приостановки игрового процесса и выхода в меню;
* перезапуск игрового процесса.

1. Разработка программного средства
   1. Структура программного средства
      1. Основные модули

В данном приложении реализованы следующие модули:

* SpaceShooter.asm – основной модуль приложения, содержит функцию WinMain, которая с помощью цикла удаляет сообщения окна из очереди и пренаправляет их в оконную процедуру или в другие модули для обработки;
* MainWindow.c – модуль, содержащий оконную процедуру и функцию создания окна;
* MainWindow.d – модуль данных для создания окна;
* Game.c – модуль, содержащий функции для инициализации приложения, а также функцию отрисовки игрового пространства;
* Game.d – модуль данных, содержащий классы объектов для реализации игрового процесса;
* Game.h – модуль, содержащий константные данные и имена файлов;
* Game.di – модуль импортируемых данных;
* Game.Welcome.c, Game.Run.c, Game.Pause.c, Game.GameOver.c – модули-контроллеры, которые управляют игровым процессом в зависимости от состояния автомата;
* Calculate.c, Calculate.d – вспомогательные модули, содержащие математические функции и необходимые данные для расчетов;
  + 1. Создание окна

Для создания приложения Windows с помощью WinAPI требуются две функции: оконная процедура и точка входа в приложение, в которой необходимо:

* зарегистрировать класс окна;

Для этого нужно заполнить поля структуры WNDCLASSEX, а затем передать указатель на структуру в функцию RegisterClassEx.

* создать окно;

Чтобы создать окно нужно вызвать функцию CreateWindowEx, аргументами которой являются класс окна, имя окна, стили окна, размер, положение, дескриптор меню, дескриптор экземпляра и данные создания. Для организации полноэкранного режима необходимо определить ширину и высоту монитора с помощью функции GetSystemMetrics.

* запустить цикл обработки сообщений;

Функция PeekMessage используется для проверки наличия сообщений в очереди, если возвращаемое значение равно нулю – сообщений нет. В это время выполняется обновление игрового процесса и отрисовка графики.

Такая организация цикла обработки сообщений позволяет более эффективно использовать ресурсы процессора и обеспечивает более качественную перерисовку, в сравнении с использованием системных таймеров. Параметр wRemoveMsg определяет, как сообщения должны быть обработаны, при передаче PM\_NOREMOVE сообщения не удаляются из очереди. Функция GetMessage извлекает сообщение из очереди, а DispatchMessage отправляет его на последующую обработку в оконную процедуру.

Фрагмент функции WinMain, который управляет обработкой сообщений, представлен ниже:

locals

Msg MSG

endl

lea edi, [Msg]

.MsgLoop:

invoke PeekMessage, edi, 0, 0, 0, PM\_NOREMOVE

test eax, eax

jz .Idle

invoke GetMessage, edi, 0, 0, 0

test eax, eax

jz .EndMsgLoop

invoke DispatchMessage, edi

jmp .MsgLoop

.Idle:

;обновление и отрисовка

Оконная процедура – это определяемая приложением функция, обрабатывающая сообщения, отправленные окну. Она имеет четыре параметра: дескриптор окна hWnd, идентификатор сообщения uMsg и два параметра, определяемых типом сообщения, wParam и lParam, которые содержат дополнительную информацию.

Так как было принято решение не использовать системные сообщения для организации игрового процесса, в оконной процедуре MainWindow.WindowProc обрабатываются только необходимые для функционирования приложения сообщения: WM\_CREATE, WM\_PAINT и WM\_DESTROY. При создании окна вызывается функция Game.Welcome, которая инициализирует состояние игры в GS\_WELCOME.

* + 1. Организация обновления игрового процесса

Как было описано ранее, приложение произвоит все обновления во время отсутствия сообщения в очереди. Таким образом, когда происходит выход из цикла, сначала проверяется, в каком состоянии находится игра, затем в зависимости от этого вызываются функция обновления для нужного состояния:

.Idle:

invoke GetTickCount

mov esi, eax

cmp [Game.state], GS\_WELCOME

je .welcome

cmp [Game.state], GS\_PAUSE

je .pause

cmp [Game.state], GS\_GAMEOVER

je .gameover

cmp [Game.state], GS\_RUN

je .run

jmp .MsgLoop

.welcome:

stdcall Game.Welcome.Update, esi

jmp .MsgLoop

.pause:

stdcall Game.Pause.Update, esi

jmp .MsgLoop

.gameover:

stdcall Game.GameOver.Update, esi

jmp .MsgLoop

.run:

stdcall Game.Run.Update, esi

jmp .MsgLoop

Организация таймеров в приложении происходит следующим образом: в начале игры происходит инициализация данных таймеров (текущего времени), затем при каждом вызове процедуры обновления производится проверка интервала прошедшего времени, если время истекло, выполняется обновление и сохранение нового времени, если нет – выход из процедуры. Функция GetTickCount возвращает количество миллисекунд, прошедших с момента запуска системы. Ее использование позволяет определять интервалы между обновлениями.

В приложении используется восемь таймеров, интервалы времени для которых заданы константами TIMER\_UPDATE (18), TIMER\_BULLET (200), TIMER\_EWAVE (1000), TIMER\_METEOR (3000), TIMER\_BONUS (4000), TIMER\_SCORE (1000), TIMER\_BACKGROUND (50), TIMER\_ANIMATION (20).

Один глобальный таймер обеспечивает прокрутку фонового изображения и используется во всех состояниях. Таймер UPDATE используется в состояниях GS\_WELCOME, GS\_PAUSE и GS\_GAMEOVER, и обеспечивает обновление графики. Все остальные таймеры используются в состоянии GS\_RUN для определения интервала между волнами врагов, скорости генерирования патронов, частоты появления бонусов и помех, начисления бонусов. Обновление графики в состоянии игры происходит каждый раз, когда очередь сообщений пуста для обеспечения более высокого качества.

Фрагмент процедуры Game.Run.Update, в котором производится проверка таймеров, представлен ниже:

mov esi, [TickCount]

mov eax, [Timer.Bonus.TickCount]

add eax, [TIMER\_BONUS]

cmp eax, esi

jb .bonus

…

mov eax, [Timer.Bullet.TickCount]

add eax, [TIMER\_BULLET]

cmp eax, esi

jb .bullet

;если сигнал не поступал от других таймеров – обновление графики

.update:

* + 1. Объекты игрового пространства

Для приближения стиля написания программы к объектно-ориентированной модели, улучшения читаемости кода и создания четкой структуры были разработаны классы объектов, каждый из которых имеет свой набор данных и методы.

Данные класса представляют собой набор переменных, в которых содержится информация о положении, размерах, состояниии и типе объекта. Методы представляют собой функции, которые вызываются при необходимости обновления объекта.

Общая структура объектов приведена в схеме на рисунке 2.1:



Рисунок 2.1 – Схема общей структуры объектов

* 1. Интерфейс программного средства
     1. Инициализация графических данных

Производить подготовку графического материала оказалось особо удобно в программах Krita и Paint.net. Выбор Paint.net обусловлен быстротой реализации и необходимостью использования лишь нескольких простых функций. Графический редактор Krita использовался в тех случаях, когда функционал Paint.net не позволял реализовать текущую задачу за короткий промежуток времени. Так, например, были отделены текстуры кораблей от их фона, созданы спрайты взрывов, движения снарядов и элементов космических кораблей.

В результате анализа средств WinAPI был сделан вывод, что имеющиеся визуальные средства управления (BUTTON, EDIT и другие) не подходят для гибкой работы с изображением из-за отсутствия возможности тонкой настройки. Таким образом, было принято решение отказаться от использования базовых компонентов. Вместо этого происходит отрисовка всего игрового пространства.

При запуске программа выполняет три функции инициализации графики Game.InitGraphics, Game.LoadImages и Calc.Coordinates.

В первой процедуре с помощью функции GetDC приложение получает дескриптор контекста устройства для клиентской области окна, также создается контекст устройства в памяти (CreateCompatibleDC), совместимый с данным утройством.

Процедура Game.InitGraphics приведена ниже:

proc Game.InitGraphics uses esi eax

invoke GetDC, [MainWindow.hWnd]

mov esi, eax

invoke CreateCompatibleDC, esi

mov [Buffer.hdc], eax

invoke CreateCompatibleDC, esi

mov [Memory.hdc], eax

invoke ReleaseDC, 0, esi

ret

endp

В процедуре Game.LoadImages происходит загрузка из файла всех необходимых приложению графических ресурсов с помощью функции LoadImadeW. Возвращаемые значения – это дескрипторы изображений, которые сохраняются в переменные в памяти для последующего использования. В качестве основных параметров указывается нуль-терминированная строка с адресом картинки, тип изображения, высота и ширина (при передаче нуля функция использует действиетельные значение высоты и ширины).

Пример вызова функции при загрузке растрового фонового изображения:

invoke LoadImageW, 0, IMG\_BG, IMAGE\_BITMAP, 0, 0,\

LR\_LOADFROMFILE

mov [Background.hbm], eax

Для того чтобы обеспечить качественное рисование графики в полноэкранном приложении на мониторах разного разрешения, все используемые координаты и размеры объектов рассчитываются относительно размеров экрана перед загрузкой программы в процедуре Calc.Coordinates. В стадии разработки было выявлено, что для проведения всех расчетов достаточно двух функций, которые выполняются наиболее часто.

Процедура Calc.RelativeValue нужна для расчета относительных размеров объекта и имеет три параметра: Value, DivValue, MulValue. Например, известно, что высота изображения корабля игрока относится к ширине как 3:2, тогда для ее расчета необходимо умножить ширину на три и разделить на два, то есть вызов функции будет выглядеть следующим образом:

stdcall Calc.RelativeValue, [Player.width], 2, 3

mov [Player.height], eax

Вторая функция Calc.GetCenterCoord определяет координаты объекта для расположения его в центре относительно другого, ее праметры – WidthMain, WidthObj. Например, необходимо расчитать координаты кнопки так, чтобы она располагалась в центре экрана. Для этого нужно разделить ширину экрана и кнопки на два, а затем отнять от половины ширины экрана половину ширины кнопки, получим координату x.

Пример вызова функции для подобного расчета приведен ниже:

;edi – ширина экрана, [Welcome.Start.right] – ширина кнопки

stdcall Calc.GetCenterCoord, edi, [Welcome.Start.right]

mov [Welcome.Start.left], eax

* + 1. Оптимизация графики

Одной из основных целей курсового проекта была оптимизация расходов времени и ресурсов процессора на отрисовку и при этом улучшение качетсва графики. В ходе изучения теоретических материалов и разработки приложения были выявлены основные параметры, влияющие на скорость и эффективность программного средства:

* частота перерисовки;
* способ отрисовки;
* количество используемых изображений;
* сложность расчетов непосредственно в течение игрового процесса;

Частота перерисовки – важный фактор, определящий эффективность приложения. В ходе изучения теории архитектуры компьютера было выявлено, что максимальная частота обновления системного счетчика составляет 15 мкс, из чего следует, что использование большей частоты не имеет смысла, а наоборот, только загружает процессор лишними вычислениями. При тестировании программного средства оптимальной частотой для получения наилучшего качества графики оказалось 18 мкс.

Для улучшения качетсва изображения и увеличения скорости отрисовки используется двойная буфферизация, то есть отрисовка отдельных объектов происходит сначала в контексте графического устройства в памяти, затем изображение переносится на экран.

Картинка – это массив байтов, в которых содержится информация о цветах. Отрисовка происходит за счет копирования набора байт из одного устройства в другое, в памяти этот процесс занимает значительно меньше времени, что позволяет разместить все изображения игровых объектов в одной области быстро. Также такая технология позволяет избежать миганий при перерисовке и увеличить частоту смены изображений.

Оптимизировать способ отрисовки можно за счет неполного копирования байт изображения. В данном приложении такая оптимизация происхоит за счет использования функции TransparentBlt, одним из параметров которой является шестнадцатеричное значение цвета, который пропускается при копировании. За счет этого получается эффект частичной прозрачности картинки. При тестировании также было замечено значительное увеличение скорости при частом обновлении, в сравнении с функциями StretchBlt и BitBlt.

Еще одним плюсом функции TransparentBlt является возможность копирования растрового изображения из исходного прямоугольника в целевой, растягивая или сжимая его до нужных размеров. Система растягивает или сжимает растровое изображение в соответствии с режимом растяжения, установленным в настоящее время в контексте целевого устройства. При тестировании программного средства был сделан вывод о том, что режим HALFTONE лучше других обеспечивает сохранение качества изображения.

Выбор режима растяжения и пример использования функции TransparentBlt представлен во фрагменте функции Game.Draw:

invoke SetStretchBltMode, [Buffer.hdc], HALFTONE

…

invoke GetObject, [Meteor.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Meteor.hbm]

invoke TransparentBlt, [Buffer.hdc], [Meteor.x],\

[Meteor.y], [Meteor.width], [Meteor.height],\

[Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight],\ $00000000

Отрисовка фонового изображения занимает больше всего времени из-за размера, поэтому этот вопрос пришлось рассматривать отдельно. В процессе разработки был найден способ оптимизации, который повлек за собой итересный графический эффект.

В качестве фона была выбрана картинка космоса черного цвета с белыми звездами. Чтобы обеспечить движение фона, производится последовательная отрисовка разных частей одного изображения со смещением вниз, для этого изначально картинка была изменена, а именно увеличена по высоте за счет объединения исходного изображения с его зеркальным отражением.

Простая передача большого битового блока цветовых данных сильно замедляет работу программы, поэтому используется функция TransparentBlt с указанием белого цвета в качестве прозрачного. Это достаточно ускоряет отрисовку. За счет того, что полного перекрытия не происходит, получается эффект мерцания звездного неба. Такой способ оптимизации хорошо работает только с выбранным изображением, однако было принято решение его оставить, так как для обеспечения выше описанного эффекта не затрачиваются дополнительные программные и системные ресурсы.

Фрагмент фонового изображения представлен ниже:

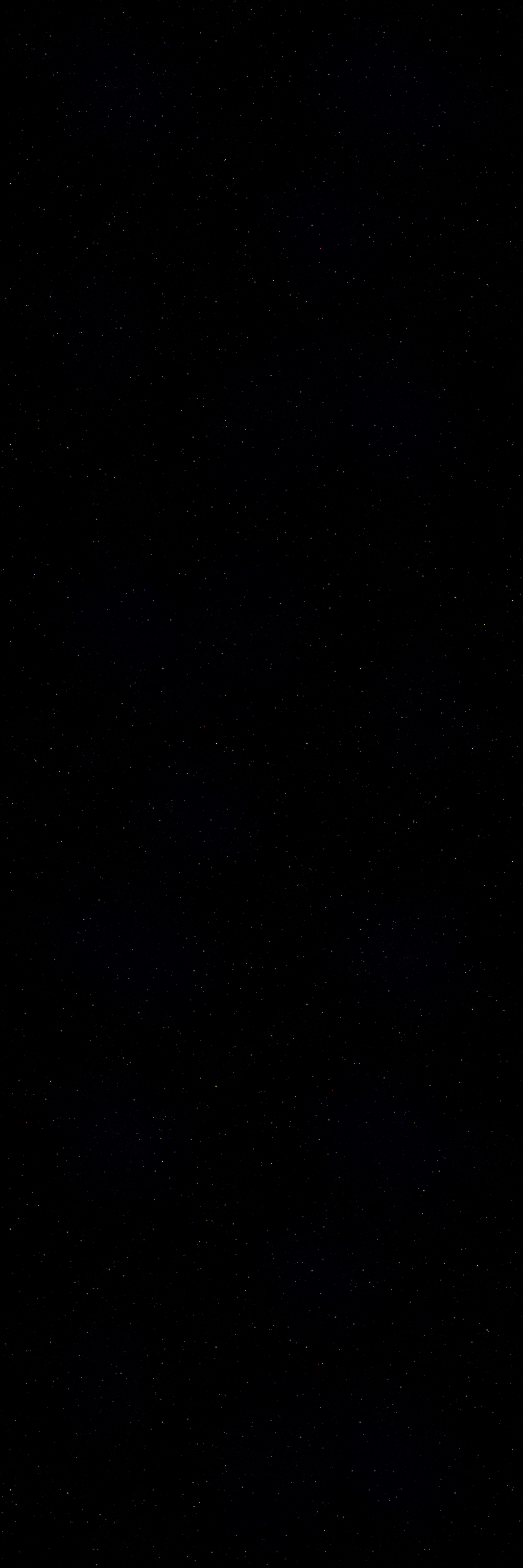


Рисунок 2.2 – Фрагмент фонового изображения

Дескрипторы всех изображений, которые используются в приложении, хранятся в разных переменных, и сами изображения (массивы байтов) загружаются в разные области оперативной памяти. При использовании большого количества картинок, работа с ними замедляется из-за сложного доступа к данным, также увеличивается размер программы. Хорошим решением этой проблемы стало использование спрайтов. Спрайтом, называют растровое изображение, содержащее различные состояния одного объекта, что эффективно применяется для работы с анимацией. В идеальном случае необходимо найти баланс между количеством изображений, размещенных на одном спрайте, и сложностью расчета координат. В данном проекте спрайты используются для объедения состояний отдельных объектов.

Пример спрайта для вражеских кораблей приведен ниже:

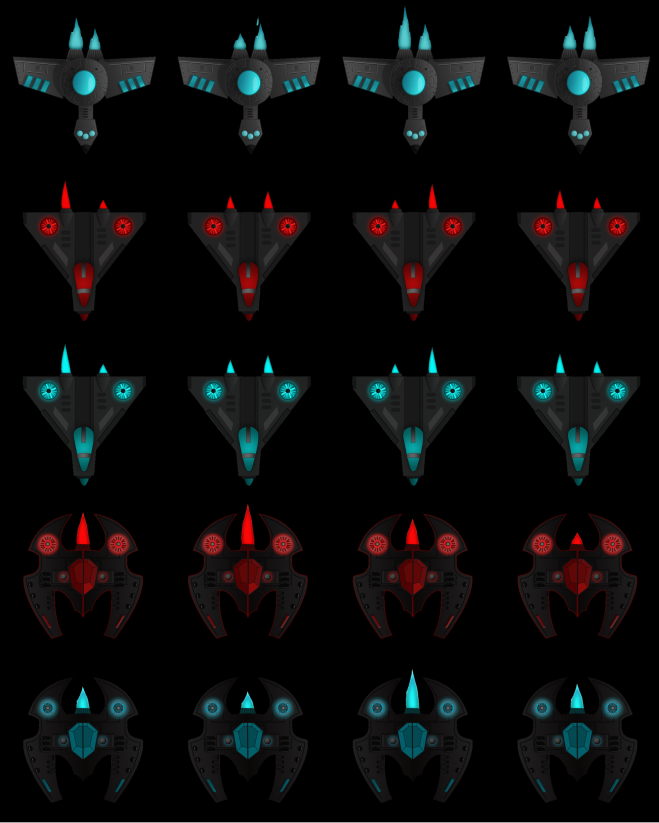


Рисунок 2.3 – Спрайт для вражеских кораблей

В программе были введены дополнительные переменные, в которых хранится состояние и тип объекта. С помощью этих переменных рассчитываются координаты нужной части спрайта. Например: необходимо отрисовать вражеский корабль, тип которого равен двум (типы представлены номерами от нуля до четырех), а сотояние – трем (от нуля до трех). Из примера видно, что в каждой строке размещены разные типы кораблей, а их состояния меняются по столбцам. Значит, для получения координаты y нужно умножить высоту одного корабля на два, для получения координаты x – ширину корабля умножить на три.

Цикл отрисовки всех вражеских кораблей представлен ниже:

xor esi, esi

.enemy:

mov eax, [Enemy.type + esi \* sizeof.DWORD]

imul eax, ENEMY\_WIDTH

mov ebx, [Enemy.state + esi \* sizeof.DWORD]

imul ebx, ENEMY\_HEIGHT

invoke TransparentBlt, [Buffer.hdc],\

[Enemy.x + esi \* sizeof.DWORD],\

[Enemy.y + esi \* sizeof.DWORD], [Enemy.width],\

[Enemy.height], [Memory.hdc], ebx, eax,\

ENEMY\_WIDTH, ENEMY\_HEIGHT, $00000000

add esi, 1

cmp esi, [Enemy.count]

jb .enemy

Сложность расчетов в течение игрового процесса влияет на скорость обновления графики. При разработке данного приложения были применены следующие способы оптимизации:

* проведение всех длительных расчетов перед запуском приложения или во время малой загруженности процессора;

Расчет всех постоянных координат и размеров, загрузка изображений, инициализация переменных происходит при запуске программы, обновление некоторых данных происходит в состоянии паузы и конца игры.

* использование констант;

В программе отведен отдельный модуль Game.h под константы, в нем заданы все значения, которые известны заранее и используются в вычислениях.

* использование регистров;

Доступ к значениям, хранящимся в регистрах, как правило, быстрее, чем доступ к переменным в памяти. Если некоторые переменные в процедуре используются часто, они копируются в регистры в начале процедуры. Например, в функции Calc.Coordinates часто используются значения ширины и высоты экрана, поэтому в начале процедуры они сохраняются в регистры esi и edi.

* замена сложных операций более простыми;

В программе почти не используется оператор div, так как на его выполнение уходит сравнительно продолжительное время, что при частом использовании влияет на скорость. По возможности вычисления модифицируются таким образом, чтобы достаточно было оператора shr (сдвиг вправо), так как сдвиги выполняются быстрее. По этой же причине оператор mul заменяется на shl (сдвиг влево).

* + 1. Создание элементов управления

Для создания собственных элементов управления, а именно кнопок, в превую очередь необходимо было подготовить изображения. Для этой задачи использовался графический редактор Krita. Были разработаны четыре кнопки: «Start», «Continue», «Restart» и «Exit», – которые имееют два состояния. Смена состояния происходит при наведении курсора.

Кнопки в приложении есть в состояниях GS\_WELCOME, GS\_PAUSE и GS\_GAMEOVER. Обновление и проверка состояния происходит по таймеру TIMER\_UPDATE. Для проверки нажатия кнопки используются три функции: GetCursorPos (возвращает структуру типа POINT, которая содержит координаты курсора), GetKeyState (определяет состояние левой кнопки мыши), вспомогательная функция Calc.CheckCoordinates (проверяет, находится ли курсор над кнопкой). Затем с помощью операции побитового «И» складываются значения, которые вернули две последние функции. Если результат равен единице – кнопка была нажата.

Вид кнопок представлен ниже:



Рисунок 2.4 – Вид кнопок

Было принято решение не использовать системный курсор, вместо этого в состоянии игры он скрывается, а в остальных состояниях представляет собой анимацию космического явления «Белой дыры».

* 1. Игровая логика

Основные алгоритмы, реализованные в приложении:

* генерирование новой волны вражеских кораблей;
* генерирование выстрелов игрока и врагов;
* всевозможные проверки столкновений;

Враги в игре появляются группами, одно такое появление называется волной. Количество врагов увеличивается со временем и хранится в переменной Enemy.count. Генерирование новой волны производится в процедуре Game.Run.EWave, которая вызывается, когда все враги уничтожены. При помощи цикла по массиву производится инициализация. Координаты по оси X выбираются случайно, однако выполняется проверка на пересечение с другими вражескими кораблями, для этого выполняется еще один проход по массиву. Координаты по оси Y инициализируются в ноль, а координаты ymax выбираются случайно.

Фрагмент процедуры Game.Run.EWave, в котором производится выбор координаты X, представлен ниже:

.choose\_x:

stdcall Random, 0, edi

mov edx, eax

xor ebx, ebx

.cmp\_equal:

mov eax, edx

add eax, [Enemy.width]

cmp eax, [Enemy.x + ebx \* sizeof.DWORD]

jb .cmp\_continue

mov eax, [Enemy.x + ebx \* sizeof.DWORD]

add eax, [Enemy.width]

cmp eax, edx

jb .cmp\_continue

jmp .choose\_x

.cmp\_continue:

add ebx, 1

cmp ebx, ecx

jb .cmp\_equal

mov [Enemy.x + ecx \* sizeof.DWORD], edx

Данные о ракетах, запускаемых пользователем, хранятся в массиве. Каждая ракета во время игры может быть в одном из двух состояний – активна и неактивна. При генерировании выстрела выполняется проход по массиву, пока неактивная ракета не будет найдена. Количество ракет выпускаемых кораблем игрока за один выстрел увеличивается через каждые три волны.

Генерирование начальных координат происходит по следующему алгоритму:

mov esi, [Player.Bullet.state]

mov eax, [Player.width]

xor edx, edx

div esi

mov esi, eax

mov eax, [Player.Bullet.width]

shr eax, 1

sub esi, eax

mov ebx, esi

mov edi, [MainWindow.ClientRect.right]

xor ecx, ecx

xor esi, esi

.create:

cmp [Player.Bullet.x + sizeof.DWORD \* ecx], edi

je . createnew

jmp .continue

.createnew:

mov eax, ebx

xor edx, edx

mul esi

add eax, [Player.x]

mov [Player.Bullet.x + sizeof.DWORD \* ecx], eax

mov eax, [Player.y]

mov [Player.Bullet.y + sizeof.DWORD \* ecx], eax

.continue:

add ecx, 1

cmp esi, [Player.Bullet.state]

jnz .create

Для выполнения поставленной задачи необходимо было реализовать метод определяющий, столкнулись ли противники с кораблем игрока в данный момент. Для этого было решено использовать взаиморасположение границ кораблей. При наличии столкновения игрок считается поверженным и игра завершается. Фрагмент цикла проверки на столкновения:

mov esi, [Enemy.y + ecx \* sizeof.DWORD]

add esi, [Enemy.height + ecx \* sizeof.DWORD]

mov edi, [Player.y]

cmp esi, edi

jb .next\_iteration

mov edi, [Player.y]

add edi, [Player.height]

mov esi, [Enemy.y + ecx \* sizeof.DWORD]

cmp esi, edi

ja .next\_iteration

mov esi, [Enemy.x + ecx \* sizeof.DWORD]

add esi, [Enemy.width + ecx \* sizeof.DWORD]

mov edi, [Player.x]

cmp esi, edi

jb .next\_iteration

mov edi, [Player.x]

add edi, [Player.width]

mov esi, [Enemy.x + ecx \* sizeof.DWORD]

cmp esi, edi

ja .next\_iteration

stdcall Game.GameOver

3 тестирование программного средства

В процессе разработки программного средства были выявлены некоторые проблемы, связанные с оптимизацией игрового процесса.

Во-первых, использование системных таймеров значительно замедляло работу приложения и не позволяло производить обновления достаточно часто. Системный таймер помещает в очередь сообщение WM\_TIMER, когда указанный интервал времени истекает. Сообщение может быть извлечено из очереди и обработано только в случае, если очередь пуста. Точность таймера зависит от частоты системных часов и от того, как часто приложение извлекает сообщения из очереди, значение времени ожидания является лишь приблизительным. Кроме этого, оказалось невозможным организовать состояние паузы, так как в системном таймере нет функции остановки. Проблема была решена благодаря использованию функции GetTickCount для организации таймеров в приложении приложения. В состоянии паузы значения для таймеров игрового процесса не изменяются, поэтому для продолжения достаточно обновить их значения:

Invoke GetTickCount

mov [Timer.EWave.TickCount], eax

mov [Timer.Bullet.TickCount], eax

mov [Timer.Meteor.TickCount], eax

mov [Timer.Bonus.TickCount], eax

mov [Timer.Score.TickCount], eax

mov [Timer.Animation.TickCount], eax

Иной крупной проблемой оказалось замедление изменения игровой ситуации при увеличенном количестве противников и снарядов. Это было связано со сложностью вычислений. Проблема была решена путем оптимизации и упрощения структуры игровой логики.

Так как практически все структуры данных в программе статические, по большей части массивы, для каждого типа объектов было выбрано их максисмальное количество. В случае такой организации данных всегда есть неиспользуемые в данный момент объекты. Изначально было принято решение присваивать ненужным игровым единицам определенные координаты вне области экрана и производить отрисовку всех имеющихся объектов. Это повлекло за собой многочисленные ошибки, усложнение логики проверок и нарушение стуктуры программы. Решением проблемы стало введение у объетков групп еще одного поля (переменной) – \*.activity, а также проверка состояния при отрисовке:

xor esi, esi

.enemy\_hp:

cmp [Enemy.activity + esi \* sizeof.DWORD], ENEMY\_ACTIVE

jne .ehp\_continue

…

.ehp\_continue:

add esi, 1

cmp esi, [Enemy.count]

jb .enemy\_hp

Обработка нажатий клавиш для управления кораблем игрока изначально проводилась по сообщению WM\_KEYDOWN окна. Однако это оказалась неэффективно, так как перемещение происходило очень медленно, невозможно было отследить нажатие сразу двух или более клавиш, из-за чего двигаться и стрелять одновременно, было нельзя.

Решением этой проблемы стал отказ от использования прерываний клавиатуры при обработке ввода. Вместо этого используется функция GetKeyState, которая возвращает состояние клавиши в данный момент. Проверка ввода производится по таймеру. Использование этой функции позволило организовать не только более быстрое движение игрока, но и позволило внедрить новый вид движения по диогонали, так как возможно отследить нажатие сразу нескольких клавиш.

Остальные проблемы были связаны с недочетами на стадии проектирования и были устранены на стадии разработки.

4 Руководство пользователя

Находясь в главном меню, пользователь может начать новую игру, нажав на интерактивную кнопку с надписью «Start», либо завершить работу программы нажатием кнопки «Exit». Вид главного меню представлен на рисунке 4.1.



Рисунок 4.1 – Главное меню игры

Во время пребывания в игровом окне пользователь может приостановить игру, нажав Esc. Из состояния паузы игрок может продолжить игру, нажав кнопку «RESUME», или выйти, нажав кнопку «Exit».

При управлении игрой стоит пользоваться стрелками для движения и клавишей SPACE для совершения выстрелов. В случае проигрыша, появляется надпись «GAME OVER», есть возможность начать новую игру, нажав кнопку «RESTART», либо покинуть игру, нажав кнопку «EXIT». Окно завершения игры показано на рисунке 4.2.



Рисунок 4.3 – Окно завершения игры

Парвила игры:

* За каждое попадание по противнику игрок получает очко счета. Также очки начисляются за каждую секунду игры.
* Враги появляются волнами, с первой волной появляется один враг, затем каждый раз их количество увеличивается.
* Ущерб от выстрелов врагов определяется типом их кораблей.
* Дополнительную сложность представляют собой метеориты, которые тоже повреждают игрока, количество очков здоровья, которые они отнимают, заранее не известно. Чем больше метеорит, тем сильнее он повредит корабль. Если игрок стоит на одном месте, в него летит метеорит.
* С ростом уровня сложности боевые характеристики противников улучшаются, а количество снарядов игрока увеличивается.
* Игрок получает дополнительные очки здоровья, если ловит бонусы.
* Игрок проигрывает, если количество очков здоровья его корабля падает до нуля или он столкнулся с кораблями противника.

Вид игрового окна представлен на рисунке 4.3.



Рисунок 4.3 – Игровое окно

ЗАКЛЮЧЕНИЕ

В современном мире существует множество игровых приложений, которые помогают занять свободное время в очереди или в дороге, отдохнуть или просто хорошо провести время с друзьями. Огромное количество людей ежедневно играют в компьтерные игры и спрос на них только растет.

В рамках данной курсовой работы было создано игровое приложение «Space Shooter» предназначенное для развлечения пользователя. Разработанное программное средство отличается простым и понятным интерфейсом, плавностью и высокой скоростью работы, а также визуализацией взаимодействия с элементами интерфейса.

Цели, поставленные перед разработкой данного приложения, были успешно достигнуты:

* разработан конечный автомат, включающий четыре состояния, в зависисмости от состояния которого происходит отрисовка и обработка сообщений.
* разработаны объекты игрового пространства, такие как вражеские корабли, бонусы, метеориты, корабль игрока и другие, описаны все необходимые для их функционирования данные.
* проведена оптимизация графики и скорости работы приложения;
* создан алгоритм движения персонажа и его взаимодействия с окружающими объектами;
* в процессе разработки были приобретены навыки создания элементов приложения вручную при помощи графического редактора;
* был освоен навык создания приложений с использованием WinAPI;
* были углубленно изучены некоторые аспекты языка ассемблера.

Архитектура программы является близкой к объектно-ориентированной модели, что обеспечивает достаточно несложную сопровождаемость кода и его понимание. В будующем планируется введение в игру следующих дополнительных функций и особенностей:

* дополнительные жизни главного героя, которые будут расходоваться при поражении;
* разбиение игрового процесса не только на волны врагов, но и на уровни сложности;
* наличие уровня «БОСС», который переодически повторяется.

загрузка сохранения пользователя, что позволит продолжить игру в более удобное время.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Bootstrap Tabs and Pills [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.w3schools.com/bootstrap/  
bootstrap\_tabs\_pills.asp.

[2] Bootstrap Tables [Электронный ресурс]. – Электронные данные. – Режим доступа: https://getbootstrap.com/docs/4.0/content/tables/.

[3] ESP8266 core for Arduino [Электронный ресурс]. – Электронные данные. – Режим доступа: https://github.com/esp8266/Arduino.

[4] ESP8266WiFi library [Электронный ресурс]. – Электронные   
данные. – Режим доступа: https://arduino-esp8266.readthedocs.io/en/latest/  
esp8266wifi/readme.html

[5] [Adafruit\_BME280\_Library](https://github.com/adafruit/Adafruit_BME280_Library) [Электронный ресурс]. – Электронные данные. – Режим доступа: https://github.com/adafruit/  
Adafruit\_BME280\_Library/blob/master/Adafruit\_BME280.cpp.

[6] I2C Protocol [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/I²C.

приложение а  
Исходный код программы

Модуль SpaceShooter:

format PE GUI 4.0

entry WinMain

include 'win32w.inc'

include 'units\Game.h'

section '.code' code readable writeable

proc WinMain

locals

Msg MSG

endl

invoke GetModuleHandle, 0

mov [MainWindow.WindowClassEx.hInstance], eax

stdcall MainWindow.Create

lea edi, [Msg]

.MsgLoop:

invoke PeekMessage, edi, 0, 0, 0, PM\_NOREMOVE

test eax, eax

jz .Idle

invoke GetMessage, edi, 0, 0, 0

test eax, eax

jz .EndMsgLoop

invoke DispatchMessage, edi

jmp .MsgLoop

.Idle:

invoke GetTickCount

mov esi, eax

mov eax, [Timer.Background.TickCount]

add eax, [TIMER\_BACKGROUND]

cmp eax, esi

ja .state

stdcall Game.Background.Update

mov [Timer.Background.TickCount], esi

.state:

cmp [Game.state], GS\_WELCOME

je .welcome

cmp [Game.state], GS\_PAUSE

je .pause

cmp [Game.state], GS\_GAMEOVER

je .gameover

cmp [Game.state], GS\_RUN

je .run

jmp .MsgLoop

.welcome:

stdcall Game.Welcome.Update, esi

jmp .MsgLoop

.pause:

stdcall Game.Pause.Update, esi

jmp .MsgLoop

.gameover:

stdcall Game.GameOver.Update, esi

jmp .MsgLoop

.run:

stdcall Game.Run.Update, esi

jmp .MsgLoop

.EndMsgLoop:

stdcall Game.SaveRecord

invoke ExitProcess, 0

endp

include 'units\Game.c'

include 'units\MainWindow.c'

include 'units\Calculate.c'

section '.data' data readable writeable

include 'units\MainWindow.d'

include 'units\Game.d'

include 'units\Calculate.d'

section '.idata' data import readable writeable

include 'units\Game.di'

Модуль MainWindow:

proc MainWindow.Create

invoke LoadCursor, 0, IDC\_ARROW

mov [MainWindow.WindowClassEx.hCursor], eax

invoke RegisterClassEx, MainWindow.WindowClassEx

invoke GetSystemMetrics, SM\_CXSCREEN

mov [MainWindow.ClientRect.right], eax

invoke GetSystemMetrics, SM\_CYSCREEN

mov [MainWindow.ClientRect.bottom], eax

invoke CreateWindowEx, 0,MainWindow.ClassName,0,WS\_POPUP or WS\_VISIBLE,[MainWindow.ClientRect.left],[MainWindow.ClientRect.top],[MainWindow.ClientRect.right],[MainWindow.ClientRect.bottom],0,0,[MainWindow.WindowClassEx.hInstance], 0

mov [MainWindow.hWnd], eax

ret

endp

proc MainWindow.WindowProc uses ebx esi edi,\

hWnd, uMsg, wParam, lParam

locals

PS PAINTSTRUCT

endl

mov eax, [uMsg]

cmp eax, WM\_DESTROY

je .WMDestroy

cmp eax, WM\_CREATE

je .WMCreate

cmp eax, WM\_PAINT

je .WMPaint

.Default:

invoke DefWindowProc, [hWnd], [uMsg], [wParam], [lParam]

jmp .EndProc

.WMDestroy:

invoke PostQuitMessage, 0

jmp .Default

.WMCreate:

invoke ShowCursor, 0

stdcall Game.Welcome

jmp .Default

.WMPaint:

invoke GetUpdateRect, [hWnd], 0, 0

test eax, eax

jz .EndProc

stdcall Game.Draw

lea edi, [PS]

invoke BeginPaint, [hWnd], edi

invoke BitBlt, eax, 0, 0, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom], [Buffer.hdc], 0, 0, SRCCOPY

invoke EndPaint, [hWnd], edi

jmp .ReturnZero

.ReturnZero:

xor eax, eax

.EndProc:

ret

endp

Модуль Game:

include 'units\Game.Welcome.c'

include 'units\Game.Pause.c'

include 'units\Game.GameOver.c'

include 'units\Game.Run.c'

proc Game.Init

stdcall Game.LoadImages

stdcall Game.InitGraphics

stdcall Calc.Coordinates

stdcall Random.Init

stdcall Game.LoadRecord

ret

endp

proc Game.LoadRecord

invoke CreateFile, FRECORD, GENERIC\_READ + GENERIC\_WRITE, FILE\_SHARE\_READ, 0, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, 0

mov [Record.hf], eax

invoke GetLastError

test eax, eax

jz .CreateNew

invoke ReadFile, [Record.hf], Game.Record, 4, BYTESWRITTEN, 0

jmp .EndProc

.CreateNew:

invoke WriteFile, [Record.hf], Game.Record, 4, BYTESWRITTEN, 0

.EndProc:

invoke CloseHandle, [Record.hf]

ret

endp

proc Game.SaveRecord

invoke CreateFile, FRECORD, GENERIC\_WRITE, FILE\_SHARE\_READ, 0, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, 0

mov [Record.hf], eax

invoke WriteFile, [Record.hf], Game.Record, 4, BYTESWRITTEN, 0

invoke CloseHandle, [Record.hf]

ret

endp

proc Game.LoadImages uses eax esi edi

invoke LoadImageW, 0, IMG\_BG, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Background.hbm], eax

invoke LoadImageW, 0, IMG\_ENEMY, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Enemy.hbm], eax

invoke LoadImageW, 0, IMG\_BONUS, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Bonus.hbm], eax

invoke LoadImageW, 0, IMG\_PLAYER, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Player.hbm], eax

invoke LoadImageW, 0, IMG\_BOSS, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Boss.hbm], eax

invoke LoadImageW, 0, IMG\_PBULLET, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Player.Bullet.hbm], eax

invoke LoadImageW, 0, IMG\_EBULLET, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Enemy.Bullet.hbm], eax

invoke LoadImageW, 0, IMG\_EBULLET, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Enemy.Bullet.hbm], eax

invoke LoadImageW, 0, IMG\_START, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Start.hbm], eax

add [IMG\_START + sizeof.WORD \* 7], 1

invoke LoadImageW, 0, IMG\_START, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Start.hbm + sizeof.DWORD], eax

invoke LoadImageW, 0, IMG\_RESTART, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Restart.hbm], eax

add [IMG\_RESTART + sizeof.WORD \* 7], 1

invoke LoadImageW, 0, IMG\_RESTART, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Restart.hbm + sizeof.DWORD], eax

invoke LoadImageW, 0, IMG\_CONTINUE, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Continue.hbm], eax

add [IMG\_CONTINUE + sizeof.WORD \* 7], 1

invoke LoadImageW, 0, IMG\_CONTINUE, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Continue.hbm + sizeof.DWORD], eax

invoke LoadImageW, 0, IMG\_EXIT, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Exit.hbm], eax

add [IMG\_EXIT + sizeof.WORD \* 7], 1

invoke LoadImageW, 0, IMG\_EXIT, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Exit.hbm + sizeof.DWORD], eax

invoke LoadImageW, 0, IMG\_SOUND, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Sound.hbm], eax

add [IMG\_SOUND + sizeof.WORD \* 7], 1

invoke LoadImageW, 0, IMG\_SOUND, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Sound.hbm + sizeof.DWORD], eax

add [IMG\_SOUND + sizeof.WORD \* 7], 1

invoke LoadImageW, 0, IMG\_SOUND, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Sound.hbm + sizeof.DWORD \* 2], eax

invoke LoadImageW, 0, IMG\_GO, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [GameOver.hbm], eax

invoke LoadImageW, 0, IMG\_PLAYER\_HP, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Player.hp.hbm], eax

invoke LoadImageW, 0, IMG\_ENEMY\_HP, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Enemy.hp.hbm], eax

invoke LoadImageW, 0, IMG\_METEOR, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Meteor.hbm], eax

invoke LoadImageW, 0, IMG\_BOMB, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Animation.Bomb.hbm], eax

invoke LoadImageW, 0, IMG\_BOMB1, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Animation.Bomb.hbm + sizeof.DWORD], eax

invoke LoadImageW, 0, IMG\_PROTON, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Animation.Proton.hbm], eax

invoke LoadImageW, 0, IMG\_WELCOME, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE

mov [Welcome.hbm], eax

ret

endp

proc Game.InitGraphics uses esi eax

invoke GetDC, [MainWindow.hWnd]

mov esi, eax

invoke CreateCompatibleDC, esi

mov [Buffer.hdc], eax

invoke CreateCompatibleDC, esi

mov [Memory.hdc], eax

invoke CreateCompatibleBitmap, esi, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom]

mov [Buffer.hbm], eax

invoke CreateFont, 16,0,0,0,0,FALSE,FALSE,FALSE,ANSI\_CHARSET,OUT\_RASTER\_PRECIS,CLIP\_DEFAULT\_PRECIS,DEFAULT\_QUALITY,FIXED\_PITCH+FF\_DONTCARE,NULL

mov [hfont], eax

invoke CreateSolidBrush, $000000FF

mov [Red.hbr], eax

invoke CreateSolidBrush, $00000000

mov [Black.hbr], eax

invoke CreatePen, PS\_SOLID, 1, $000000FF

mov [Red.hp], eax

invoke CreatePen, PS\_SOLID, 1, $00000000

mov [Black.hp], eax

invoke ReleaseDC, 0, esi

ret

endp

proc Game.Draw uses edi eax esi ebx

locals

bm BITMAP

endl

lea edi, [bm]

.Background:

invoke GetObject, [Background.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Buffer.hdc], [Buffer.hbm]

invoke SetStretchBltMode, [Buffer.hdc], HALFTONE

invoke SelectObject, [Memory.hdc], [Background.hbm]

mov eax, [Background.state]

imul eax, BG\_SPEED

invoke TransparentBlt, [Buffer.hdc], 0, 0, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom], [Memory.hdc], 0, eax, [bm.bmWidth], [bm.bmWidth], $00FFFFFF

cmp [Game.state], GS\_WELCOME

je .Welcome

cmp [Game.state], GS\_PAUSE

je .Pause

cmp [Game.state], GS\_GAMEOVER

je .GameOver

cmp [Game.state], GS\_RUN

je .Run

jmp .EndProc

.Welcome:

stdcall Game.Draw.Button, Start.hbm, [Welcome.Start.left], [Welcome.Start.top], [Welcome.Start.right], [Welcome.Start.bottom], [Welcome.Start.State]

stdcall Game.Draw.Button, Exit.hbm, [Welcome.Exit.left], [Welcome.Exit.top], [Welcome.Exit.right], [Welcome.Exit.bottom], [Welcome.Exit.State]

invoke GetObject, [Animation.Proton.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Animation.Proton.hbm]

mov eax, [Animation.Proton.state]

imul eax, 100

invoke TransparentBlt, [Buffer.hdc], [Animation.Proton.x], [Animation.Proton.y], [Animation.Proton.width], [Animation.Proton.height], [Memory.hdc], eax, 0, 100, [bm.bmHeight], $00000000

invoke GetObject, [Welcome.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Welcome.hbm]

invoke TransparentBlt, [Buffer.hdc], [Welcome.rect.left], [Welcome.rect.top], [Welcome.rect.right], [Welcome.rect.bottom], [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], $00000000

jmp .EndProc

.Pause:

stdcall Game.Draw.Button, Continue.hbm, [Pause.Continue.left], [Pause.Continue.top], [Pause.Continue.right], [Pause.Continue.bottom], [Pause.Continue.State]

stdcall Game.Draw.Button, Exit.hbm, [Pause.Exit.left], [Pause.Exit.top], [Pause.Exit.right], [Pause.Exit.bottom], [Pause.Exit.State]

invoke GetObject, [Animation.Proton.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Animation.Proton.hbm]

mov eax, [Animation.Proton.state]

imul eax, 100

invoke TransparentBlt, [Buffer.hdc], [Animation.Proton.x], [Animation.Proton.y], [Animation.Proton.width], [Animation.Proton.height], [Memory.hdc], eax, 0, 100, [bm.bmHeight], $00000000

jmp .EndProc

.GameOver:

invoke GetObject, [GameOver.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [GameOver.hbm]

invoke TransparentBlt, [Buffer.hdc], [GameOver.rect.left], [GameOver.rect.top], [GameOver.rect.right], [GameOver.rect.bottom], [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], $00000000

stdcall Game.Draw.Button, Restart.hbm, [GameOver.Restart.left], [GameOver.Restart.top], [GameOver.Restart.right], [GameOver.Restart.bottom], [GameOver.Restart.State]

stdcall Game.Draw.Button, Exit.hbm, [GameOver.Exit.left], [GameOver.Exit.top], [GameOver.Exit.right], [GameOver.Exit.bottom], [GameOver.Exit.State]

invoke GetObject, [Animation.Proton.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Animation.Proton.hbm]

mov eax, [Animation.Proton.state]

imul eax, 100

invoke TransparentBlt, [Buffer.hdc], [Animation.Proton.x], [Animation.Proton.y], [Animation.Proton.width], [Animation.Proton.height], [Memory.hdc], eax, 0, 100, [bm.bmHeight], $00000000

jmp .EndProc

.Run:

.player:

invoke GetObject, [Player.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Player.hbm]

mov eax, [Player.state]

imul eax, 47

invoke TransparentBlt, [Buffer.hdc], [Player.x], [Player.y], [Player.width], [Player.height], [Memory.hdc], eax, 0, 47, [bm.bmHeight], $00000000

;player hp

invoke SelectObject, [Memory.hdc], [Player.hp.hbm]

invoke GetObject, [Player.hp.hbm], sizeof.BITMAP, edi

mov eax, [Player.hp.ProgressBar.width]

imul eax, [Player.hp]

xor edx, edx

mov esi, PLAYER\_HP\_MAX

div esi

invoke StretchBlt, [Buffer.hdc], [Player.hp.ProgressBar.x], [Player.hp.ProgressBar.y], eax, [Player.hp.ProgressBar.height], [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], SRCCOPY

invoke GetObject,[Player.Bullet.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Player.Bullet.hbm]

xor esi, esi

.bullet:

invoke TransparentBlt, [Buffer.hdc], [Player.Bullet.x + esi \* sizeof.DWORD], [Player.Bullet.y + esi \* sizeof.DWORD], [Player.Bullet.width], [Player.Bullet.height], [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], $00000000

add esi, 1

cmp esi, PBULLET\_MAX

jb .bullet

invoke GetObject, [Enemy.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Enemy.hbm]

xor esi, esi

.enemy:

mov eax, [Enemy.type + esi \* sizeof.DWORD]

imul eax, 512

mov ebx, [Enemy.state + esi \* sizeof.DWORD]

imul ebx, 512

invoke TransparentBlt, [Buffer.hdc], [Enemy.x + esi \* sizeof.DWORD], [Enemy.y + esi \* sizeof.DWORD], [Enemy.width], [Enemy.height], [Memory.hdc], ebx, eax, 512, 512, $00000000

add esi, 1

cmp esi, [Enemy.count]

jb .enemy

;enemy hp

invoke SelectObject, [Memory.hdc], [Enemy.hp.hbm]

invoke GetObject, [Enemy.hp.hbm], sizeof.BITMAP, edi

xor esi, esi

.enemy\_hp:

cmp [Enemy.activity + esi \* sizeof.DWORD], ENEMY\_ACTIVE

jne .ehp\_continue

mov eax, [Enemy.width]

imul eax, [Enemy.hp + esi \* sizeof.DWORD]

xor edx, edx

div [Enemy.starthp + esi \* sizeof.DWORD]

invoke StretchBlt, [Buffer.hdc], [Enemy.x + esi \* sizeof.DWORD], [Enemy.y + esi \* sizeof.DWORD], eax, 5, [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], SRCCOPY

.ehp\_continue:

add esi, 1

cmp esi, [Enemy.count]

jb .enemy\_hp

invoke GetObject, [Enemy.Bullet.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Enemy.Bullet.hbm]

xor esi, esi

.enemy\_bullet:

mov eax, [Enemy.type + esi \* sizeof.DWORD]

imul eax, 32

invoke TransparentBlt, [Buffer.hdc], [Enemy.Bullet.x + esi \* sizeof.DWORD], [Enemy.Bullet.y + esi \* sizeof.DWORD], [Enemy.Bullet.width], [Enemy.Bullet.height], [Memory.hdc], eax, 0, 32, [bm.bmHeight], $00000000

add esi, 1

cmp esi, [Enemy.count]

jb .enemy\_bullet

;meteor

invoke GetObject, [Meteor.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Meteor.hbm]

invoke TransparentBlt, [Buffer.hdc], [Meteor.x], [Meteor.y], [Meteor.width], [Meteor.height], [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], $00000000

;bonus

invoke GetObject, [Bonus.hbm], sizeof.BITMAP, edi

invoke SelectObject, [Memory.hdc], [Bonus.hbm]

mov esi, [Bonus.state]

imul esi, 32

mov eax, [Bonus.type]

imul eax, 32

invoke TransparentBlt, [Buffer.hdc], [Bonus.x], [Bonus.y], [Bonus.width], [Bonus.height], [Memory.hdc], esi, eax, 32, 32, $00000000

;score

invoke SetBkColor, [Buffer.hdc], TRANSPARENT

invoke SetTextColor,[Buffer.hdc], $00FFFFFF

stdcall UIntToStr, SCORE + 7 \* 2, [Game.Score]

stdcall UIntToStr, RECORD + 8 \* 2, [Game.Record]

invoke TextOut, [Buffer.hdc], 0, 0, SCORE, 15

invoke TextOut, [Buffer.hdc], 0, 30, RECORD, 15

;enemy wave indicator

invoke TextOut, [Buffer.hdc], [Enemy.WaveIndicator.x], [Enemy.WaveIndicator.y], E\_WAVE, 11

mov [E\_WAVE\_N], ' '

mov [E\_WAVE\_N + 2], ' '

stdcall UIntToStr, E\_WAVE\_N, [Game.WaveCount]

invoke TextOut, [Buffer.hdc], [Enemy.WaveIndicator.x + 4], [Enemy.WaveIndicator.y], E\_WAVE\_N, 2

;player hp indicator

invoke TextOut, [Buffer.hdc], [Player.hpIndicator.x], [Player.hpIndicator.y], HP, 15

mov [HP\_NOW], ' '

mov [HP\_NOW + 2], ' '

mov [HP\_NOW + 4], ' '

stdcall UIntToStr, HP\_NOW, [Player.hp]

invoke TextOut, [Buffer.hdc], [Player.hpIndicator.x + 4], [Player.hpIndicator.y], HP\_NOW, 3

invoke TextOut, [Buffer.hdc], [Player.hpIndicator.x + 4 \* 2], [Player.hpIndicator.y], HP\_MAX, 5

;bomb

invoke GetObject, [Animation.Bomb.hbm], sizeof.BITMAP, edi

xor esi, esi

.bomb:

mov eax, [Animation.Bomb.IsActive + esi \* sizeof.DWORD]

test eax, eax

jz .bomb\_continue

mov ecx, [Animation.Bomb.type + esi \* sizeof.DWORD]

invoke SelectObject, [Memory.hdc], [Animation.Bomb.hbm + ecx \* sizeof.DWORD]

mov eax, [Animation.Bomb.state + esi \* sizeof.DWORD]

xor edx, edx

imul eax, 150

invoke TransparentBlt, [Buffer.hdc], [Animation.Bomb.x + esi \* sizeof.DWORD], [Animation.Bomb.y + esi \* sizeof.DWORD], [Animation.Bomb.width], [Animation.Bomb.height], [Memory.hdc], eax, 0, 150, 150, $00000000

.bomb\_continue:

add esi, 1

cmp esi, ANIMATION\_BOMB\_MAX

jb .bomb

.EndProc:

ret

endp

proc Game.Draw.Button uses edi esi eax,\

button.hbm, x, y, width, height, state

locals

bm BITMAP

endl

lea edi, [bm]

mov esi, [button.hbm]

invoke GetObject, [esi], sizeof.BITMAP, edi

mov eax, [state]

invoke SelectObject, [Memory.hdc], [esi + eax \* 4]

invoke TransparentBlt, [Buffer.hdc], [x], [y], [width], [height], [Memory.hdc], 0, 0, [bm.bmWidth], [bm.bmHeight], $00000000

ret

endp

proc Game.Background.Update uses eax esi

mov eax, [Background.state]

add eax, 1

mov esi, BG\_STATE\_COUNT

div esi

mov [Background.state], edx

ret

endp

Модуль Game.Run:

proc Game.Run

mov [Game.state], GS\_RUN

stdcall Calc.Coordinates

stdcall Game.InitEnemy

stdcall Game.InitBullet

stdcall Game.InitPlayer

stdcall Game.InitScore

invoke GetTickCount

mov [Timer.EWave.TickCount], eax

mov [Timer.Bullet.TickCount], eax

mov [Timer.Meteor.TickCount], eax

mov [Timer.Bonus.TickCount], eax

mov [Timer.Score.TickCount], eax

mov [Timer.Animation.TickCount], eax

mov [Timer.Update.TickCount], eax

ret

endp

proc Game.InitPlayer

mov [Player.hp], PLAYER\_HP

mov [Player.state], 0

ret

endp

proc Game.InitScore uses ecx

mov [Game.Score], 0

mov ecx, 7

.loop:

mov [SCORE + ecx \* 2], ' '

add ecx, 1

cmp ecx, 15

jne .loop

ret

endp

proc Game.InitEnemy uses ecx edi esi

mov [Enemy.count], 1

mov [Game.WaveCount], 0

mov edi, [MainWindow.ClientRect.right]

mov esi, [MainWindow.ClientRect.bottom]

xor ecx, ecx

.init\_enemy:

mov [Enemy.x + ecx \* sizeof.DWORD], edi

mov [Enemy.Bullet.x + ecx \* sizeof.DWORD], edi

mov [Enemy.y + ecx \* sizeof.DWORD], esi ; --

mov [Enemy.Bullet.y + ecx \* sizeof.DWORD], esi ; --

mov [Enemy.activity + ecx \* sizeof.DWORD], ENEMY\_UNUSED ; \*

mov [Enemy.ymax + ecx \* sizeof.DWORD], ENEMY\_DIST

mov [Enemy.hp + ecx \* sizeof.DWORD], 0

mov [Enemy.state + ecx \* sizeof.DWORD], 0

add ecx, 1

cmp ecx, ENEMY\_MAX

jb .init\_enemy

ret

endp

proc Game.InitBullet uses edi esi ecx

mov [Player.Bullet.state], PBULLET\_STATE\_1

mov [Player.Bullet.need], 0

mov edi, [MainWindow.ClientRect.right]

mov esi, [MainWindow.ClientRect.bottom]

;meteor

mov [Meteor.x], edi

mov [Meteor.y], esi

;bonus

mov [Bonus.x], edi

mov [Bonus.y], esi

mov [Bonus.hp], 1

xor ecx, ecx

.init\_bullet:

mov [Player.Bullet.x + ecx \* sizeof.DWORD], edi ; --\*

mov [Player.Bullet.y + ecx \* sizeof.DWORD], 0 ; --

add ecx, 1

cmp ecx, PBULLET\_MAX

jb .init\_bullet

xor ecx, ecx

.init\_bomb:

mov [Animation.Bomb.IsActive + ecx \* sizeof.DWORD], 0

mov [Animation.Bomb.x + ecx \* sizeof.DWORD], edi

mov [Animation.Bomb.y + ecx \* sizeof.DWORD], esi

mov [Animation.Bomb.state + ecx \* sizeof.DWORD], 0

mov [Animation.Bomb.type + ecx \* sizeof.DWORD], 0

add ecx, 1

cmp ecx, ANIMATION\_BOMB\_MAX

jb .init\_bomb

ret

endp

proc Game.Run.Update uses eax esi,\

TickCount

mov esi, [TickCount]

mov eax, [Timer.Bonus.TickCount]

add eax, [TIMER\_BONUS]

cmp eax, esi

jb .bonus

mov eax, [Timer.Meteor.TickCount]

add eax, [TIMER\_METEOR]

cmp eax, esi

jb .meteor

mov eax, [Timer.Score.TickCount]

add eax, [TIMER\_SCORE]

cmp eax, esi

jb .score

mov eax, [Timer.EWave.TickCount]

add eax, [TIMER\_EWAVE]

cmp eax, esi

jb .ewave

mov eax, [Timer.Animation.TickCount]

add eax, [TIMER\_ANIMATION]

cmp eax, esi

jb .animation

mov eax, [Timer.Bullet.TickCount]

add eax, [TIMER\_BULLET]

cmp eax, esi

jb .bullet

.update:

stdcall Game.Run.Update.ProcessInput

stdcall Game.Run.Update.Bullet

stdcall Game.Run.Update.Enemy

stdcall Game.Run.Update.EnemyCrash

stdcall Game.Run.Update.PlayerCrash

stdcall Game.Run.Update.Meteor

stdcall Game.Run.Update.Bonus

stdcall Game.Run.Update.Record

invoke InvalidateRect, [MainWindow.hWnd], 0, 0

jmp .EndProc

.bonus:

stdcall Game.Run.Bonus

mov [Timer.Bonus.TickCount], esi

jmp .EndProc

.meteor:

stdcall Game.Run.Meteor

mov [Timer.Meteor.TickCount], esi

jmp .EndProc

.score:

add [Game.Score], SCORE\_TIME

mov [Timer.Score.TickCount], esi

jmp .EndProc

.ewave:

stdcall Game.Run.CheckEnemy

mov [Timer.EWave.TickCount], esi

jmp .EndProc

.bullet:

stdcall Game.Run.Bullet

mov [Timer.Bullet.TickCount], esi

jmp .EndProc

.animation:

stdcall Game.Run.Update.Animation

stdcall Game.Run.Update.PlayerState

stdcall Game.Run.Update.EnemyState

mov [Timer.Animation.TickCount], esi

jmp .EndProc

.EndProc:

ret

endp

proc Game.Run.Update.ProcessInput uses edi eax edx

mov edi, PLAYER\_DIST

invoke GetKeyState, VK\_LEFT

and eax, $8000

shr eax, 15

mul edi

sub [Player.x], eax

stdcall Game.CheckCoordinates, [Player.x], [Player.y], 0, 0, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom]

xor eax, 1

mul edi

add [Player.x], eax

invoke GetKeyState, VK\_RIGHT

and eax, $8000

shr eax, 15

mul edi

add [Player.x], eax

mov eax, [Player.x]

add eax, [Player.width]

stdcall Game.CheckCoordinates, eax, [Player.y], 0, 0, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom]

xor eax, 1

mul edi

sub [Player.x], eax

invoke GetKeyState, VK\_DOWN

and eax, $8000

shr eax, 15

mul edi

add [Player.y], eax

mov eax, [Player.y]

add eax, [Player.height]

stdcall Game.CheckCoordinates, [Player.x], eax, 0, 0, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom]

xor eax, 1

mul edi

sub [Player.y], eax

invoke GetKeyState, VK\_UP

and eax, $8000

shr eax, 15

mul edi

sub [Player.y], eax

stdcall Game.CheckCoordinates, [Player.x], [Player.y], 0, 0, [MainWindow.ClientRect.right], [MainWindow.ClientRect.bottom]

xor eax, 1

mul edi

add [Player.y], eax

invoke GetKeyState, VK\_SPACE

and eax, $8000

jz .no\_bullet

mov [Player.Bullet.need], 1

.no\_bullet:

invoke GetKeyState, VK\_ESCAPE

and eax, $8000

jz .no\_pause

stdcall Game.Pause

.no\_pause:

ret

endp

proc Game.Run.Update.PlayerState uses eax edx ebx

mov eax, [Player.state]

add eax, 1

mov ebx, PLAYER\_STATE\_MAX

xor edx, edx

div ebx

mov [Player.state], edx

ret

endp

proc Game.Run.Update.EnemyState uses ecx eax edx esi

xor ecx, ecx

.update\_enemy:

mov eax, [Enemy.state + ecx \* sizeof.DWORD]

add eax, 1

mov esi, ENEMY\_STATE\_MAX

xor edx, edx

div esi

mov [Enemy.state + ecx \* sizeof.DWORD], edx

add ecx, 1

cmp ecx, [Enemy.count]

jb .update\_enemy

ret

endp

proc Game.Run.Update.Record

mov eax, [Game.Score]

cmp eax, [Game.Record]

jb .EndProc

mov [Game.Record], eax

.EndProc:

ret

endp

proc Game.Run.Update.Enemy uses ecx eax esi

xor ecx, ecx

.update\_enemy:

stdcall Calc.AddMod, [Enemy.y + ecx \* sizeof.DWORD], ENEMY\_DIST, [Enemy.ymax + ecx \* sizeof.DWORD]

mov [Enemy.y + ecx \* sizeof.DWORD], eax

add ecx, 1

cmp ecx, [Enemy.count]

jb .update\_enemy

stdcall Game.Run.Update.EBullet

ret

endp

proc Game.Run.Update.EnemyCrash uses esi edi

xor esi, esi

.enemy:

xor edi, edi

.bullet:

cmp [Enemy.activity + esi \* sizeof.DWORD], ENEMY\_ACTIVE

jne .continue

mov eax, [Enemy.height]

sub eax, 50

stdcall Game.CheckCoordinates, [Player.Bullet.x + edi \* sizeof.DWORD], [Player.Bullet.y + edi \* sizeof.DWORD], [Enemy.x + esi \* sizeof.DWORD], [Enemy.y + esi \* sizeof.DWORD], [Enemy.width], eax

test eax, eax

jz .continue

;hide bullet

mov eax, [Player.Bullet.y + edi \* sizeof.DWORD]

sub eax, 35

stdcall Game.Run.Animation.Bomb, [Player.Bullet.x + edi \* sizeof.DWORD], eax, 50, 50, 0

mov ebx, [MainWindow.ClientRect.right]

mov [Player.Bullet.x + edi \* sizeof.DWORD], ebx

mov [Player.Bullet.y + edi \* sizeof.DWORD], 0

;sub enemy hp

add [Game.Score], SCORE\_DAMAGE

sub [Enemy.hp + esi \* sizeof.DWORD], 1

jnz .continue

;enemy die

add [Game.Score], SCORE\_EDIE

mov [Enemy.activity + esi \* sizeof.DWORD], ENEMY\_UNUSED

mov [Enemy.Bullet.x + + esi \* sizeof.DWORD], ebx

mov [Enemy.x + esi \* sizeof.DWORD], ebx

.continue:

add edi, 1

cmp edi, PBULLET\_MAX

jb .bullet

add esi, 1

cmp esi, [Enemy.count]

jb .enemy

ret

endp

proc Game.Run.Animation.Bomb uses esi edi eax,\

x, y, width, height, type

mov edi, [x]

cmp edi, [MainWindow.ClientRect.right]

jns .EndProc

xor esi, esi

.find\_unused\_animation:

mov eax, [Animation.Bomb.IsActive + esi \* sizeof.DWORD]

test eax, eax

jnz .continue

mov [Animation.Bomb.IsActive + esi \* sizeof.DWORD], 1

mov edi, [x]

mov [Animation.Bomb.x + esi \* sizeof.DWORD], edi

mov edi, [y]

mov [Animation.Bomb.y + esi \* sizeof.DWORD], edi

mov edi, [width]

mov [Animation.Bomb.width + esi \* sizeof.DWORD], edi

mov edi, [height]

mov [Animation.Bomb.height + esi \* sizeof.DWORD], edi

mov edi, [type]

mov [Animation.Bomb.type + esi \* sizeof.DWORD], edi

mov [Animation.Bomb.state + esi \* sizeof.DWORD], 0

jmp .EndProc

.continue:

add esi, 1

cmp esi, ANIMATION\_BOMB\_MAX

jb .find\_unused\_animation

.EndProc:

ret

endp

proc Game.Run.Update.Animation uses esi eax

xor esi, esi

.update\_animation:

mov eax, [Animation.Bomb.IsActive + esi \* sizeof.DWORD]

test eax, eax

jz .continue

add [Animation.Bomb.state + esi \* sizeof.DWORD], 1

cmp [Animation.Bomb.state + esi \* sizeof.DWORD], ANIMATION\_BOMB\_STATE\_MAX

jb .continue

mov [Animation.Bomb.IsActive + esi \* sizeof.DWORD], 0

.continue:

add esi, 1

cmp esi, ANIMATION\_BOMB\_MAX

jb .update\_animation

ret

endp

proc Game.Run.Update.PlayerCrash uses esi edi ecx eax

;player <-> enemy bullet

xor esi, esi

.enemy\_bullet:

stdcall Game.CheckCoordinates, [Enemy.Bullet.x + esi \* sizeof.DWORD], [Enemy.Bullet.y + esi \* sizeof.DWORD], [Player.x], [Player.y], [Player.width], [Player.height]

test eax, eax

jz .continue

mov eax, [Enemy.Bullet.y + esi \* sizeof.DWORD]

add eax, 15

stdcall Game.Run.Animation.Bomb, [Enemy.Bullet.x + esi \* sizeof.DWORD], eax, 45, 45, 1

;hide bullet

mov ebx, [MainWindow.ClientRect.right]

mov [Enemy.Bullet.x + esi \* sizeof.DWORD], ebx

;sub player hp

mov eax, [Enemy.Bullet.hp + esi \* sizeof.DWORD]

sub [Player.hp], eax

mov eax, [Player.hp]

test eax, eax

js .gameover

.continue:

add esi, 1

cmp esi, [Enemy.count]

jb .enemy\_bullet

;meteor <-> player

mov esi, [Meteor.width]

shr esi, 1

add esi, [Meteor.x]

mov edi, [Meteor.y]

add edi, [Meteor.height]

stdcall Game.CheckCoordinates, esi, edi, [Player.x], [Player.y], [Player.width], [Player.height]

test eax, eax

jz .no\_meteor\_crash

add edi, 15

stdcall Game.Run.Animation.Bomb, esi, edi, 60, 60, 1

mov ebx, [MainWindow.ClientRect.right]

mov [Meteor.x], ebx

mov eax, [Meteor.damage]

sub [Player.hp], eax

mov eax, [Player.hp]

test eax, eax

js .gameover

.no\_meteor\_crash:

;bonus <-> player

mov esi, [Bonus.width]

shr esi, 1

add esi, [Bonus.x]

mov edi, [Bonus.y]

add edi, [Bonus.height]

stdcall Game.CheckCoordinates, esi, edi, [Player.x], [Player.y], [Player.width], [Player.height]

test eax, eax

jz .no\_bonus

add [Game.Score], SCORE\_BONUS

mov edi, [MainWindow.ClientRect.right]

mov [Bonus.x], edi

mov eax, [Player.hp]

add eax, [Bonus.hp]

mov esi, PLAYER\_HP\_MAX

add esi, 1

xor edx, edx

div esi

test eax, eax

jz .no\_overflow

mov [Player.hp], PLAYER\_HP\_MAX

jmp .no\_bonus

.no\_overflow:

mov [Player.hp], edx

.no\_bonus:

;enemy <-> player

xor ecx, ecx

.loop\_player\_enemy:

mov esi, [Enemy.y + ecx \* sizeof.DWORD]

add esi, [Enemy.height + ecx \* sizeof.DWORD]

mov edi, [Player.y]

add edi, 10

cmp esi, edi

jb .next\_iteration

mov edi, [Player.y]

add edi, [Player.height]

sub edi, 10

mov esi, [Enemy.y + ecx \* sizeof.DWORD]

cmp esi, edi

ja .next\_iteration

mov esi, [Enemy.x + ecx \* sizeof.DWORD]

add esi, [Enemy.width + ecx \* sizeof.DWORD]

mov edi, [Player.x]

add edi, 10

cmp esi, edi

jb .next\_iteration

mov edi, [Player.x]

add edi, [Player.width]

sub edi, 10

mov esi, [Enemy.x + ecx \* sizeof.DWORD]

cmp esi, edi

ja .next\_iteration

stdcall Game.GameOver

jmp .EndProc

.next\_iteration:

add ecx, 1

cmp ecx, [Enemy.count]

jb .loop\_player\_enemy

jmp .EndProc

.gameover:

stdcall Game.GameOver

.EndProc:

ret

endp

proc Game.Run.CheckEnemy uses ecx esi

xor esi, esi

xor ecx, ecx

.check\_enemy:

cmp [Enemy.activity + ecx \* sizeof.DWORD], ENEMY\_ACTIVE

je .EndProc

add ecx, 1

cmp ecx, [Enemy.count]

jb .check\_enemy

stdcall Game.Run.EWave

.EndProc:

ret

endp

proc Game.Run.EWave uses eax ebx ecx edx edi esi

mov edi, [MainWindow.ClientRect.right]

sub edi, [Enemy.width]

xor ecx, ecx

.ewave:

;x

.choose\_x:

stdcall Random, 0, edi

mov edx, eax

xor ebx, ebx

.cmp\_equal:

mov eax, edx

add eax, [Enemy.width]

cmp eax, [Enemy.x + ebx \* sizeof.DWORD]

jb .cmp\_continue

mov eax, [Enemy.x + ebx \* sizeof.DWORD]

add eax, [Enemy.width]

cmp eax, edx

jb .cmp\_continue

jmp .choose\_x

.cmp\_continue:

add ebx, 1

cmp ebx, ecx

jb .cmp\_equal

mov [Enemy.x + ecx \* sizeof.DWORD], edx

;y

mov [Enemy.y + ecx \* sizeof.DWORD], 0

;ymax

stdcall Random, ENEMY\_DIST, [ENEMY\_YMAX]

mov [Enemy.ymax + ecx \* sizeof.DWORD], eax

;hp

stdcall Random, ENEMY\_HP\_MIN, ENEMY\_HP\_MAX

mov [Enemy.hp + ecx \* sizeof.DWORD], eax

mov [Enemy.starthp + ecx \* sizeof.DWORD], eax

;type

stdcall Random, 0, ENEMY\_TYPE\_COUNT

mov [Enemy.type + ecx \* sizeof.DWORD], eax

mov [Enemy.Bullet.hp + ecx \* sizeof.DWORD], eax

add [Enemy.Bullet.hp + ecx \* sizeof.DWORD], 1

mov [Enemy.state + ecx \* sizeof.DWORD], 0

;make active

mov [Enemy.activity + ecx \* sizeof.DWORD], ENEMY\_ACTIVE

;bullet

stdcall Game.Run.EnemyBullet, ecx

add ecx, 1

cmp ecx, [Enemy.count]

jb .ewave

stdcall Calc.AddMod, [Enemy.count], 1, ENEMY\_MAX

mov [Enemy.count], eax

;increase bulet count every 4 wave

add [Game.WaveCount], 1

mov eax, [Game.WaveCount]

xor edx, edx

mov esi, 4

div esi

test edx, edx

jnz .no\_bullet\_increase

stdcall Calc.AddMod, [Player.Bullet.state], 1, PBULLET\_STATE\_COUNT

mov [Player.Bullet.state], eax

.no\_bullet\_increase:

ret

endp

proc Game.Run.EnemyBullet uses eax ecx,\

num

mov ecx, [num]

stdcall Calc.GetCenterCoord, [Enemy.width], [Enemy.Bullet.width]

add eax, [Enemy.x + ecx \* sizeof.DWORD]

mov [Enemy.Bullet.x + ecx \* sizeof.DWORD], eax

mov eax, [Enemy.y + ecx \* sizeof.DWORD]

add eax, [Enemy.height]

mov [Enemy.Bullet.y + ecx \* sizeof.DWORD], eax

ret

endp

proc Game.Run.Update.EBullet uses ecx esi eax

mov esi, [MainWindow.ClientRect.bottom]

xor ecx, ecx

.update\_ebullet:

cmp [Enemy.activity + ecx \* sizeof.DWORD], ENEMY\_ACTIVE

jne .continue

mov eax, [Enemy.y + ecx \* sizeof.DWORD]

add eax, [Enemy.height]

cmp eax , esi

ja .continue

add [Enemy.Bullet.y + ecx \* sizeof.DWORD], EBULLET\_DIST

cmp [Enemy.Bullet.y + ecx \* sizeof.DWORD], esi

jb .continue

stdcall Game.Run.EnemyBullet, ecx

.continue:

add ecx, 1

cmp ecx, [Enemy.count]

jb .update\_ebullet

ret

endp

proc Game.Run.Bullet uses edi esi ecx ebx eax edx

cmp [Player.Bullet.need], 1

jne .EndProc

mov esi, [Player.Bullet.state]

add esi, 1

mov eax, [Player.width]

xor edx, edx

div esi

mov esi ,eax

mov eax, [Player.Bullet.width]

shr eax, 1

sub esi, eax

mov ebx, esi

mov edi, [MainWindow.ClientRect.right]

xor ecx, ecx

xor esi, esi

.create:

cmp [Player.Bullet.x + sizeof.DWORD \* ecx], edi

je .createnew

jmp .continue

.createnew:

add esi, 1

mov eax, ebx

xor edx, edx

mul esi

add eax, [Player.x]

mov [Player.Bullet.x + sizeof.DWORD \* ecx], eax

mov eax, [Player.y]

mov [Player.Bullet.y + sizeof.DWORD \* ecx], eax

.continue:

add ecx, 1

cmp esi, [Player.Bullet.state]

jnz .create

mov [Player.Bullet.need], 0

invoke PlaySound, SND\_BULLET, 0, SND\_ASYNC or SND\_NODEFAULT

.EndProc:

ret

endp

proc Game.Run.Update.Bullet uses edi ecx

mov edi, [MainWindow.ClientRect.right]

xor ecx, ecx

.update\_pbullet:

sub [Player.Bullet.y + ecx \* sizeof.DWORD], PBULLET\_DIST

cmp [Player.Bullet.y + ecx \* sizeof.DWORD], 0

jns .next\_iteration

mov [Player.Bullet.x + ecx \* sizeof.DWORD], edi

mov [Player.Bullet.y + ecx \* sizeof.DWORD], 0

.next\_iteration:

add ecx, 1

cmp ecx, PBULLET\_MAX

jnz .update\_pbullet

ret

endp

proc Game.Run.Meteor uses esi edi eax edx

mov edi, [Player.x]

mov [Meteor.x], edi

mov [Meteor.y], 0

stdcall Random, METEOR\_DAMAGE\_MIN, METEOR\_DAMAGE\_MAX

mov [Meteor.damage], eax

imul eax, 3

mov [Meteor.width], eax

mov [Meteor.height], eax

ret

endp

proc Game.Run.Update.Meteor

add [Meteor.y], METEOR\_DIST

ret

endp

proc Game.Run.Bonus uses eax edi

mov edi, [MainWindow.ClientRect.right]

sub edi, [Bonus.width]

stdcall Random, 1, edi

mov [Bonus.x], eax

mov [Bonus.y], 0

stdcall Calc.AddMod, [Bonus.hp], 1, BONUS\_HP\_MAX

mov [Bonus.hp], eax

mov [Bonus.state], 0

stdcall Random, 1, ENEMY\_TYPE\_COUNT

sub eax, 1

mov [Bonus.type], eax

ret

endp

Модуль Game.Welcome:

proc Game.Welcome uses eax

mov [Game.state], GS\_WELCOME

stdcall Game.Init

stdcall Animation.Proton

invoke GetTickCount

mov [Timer.Update.TickCount], eax

mov [Timer.Background.TickCount], eax

invoke PlaySound, SND\_START, 0, SND\_ASYNC or SND\_NODEFAULT

ret

endp

proc Game.Welcome.Update uses eax esi ,\

TickCount

mov esi, [TickCount]

mov eax, [Timer.Update.TickCount]

add eax, [TIMER\_UPDATE]

cmp eax, esi

ja .EndProc

stdcall Game.Welcome.Update.ProcessInput

stdcall Animation.Proton.Update

mov [Timer.Update.TickCount], esi

invoke InvalidateRect, [MainWindow.hWnd], 0, 0

.EndProc:

ret

endp

proc Animation.Proton

mov [Animation.Proton.state], 0

ret

endp

proc Animation.Proton.Update uses eax edx esi edi

locals

Cursor\_Pos POINT

endl

;get cursor position

lea edi, [Cursor\_Pos]

invoke GetCursorPos, edi

mov esi, [Cursor\_Pos.x]

mov eax, [Animation.Proton.width]

shr eax, 1

sub esi, eax

mov [Animation.Proton.x], esi

mov esi, [Cursor\_Pos.y]

mov eax, [Animation.Proton.width]

shr eax, 1

sub esi, eax

mov [Animation.Proton.y], esi

mov eax, [Animation.Proton.state]

add eax, 1

mov esi, ANIMATION\_PROTON\_STATE\_MAX

xor edx, edx

div esi

mov [Animation.Proton.state], edx

ret

endp

proc Game.Welcome.Update.ProcessInput uses eax esi edi

locals

Cursor\_Pos POINT

endl

;check if the left mouse button is down

invoke GetKeyState, VK\_LBUTTON

and eax, $8000

shr eax, 15

mov esi, eax

;get cursor position

lea edi, [Cursor\_Pos]

invoke GetCursorPos, edi

;check if cursor is under a button

stdcall Game.CheckCoordinates, [Cursor\_Pos.x], [Cursor\_Pos.y], [Welcome.Start.left], [Welcome.Start.top], [Welcome.Start.right], [Welcome.Start.bottom]

mov [Welcome.Start.State], eax

test eax, esi ;if cursor is under button (eax) and is button is down(esi) - do something

jnz .GS\_Run

stdcall Game.CheckCoordinates, [Cursor\_Pos.x], [Cursor\_Pos.y], [Welcome.Exit.left], [Welcome.Exit.top], [Welcome.Exit.right], [Welcome.Exit.bottom]

mov [Welcome.Exit.State], eax

test eax, esi

jnz .Exit

stdcall Game.CheckCoordinates, [Cursor\_Pos.x], [Cursor\_Pos.y], [Sound.left], [Sound.top], [Sound.right], [Sound.bottom]

mov [Sound.State], eax

test eax, esi

jnz .SoundStateChange

jmp .EndProc

.GS\_Run:

stdcall Game.Run

jmp .EndProc

.Exit:

invoke SendMessage, [MainWindow.hWnd], WM\_DESTROY, 0, 0

jmp .EndProc

.SoundStateChange:

;stdcall Sound.ChangeState

.EndProc:

ret

endp

Модуль Game.Calculate:

proc Random.Init uses eax

invoke GetTickCount

mov [Random.wPrevNumber], eax

ret

endp

proc Random uses edx ecx,\

nMinValue, nMaxValue

xor edx, edx

mov ecx, [nMaxValue]

sub ecx, [nMinValue]

inc ecx

invoke GetTickCount

add eax, [Random.wPrevNumber]

rol ax, 7

add ax, 23

mov [Random.wPrevNumber], eax

div ecx

mov eax, edx

add eax, [nMinValue]

ret

endp

proc Game.CheckCoordinates uses edi esi ebx edx,\

x, y, left, top, width, height

mov esi, [x]

mov edi, [y]

mov ebx, [left]

mov edx, [top]

cmp esi, ebx

js .Return\_FALSE

cmp edi, edx

js .Return\_FALSE

add ebx, [width]

cmp ebx, esi

js .Return\_FALSE

add edx, [height]

cmp edx, edi

js .Return\_FALSE

.Return\_TRUE:

mov eax, 1

jmp .EndProc

.Return\_FALSE:

xor eax, eax

.EndProc:

ret

endp

proc Calc.RelativeValue uses edx ebx ecx,\

Value, DivValue, MulValue

mov eax, [Value]

mov ebx, [DivValue]

mov ecx, [MulValue]

xor edx, edx

div ebx

mul ecx

ret

endp

proc Calc.GetCenterCoord uses ebx,\

WidthMain, WidthObj

mov ebx, [WidthObj]

shr ebx, 1

mov eax, [WidthMain]

shr eax, 1

sub eax, ebx

ret

endp

proc Calc.AddMod uses edx,\

x, y, p

mov eax, [x]

add eax, [y]

xor edx, edx

div [p]

cmp eax, 1

je .nochange

mov eax, edx

jmp .EndProc

.nochange:

mov eax, [x]

.EndProc:

ret

endp