Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Игровое приложение «Змейка»

БГУИР КП 1–40 01 01 326 ПЗ

Выполнил

студент гр. 951003 Туромша В.А.

Проверил: Яскевич Д.А.

Минск 2021

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2021г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Туромше Владимиру Александровичу*

1. Тема работы *Игровое приложение «Змейка»* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *11.12.2021г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Тип операционной системы – ОС Windows 10; Язык программирования – C++.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Обзор аналогов\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Постановка задачи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Моделирование предметной области\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*4 Разработка программного средства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*5 Тестирование и проверка работоспособности прогрпммного средства \_\_\_*

*6 Руководство по установке и использованию программного средства\_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе *Яскевич Д.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *16.09.2021г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

*Раздел 1. Введение к 28.09.2021г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 15.10.2021г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 25.10.2021г. – 45% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 15.11.2021г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 5 к 25.11.2021г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 6.Заключение. Приложения к 01.12.2021г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 08.12.2021г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 01.12.2021г. по 11.12.2021г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Яскевич Д.А.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Туромша В.А. 16.09.2021г.*

*(дата и подпись студента)*

Содержание

[Введение 6](#_Toc90416264)

[1 Обзор аналогов 7](#_Toc90416265)

[1.1 Анализ существующих аналогов 7](#_Toc90416266)

[2 Постановка задачи 10](#_Toc90416267)

[3 Моделирование предметной области 11](#_Toc90416268)

[4 Разработка программного средства 12](#_Toc90416269)

[4.1 Описание алгоритмов решения задачи 12](#_Toc90416270)

[4.2 Интерфейс программы 15](#_Toc90416271)

[4.3 Структура программного средства 16](#_Toc90416272)

[5 Тестирование и проверка работоспособности программного средства 17](#_Toc90416273)

[5.1 Завершение игры 17](#_Toc90416274)

[5.2 Движение объекта по игровому полю 19](#_Toc90416275)

[5.3 Изменение скорости движения объекта 22](#_Toc90416276)

[5.4 Проверка работоспособности меню 23](#_Toc90416277)

[6 Руководство по использованию программного средства 25](#_Toc90416278)

[6.1 Работа с приложением 25](#_Toc90416279)

[Заключение 26](#_Toc90416280)

[Список использованной литературы 27](#_Toc90416281)

[Приложение А 28](#_Toc90416282)

Введение

Даже в современном стремительном мире, где каждая секунда на вес золота, невозможно постоянно оставаться продуктивным. Часто мы ожидаем какого-либо события, сидим в очереди к какому-либо специалисту, едем в транспорте или просто решили отдохнуть. В такое время бывает неудобно заниматься делами, требующими серьезной концентрции, хочется отвлечься и расслабиться. В этом могут помочь различные игры, ведь когда мы играем, погружаемся в совершенно другой мир и примеряем роли таких персонажей, которыми в реальной жизни стать невозможно.

Изначально все игры существовали в реальном пространстве, но с появлением и развитием информационных технологий и научно-техническим прогрессом, у людей появилась возможность перенести игровой процесс в вирутальное пространство. Компьютерные игры позволили людям, которые в данный момент не имеют возможности поиграть с кем-то другим, посоперничать с виртуальным соперником, либо вообще играть в одиночном режиме. Также есть люди, которые любят делать все сами и полагаться только на себя, поэтому одиночный игровой режим позволяет им полностью погрузиться в ситуацию, где можно всем управлять самому.

На сегодняшний день существует огромное количество таких игр. Некоторые уже давно утратили свою актуальность, но есть и те, что стали нестареющей классикой. Примером является игровое приложение «Змейка». Простые и понятные правила позволяют легко освоить данную игру людям любого возраста. Постоянно усложняющийся, но также и быстрый процесс игры, является самым привлекательным аспектом, благодаря этому игра «Змейка» стала по-настоящему народной. Почти невозможно встретить человека, который никогда не слышал об этой игре. Поскольку такая игра будет интересна большому количеству людей, было принято решение ее реализовать.

Целью данного проекта является создание игрового приложения «Змейка», сопровождающегося документацией в виде пояснительной записки.

# Анализ существующих аналогов

Существует множество аналогов игры «Змейка». Все они имеют схожий игровой процесс и различаются только функционалом и игровыми режимами. Свою популярность игра получила благодаря тому, что слала одной из первых на мобильной платформе и не имела альтернатив. Сейчас в «Змейку» так же играют преимущественно на телефонах, но существуют так же десктопные и веб-версии.

## Приложение «Snake» в «Google Play»

Приложение «Snake» в «Google Play» является популярным выбором данной игры. Управление осуществляется с помощью отображенных на экране стрелок движения, которые соотвествуют ранее сущетсвовавшим кнопкам. В игре присутствует выбор режимов, которые отличаются видами игрового поля и самой змейки. Интерфейс «Snake» представлен на рисунке 1.1.

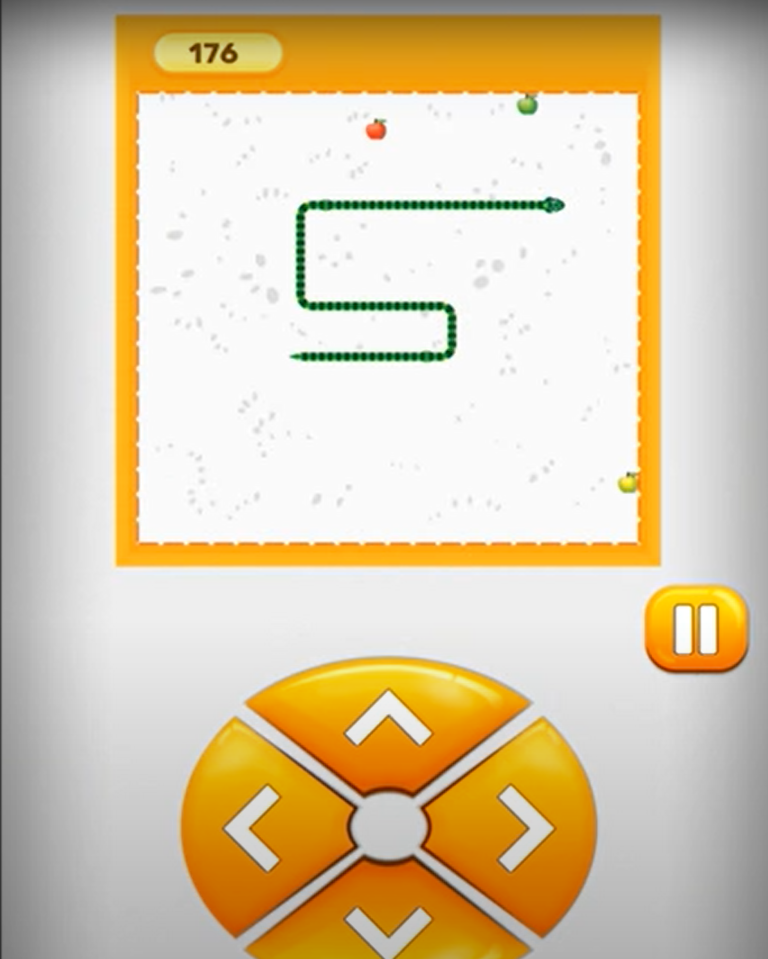


Рисунок 1.1 – Интерфейс «Snake»

Преимущества данного приложения:

* большие удобные элементы управления;
* возможность остановки игры;
* возможность выбора различных игровых полей и змей.

Недостатки данного приложения:

* отсутствие сетевого режима;
* маленькое игровое поле по сравнению с элементами управления;
* только игровое поле с границами.

## Приложение «HungrySnake»

«HungrySnake» — десктопное игровое приложение. В данном приложении реализован стандартный функционал, указаны уровни скорости движения змейки, набранный на данный момент счет и количество оставшихся жизней. Интерфейс «HungrySnake» представлен на рисунке 1.2.

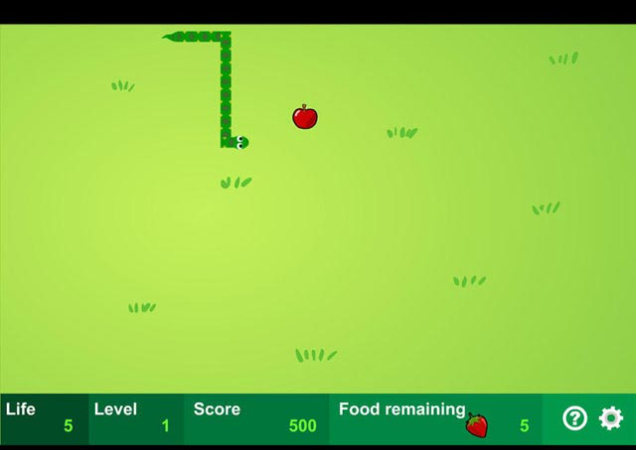


Рисунок 1.2 – Интерфейс программы «HungrySnake»

Преимущества данного приложения:

* наличие нескольких жизней змейки;
* наличие уровней скорости движения змейки;
* отображение оставшегося количества еды, которую нужно съесть до перехода на следующий уровень.

Недостатки данного приложения:

* один игровой режим;
* отсутствие кастомизации игрового поля;
* отсутствие кастомизации змейки.

## Приложение «Змейка» от «Google»

Последним рассмотрим веб-версию игрового приложения «Змейка» от «Google». Существует также мобильная версия, отличия для пользователя заключается лишь в управлении: в мобильной – свайпами, в веб-версии – клавишами. В приложении реализован стандартный функционал, игровое поле не является сквозным, во время игры пользователь может видеть набранный счет. Интерфейс «Snake» представлен на рисунке 1.3.

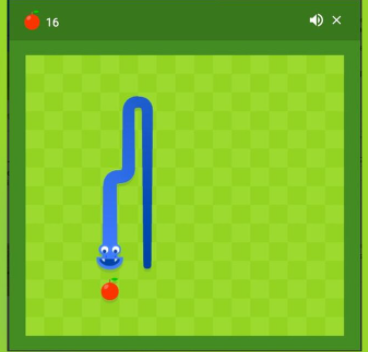


Рисунок 1.3 – Интерфейс программы «Snake»

Преимущества данного приложения:

* наличие веб-версии и мобильной версии приложения;
* плавное передвижение объекта по игровуму полю;
* форма змейки отображает форму реальной змеи;
* возможность остановки игры.

Недостатки данного приложения:

* один игровой режим;
* отсутствие кастомизации игрового поля;
* отсутствие кастомизации змейки;
* отсутствие игровых уровней.

# Постановка задачи

Целью курсового проектирования является реализация программного средства, представляющее собой игровое приложение «Змейка».

По результатам обзора аналогов было решено выдвинуть следующие требования к функционалу программного средства:

* управление объектом «змейка» посредством использования стрелок клавиатуры;
* отображение набранного счета;
* возможность перезапуска игры;
* возможность остановки игры;
* завершение игры при столкновении объекта с границей игрового поля;
* завершение игры при столкновении объекта с самим собой;
* ускорение движения объекта при достижении определенного счета;
* возможность ускорения движения объекта при помощи стрелок клавиатуры.

Разработка программного средства должна производиться на языке программирования С++ в интегрированной среде разработки Microsoft Visual Studio 2019. Интерфейс будет реализован с помощью задействования функций WinAPI.

# Моделирование предметной области

## Windows API

Программное средство должно быть реализовано с использованием WinAPI. Далее будет приведены основные понятия.

Windows API – общее наименование набора базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows корпорации Microsoft.

Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системой Windows. Работа через WinAPI – это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ.

У каждого окна, создаваемого программой, имеется соответствующая оконная процедура. Эта процедура является функцией, которая может находиться либо в самой программе, либо в динамически подключаемой библиотеке. Windows посылает сообщение окну путем вызова оконной процедуры, на основе этого сообщения окно совершает какие-то действия и затем возвращает управление Windows. Более точно, окно всегда создается на основе "класса окна". Класс окна определяет оконную процедуру, обрабатывающую поступающие окну сообщения. Использование класса окна позволяет создавать множество окон на основе одного и того же класса окна и, следовательно, использовать одну и ту же оконную процедуру. Например, все кнопки во всех программах для Windows созданы на основе одного и того же класса окна. Этот класс связан с оконной процедурой (расположенной в динамически подключаемой библиотеке Windows), которая управляет процессом передачи сообщений всем кнопкам всех окон.

Алгоритм работы простейшей программы с событийным управлением представлен на рисунке 3.1.

Оконная процедура обрабатывает сообщения, поступающие окну. Очень часто эти сообщения передают окну информацию о том, что пользователь осуществил ввод с помощью клавиатуры или мыши. Таким образом, например, кнопки "узнают" о том, что они нажаты. Другие сообщения говорят окну о том, что необходимо изменить размер окна или о том, что поверхность окна необходимо перерисовать. Когда программа для Windows начинает выполняться, Windows строит для программы очередь сообщений (message queue). В этой очереди хранятся сообщения для любых типов окон, которые могли бы быть созданы программой. Небольшая часть программы, которая называется циклом обработки сообщений (message loop), выбирает эти сообщения из очереди и переправляет их соответствующей оконной процедуре. Другие сообщения отправляются непосредственно оконной процедуре, минуя очередь сообщений.

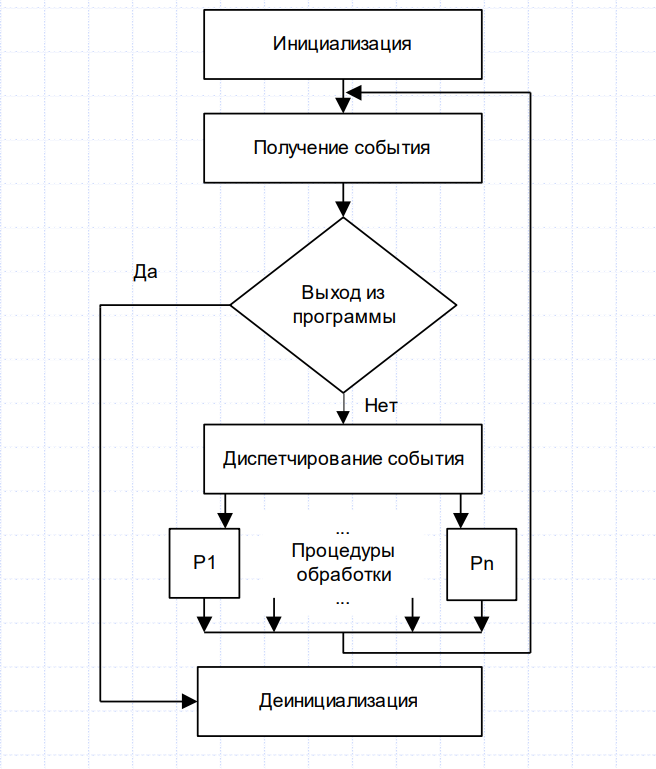


Рисунок 3.1 – Программа с событийным управлением

Отрисовка в рабочей области окна должно происходить при обработке сообщения WM\_PAINT. Это сообщение крайне важно для программирования под Windows. Оно сообщает программе, что часть или вся рабочая область окна недействительна (invalid), и ее следует перерисовать. Но если реализовать отрисовку сразу на экран, то есть вероятность возникновения «мерцания». Картинка будет обновляться быстрее, чем происходит вывод изображения на экран. Чтобы избежать данной проблемы, обычно используют двойную буферизацию. Для этого сначала выделяют память нужного размера под изображение. Затем каждый раз происходит изменяется именно изображения в памяти, а при срабатывании сообщения WM\_PAINT картинка из памяти выводится на экран. Данный способ позволяет избавиться от мерцания в приложении при быстром изменении данных.

В большинство программ для Windows включаются пользовательские значки, которые Windows выводит на экран в левом верхнем углу строки заголовка окна приложения. Кроме этого, Windows выводит на экран значок программы в списках программ меню Start, или в панели задач в нижней части экрана, или в списке программы Windows Explorer. Некоторые программы – наиболее известными из которых являются графические программы для рисования, например Windows Paint, используют собственные курсоры мыши для отражения различных действий программы. В очень многих программах для Windows используются окна меню и диалога. Вместе с полосами прокрутки окна меню и диалога — это основа стандартного пользовательского интерфейса Windows. Значки, курсоры, окна меню и диалога связаны между собой. Все это виды ресурсов (resources) Windows. Ресурсы являются данными, и они хранятся в .EXE файле программы, но расположены они не в области данных, где обычно хранятся данные исполняемых программ. Таким образом, к ресурсам нет непосредственного доступа через переменные, определенные в исходном тексте программы. Они должны быть явно загружены из файла с расширением .EXE в память.

Строка меню выводится на экране непосредственно под строкой заголовка. Эта строка иногда называется главным меню (main menu) или меню верхнего уровня (top-level menu) программы. Выбор элемента главного меню обычно приводит к вызову другого меню, появляющегося под главным, и которое обычно называют всплывающим меню (popup menu) или подменю (submenu). В данном приложении будет реализована строка меню с выбором функций остановки, перезапуска и выхода из игры.

# Разработка программного средства

## Описание алгоритмов решения задачи

Первичная структура и основной алгоритм работы любой программы, предназначенной для работы под ОС Windows, заранее предопределены механизмом сообщений, принятым в данной операционной системе.

Использование сообщений для связи операционной системы с любой запускаемой программой делает необходимым использование как минимум двух основных функций – WinMain и WndProc, так же были использованы функции, реализующие логику игрового процесса.

Был реализован алгоритм добавления объекта(еда) на игровое поле, схема которого представлена на рисунке 4.1.



Рисунок 4.1. – схема алгоритма добавления объекта(еда)

Ещё одним из основных алгоритмов является алгоритм столкновения объекта с границами игрового поля либо с самим собой. Его схема представлена на рисунке 4.2.



Рисунок 4.2. – схема алгоритма столкновения объекта

Одним из важных алгоритмов данной программы является алгоритм движения объекта по игровому полю. Схема этого алгоритма представлена на рисунках 4.3.



Рисунок 4.3.1 – схема алгоритма движения объекта

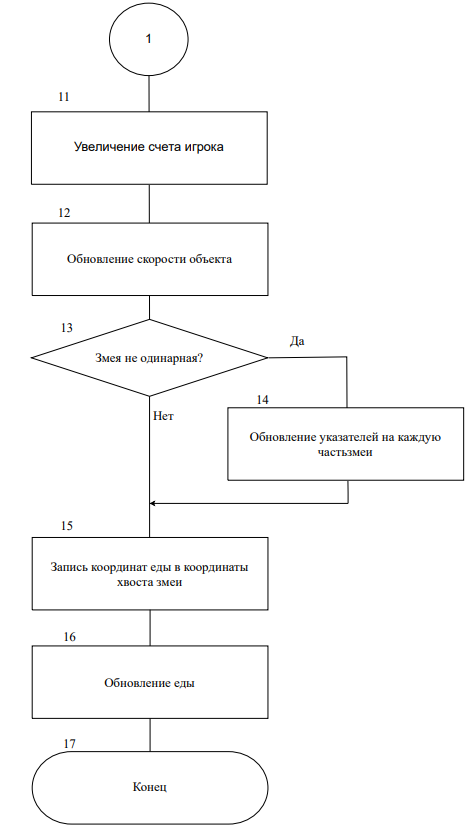


Рисунок 4.3.2 – схема алгоритма движения объекта

## Интерфейс программы

Проанализировав существующие аналоги программы, с помощью интегрированной среды разработки Microsoft Visual Studio 2019 был разработан пользовательский интерфейс. Окно приложения, получило вид, который продемонстрирован на рисунке 4.4.

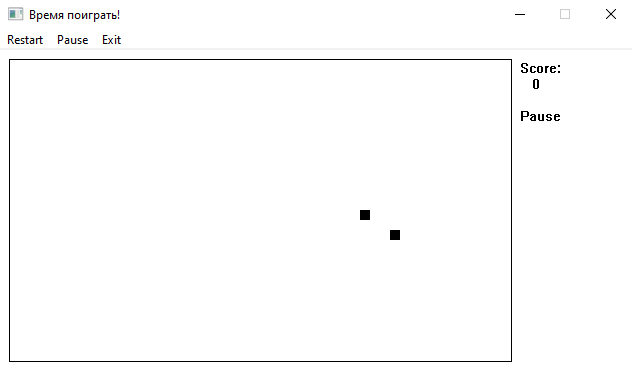


Рисунок 4.4 – Окно приложения

На данном окне расположено меню. В меню расположены 2 кнопки: «Restart» и «Pause».

Кнопка «Restart», позволяет начать новую игру.

Кнопка «Pause» позволяет поставить игру на паузу и при повторном ее нажатии продолжить текущую игру.

При нажатии на кнопку «Exit» осуществляется выход из программы.

## Структура программного средства

В итоге программа игрового приложения «Змейка» была реализована в одном файле:

* Source.cpp — модуль, в котором реализованы все игровые функции и игровая логика, а так же происходит отрисовка интерфейса приложения.

# Тестирование и проверка работоспособности программного средства

## Завершение игры

### Тест 1

Таблица 1 – Тест 1

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка завершения игры проигрышем при столкновении объекта с границами игрового поля |
| Исходный набор данных: |  |
| Ожидаемый результат: | Остановка движения объекта и вывод сообщения «Game over» |
| Полученный результат: |  |

### Тест 2

Таблица 2 – Тест 2

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка завершения игры проигрышем при столкновении объекта с самим собой |
| Исходный набор данных: |  |
| Ожидаемый результат: | Остановка движения объекта и вывод сообщения «Game over» |
| Полученный результат: |  |

### Тест 3

Таблица 3 – Тест 3

|  |  |
| --- | --- |
| Тестовая  ситуация: | Нажатие кнопки «Exit» |
| Исходный набор данных: |  |
| Ожидаемый результат: | Прекращение игры и закрытие окна приложения |
| Полученный результат: | Завершение работы приложения |

## Движение объекта по игровому полю

### Тест 1

Таблица 4 – Тест 1

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка возможности движения объекта вниз при нажатии соответствующей стрелки |
| Исходный набор данных: |  |
| Ожидаемый результат: | Смена направления движения объекта(вниз) |

Продолжение таблицы 4

|  |  |
| --- | --- |
| Полученный результат: |  |

### Тест 2

Таблица 5 – Тест 2

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка возможности движения объекта вверх при нажатии соответствующей стрелки |
| Исходный набор данных: |  |
| Ожидаемый результат: | Смена направления движения объекта(вверх) |
| Полученный результат: |  |

### Тест 3

Таблица 6 – Тест 3

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка возможности движения объекта влево при нажатии соответствующей стрелки |
| Исходный набор данных: |  |
| Ожидаемый результат: | Смена направления движения объекта(влево) |
| Полученный результат: |  |

### Тест 4

Таблица 7 – Тест 4

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка возможности движения объекта вправо при нажатии соответствующей стрелки |
| Исходный набор данных: |  |
| Ожидаемый результат: | Смена направления движения объекта(вправо) |
| Полученный результат: |  |

## Изменение скорости движения объекта

### Тест 1

Таблица 8 – Тест 1

|  |  |
| --- | --- |
| Тестовая  ситуация: | Набранный счет равен 10, изменение скорости объекта |
| Исходный набор данных: |  |

Продолжение таблицы 8

|  |  |
| --- | --- |
| Ожидаемый результат: | Увеличение скорости движения объекта |
| Полученный результат: | Скорость движения объекта увеличилась |

## Проверка работоспособности меню

### Тест 1

Таблица 9 – Тест 1

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка работоспособности пункта меню «Restart» |
| Исходный набор данных: |  |
| Ожидаемый результат: | Обнуление счета и старт игры заново |
| Полученный результат: |  |

### Тест 2

Таблица 10 – Тест 2

|  |  |
| --- | --- |
| Тестовая  ситуация: | Проверка работоспособности пункта меню «Pause» |
| Исходный набор данных: |  |
| Ожидаемый результат: | Остановка игры, вывод текста «Pause» |
| Полученный результат: |  |

# Руководство по использованию программного средства

## Работа с приложением

При запуске программы появляется окно приложения, где находятся пункты меню «Restart», «Pause» и «Exit», счет игрока на данный момент, область, где происходит игровой процесс. Окно приложения представлено на рисунке 6.1.

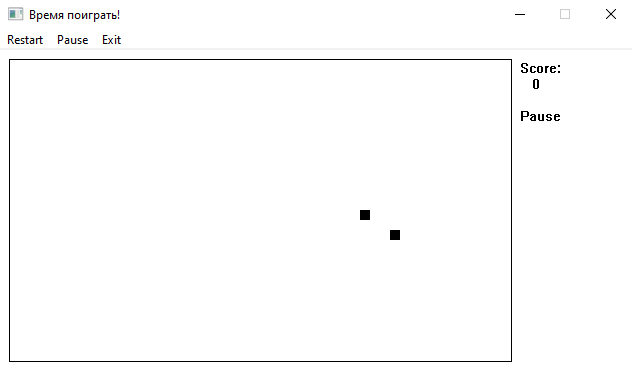


Рисунок 6.1 – Главное окно программы

Процесс игры начинается при запуске приложения. Управление объектом осуществляется с помощью стрелок клавиатуры, каждая из которых определяет направление движения объекта по игровому полю.

C помощью нажатия на кнопку меню «Restart» либо нажатия клавиши «R» игрок может начать игру заново.

C помощью нажатия на кнопку меню «Pause» либо нажатия клавиши «P» игрок может остановить игру, а при повторном нажатии возобновить текущую игру.

C помощью нажатия на кнопку меню «Exit» игрок может завершить работу приложения.

Заключение

По итогу работы над курсовым проектом было разработано игровое приложение «Змейка», обладающее графическим пользовательским интерфейсом для взаимодействия с пользователем.

Для разработки данного приложения были проанализированы программы со схожей тематикой и сформированы требования для программного средства такого типа.

Приложение прошло все этапы тестирования и продемонстрировало корректную работу.

Данное программное средство имеет минимально необходимый набор функций для осуществления игрового процесса по правилам классической игры «Змейка». Среди преимуществ программы можно отметить понятный классический интерфейс и простоту реализации.

В перспективе данное программное средство может быть усовершенствовано (оптимизация кода, добавление новых функций и игровых режимов, добавление сети и возможности игры с другими игроками).

Список использованной литературы

[1] Липпман С., Лажойе Ж. Язык программирования с++. Базовый курс. Киев: Диалектика, 201414. – 601 с.

[2] Дж. Рихтер. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows. - Москва, 2008. – 743 с.

[3] Эккель Б. Философия с++. Введение в стандартный с++. – СПб.: Питер, 2004. – 524 с.

[4] Кенинг Э., Му Б. Эффективное программирования на с++. — М.: Издательский дом "Вильямс", 2002. — 576 с.

[5] Страуструп Б. Программирование принципы и практика с использованием с++. — М.: «Вильямс», 2015. — 1248 с.

[6] Страуструп Б. Язык программирования с++. – Москва.: Бином, 2011. – 1136 с.

[7] MSDN [Электронный ресурс]. - Электронные данные. - Режим доступа: <https://msdn.microsoft.com/en-us/dn308572.aspx>

[8] Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017>

Приложение А

(обязательное)

Исходный код программы (модуль Source.cpp)

#include <windows.h>

#include <time.h>

#include <stdlib.h>

// Определяем 4 направления

#define UP 1 // Вверх, координата y головы змеи продолжает уменьшаться

#define DOWN 2 // Вниз, координата y головы змеи продолжает увеличиваться

#define LEFT 3 // Слева координата x головы змеи продолжает уменьшаться

#define RIGHT 4 // Справа координата x головы змеи продолжает увеличиваться

#define SNAKEWIDTH 10 // Размер одиночной змеи

#define XWIDTH 50 // Ширина игровой области

#define YHEIGHT 30 // Высота игровой зоны

#define GETSCORE 10 // Счет еды

#define ID\_TIMER 1 // ID таймера

//#define TIMERSET 300 //Временной интервал

static int timechunk = 300;

static int PrevScore = 0;

static int direct = RIGHT; // Направление (Примечание: направление определяется головой змеи)

static int Score; // Количество съеденной еды

bool IsOver = false;

bool bEat = false;

struct Snake

{

Snake() :next(NULL), before(NULL)

{

pt.x = 0; pt.y = 0;

}

Snake(const POINT& npt, Snake\*, Snake\*);

POINT pt;

Snake\* next, \* before;

} snake; // Определение головы змеи

Snake::Snake(const POINT& npt, Snake\* nnext, Snake\* nbefore) :

pt(npt), next(nnext), before(nbefore) {}

Snake\* psLast = NULL, \* food = NULL;

bool GameClient[XWIDTH][YHEIGHT]; // Определяем логическое значение координат игровой области, указывающее тело змеи в соответствующей точке и наличие еды.

void GameMenu(HWND hwnd){

HMENU hmenu1;

hmenu1 = CreateMenu();

AppendMenu(hmenu1, MF\_STRING, 1, L"&Restart");

AppendMenu(hmenu1, MF\_STRING, 2, L"&Pause");

AppendMenu(hmenu1, MF\_STRING, 3, L"&Exit");

SetMenu(hwnd, hmenu1);

}

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

TCHAR szAppName[] = TEXT("Snake"),

szWindowName[] = TEXT("Время поиграть!");

WNDCLASS wndcls;

wndcls.cbClsExtra = 0;

wndcls.cbWndExtra = 0;

wndcls.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndcls.hCursor = LoadCursor(hInstance, IDC\_ARROW);

wndcls.hIcon = LoadIcon(hInstance, IDI\_APPLICATION);

wndcls.hInstance = hInstance;

wndcls.lpfnWndProc = WndProc;

wndcls.lpszClassName = szAppName;

wndcls.lpszMenuName = NULL;

wndcls.style = CS\_HREDRAW | CS\_VREDRAW;

RegisterClass(&wndcls);

HWND hwnd = CreateWindow(szAppName, szWindowName,

WS\_OVERLAPPED | WS\_MINIMIZEBOX | WS\_SYSMENU,

CW\_USEDEFAULT, CW\_USEDEFAULT, (XWIDTH + 15) \* SNAKEWIDTH, (YHEIGHT + 8) \* SNAKEWIDTH,

NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

// Обновление новой еды

void NewFood(HWND hwnd)

{

food = new Snake();

SYSTEMTIME st;

GetLocalTime(&st);

srand(st.wMilliseconds);

food->pt.x = rand() % XWIDTH, food->pt.y = rand() % YHEIGHT;

while (GameClient[food->pt.x][food->pt.y])

{

GetLocalTime(&st);

srand(st.wMilliseconds);

food->pt.x = rand() % XWIDTH, food->pt.y = rand() % YHEIGHT;

}

GameClient[food->pt.x][food->pt.y] = true;

InvalidateRect(hwnd, NULL, TRUE);

}

// Определяем, есть ли впереди еда

inline bool IsFood()

{

if (snake.pt.x == food->pt.x && snake.pt.y == food->pt.y)

return true;

else

return false;

}

// Определяем, ударились ли вы об стену или о тело

bool TouchWall()

{

Snake\* temp = psLast;

// За гранью суждения

if (snake.pt.x >= XWIDTH ||

snake.pt.x < 0 ||

snake.pt.y < 0 ||

snake.pt.y >= YHEIGHT)

return true;

// Столкновение с вашим телом

while (temp != &snake)

{

if (snake.pt.x == temp->pt.x &&

snake.pt.y == temp->pt.y)

return true;

temp = temp->before;

}

return false;

}

void Move(HWND hwnd)

{

int x, y; // Используется для записи текущего положения головы змеи (когда не движется вперед)

// Отмечаем, является ли фронт еда, по умолчанию не еда, указывая, что система обычно вперед

Snake\* temp = psLast;

// Записываем текущие координаты головы змеи

x = snake.pt.x;

y = snake.pt.y;

// Проверяем, подходит ли это место для еды, и перемещаем голову змеи вперед

switch (direct)

{

case UP:

--snake.pt.y;

break;

case DOWN:

++snake.pt.y;

break;

case LEFT:

--snake.pt.x;

break;

case RIGHT:

++snake.pt.x;

break;

default:

break;

}

// Если это не еда, делаем следующее

if (!IsFood())

{

// Последний раздел - не змеиная голова

if (temp != &snake)

{

GameClient[temp->pt.x][temp->pt.y] = false;

while (temp != snake.next)

{

temp->pt.x = temp->before->pt.x;

temp->pt.y = temp->before->pt.y;

temp = temp->before;

}

temp->pt.x = x;

temp->pt.y = y;

GameClient[temp->pt.x][temp->pt.y] = true;

}

else // psLast == & snake, то есть у змей только змеиные головы

GameClient[x][y] = false;

if (TouchWall())

IsOver = true; // Затем отмечаем конец игры

else

GameClient[snake.pt.x][snake.pt.y] = true; // В противном случае положение головы змеи после движения помечается квадратом

InvalidateRect(hwnd, NULL, TRUE); // Обновляем игровую область

}

// Если это еда, сделайте следующее

else

{

++Score;

if (timechunk >= 40) {

if (Score > PrevScore + 9) {

timechunk = timechunk - 10;

SetTimer(hwnd, ID\_TIMER, timechunk, NULL);

PrevScore = Score;

}

}

food->before = psLast;

psLast->next = food;

if (temp != &snake)

{

food->pt.x = psLast->pt.x;

food->pt.y = psLast->pt.y;

while (temp != snake.next)

{

temp->pt.x = temp->before->pt.x;

temp->pt.y = temp->before->pt.y;

temp = temp->before;

}

temp->pt.x = x;

temp->pt.y = y;

}

else // psLast == & snake, то есть у змей только змеиные головы

{

food->pt.x = x; food->pt.y = y;

}

psLast = food;

NewFood(hwnd);

}

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)

{

HDC hdc;

PAINTSTRUCT ps;

TEXTMETRIC tm;

static int cxChar, cyChar;

TCHAR szScore[] = TEXT("Score:"),

szGameOver[] = TEXT("Game over"),

szPause[] = TEXT("Pause"),

szBuffer[20],

szCounterFileName[] = L"counter.txt",

\* szText = NULL;

int x, y;

static bool pause = false;

switch (message)

{

case WM\_CREATE:

GameMenu(hwnd);

direct = RIGHT; // Игра запускается, направление по умолчанию правильное

snake.pt.x = XWIDTH / 2; // По умолчанию отображается в середине игровой области

snake.pt.y = YHEIGHT / 2;

psLast = &snake;

GameClient[snake.pt.x][snake.pt.y] = true;

hdc = GetDC(hwnd);

GetTextMetrics(hdc, &tm);

cyChar = tm.tmExternalLeading + tm.tmHeight;

ReleaseDC(hwnd, hdc);

NewFood(hwnd);

SetTimer(hwnd, ID\_TIMER, timechunk, NULL);

return 0;

case WM\_TIMER:

if (pause) return 0;

Move(hwnd);

if (IsOver)

{

KillTimer(hwnd, ID\_TIMER);

InvalidateRect(hwnd, NULL, TRUE);

}

return 0;

case WM\_COMMAND: {

switch (LOWORD(wParam))

{

case 1:

for (int x = 0; x < XWIDTH; ++x)

for (int y = 0; y < YHEIGHT; ++y)

GameClient[x][y] = false;

PrevScore = 0;

timechunk = 300;

SetTimer(hwnd, ID\_TIMER, timechunk, NULL);

direct = RIGHT; // Игра запускается, направление по умолчанию правильное

snake.pt.x = XWIDTH / 2; // По умолчанию отображается в середине игровой области

snake.pt.y = YHEIGHT / 2;

psLast = &snake;

IsOver = false;

pause = false;

GameClient[snake.pt.x][snake.pt.y] = true;

NewFood(hwnd);

break;

case 2:

pause = !pause;

InvalidateRect(hwnd, NULL, TRUE);

break;

case 3:

DestroyWindow(hwnd);

break;

}

}

case WM\_KEYDOWN:

if (IsOver || pause) return 0;

switch (wParam)

{

case VK\_UP:

if (direct != DOWN)

{

direct = UP;

Move(hwnd);

}

break;

case VK\_DOWN:

if (direct != UP)

{

direct = DOWN;

Move(hwnd);

}

break;

case VK\_LEFT:

if (direct != RIGHT)

{

direct = LEFT;

Move(hwnd);

}

break;

case VK\_RIGHT:

if (direct != LEFT)

{

direct = RIGHT;

Move(hwnd);

}

break;

default:

break;

}

return 0;

case WM\_PAINT:

hdc = BeginPaint(hwnd, &ps);

SetViewportOrgEx(hdc, SNAKEWIDTH, SNAKEWIDTH, NULL);

// Рисуем границу игровой области

MoveToEx(hdc, -1, -1, NULL);

LineTo(hdc, XWIDTH \* SNAKEWIDTH + 1, -1);

LineTo(hdc, XWIDTH \* SNAKEWIDTH + 1, YHEIGHT \* SNAKEWIDTH + 1);

LineTo(hdc, -1, YHEIGHT \* SNAKEWIDTH + 1);

LineTo(hdc, -1, -1);

TextOut(hdc, (XWIDTH + 1) \* SNAKEWIDTH, 0, szScore, lstrlen(szScore));

TextOut(hdc, (XWIDTH + 1) \* SNAKEWIDTH, cyChar,

szBuffer, wsprintf(szBuffer, TEXT("%4d"), Score));

if (pause)

TextOut(hdc, (XWIDTH + 1) \* SNAKEWIDTH, 3 \* cyChar, szPause, lstrlen(szPause));

else if (IsOver)

TextOut(hdc, (XWIDTH + 1) \* SNAKEWIDTH, 3 \* cyChar, szGameOver, lstrlen(szGameOver));

SelectObject(hdc, GetStockObject(BLACK\_BRUSH));

for (x = 0; x < XWIDTH; ++x)

{

for (y = 0; y < YHEIGHT; ++y)

{

if (GameClient[x][y])

Rectangle(hdc, x \* SNAKEWIDTH, y \* SNAKEWIDTH,

(x + 1) \* SNAKEWIDTH, (y + 1) \* SNAKEWIDTH);

}

}

return 0;

case WM\_DESTROY:

KillTimer(hwnd, ID\_TIMER);

PostQuitMessage(0);

return 0;

}

return DefWindowProc(hwnd, message, wParam, lParam);

}