МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение Образования

«Белорусский Государственный Университет Информатики и Радиоэлектроники»

кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Тема: Игра «Xonix»

Выполнил:

студент гр. 152002

Игнатенко.А.А.

Проверил:

Юзефович С.В.

Минск 2013

*Содержание*

[Задание 3](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228373)

[Введение 4](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228374)

[Теория 5](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228375)

[Язык C# 5](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228376)

[XNA Framework 6](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228378)

[Практика 9](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228379)

[Основные концепции 9](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228380)

[Описание объектов 10](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228382)

[Диаграмма классов 11](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228383)

[Основные методы и алгоритмы 1](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228383)3

[Руководство пользователя 16](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228385)

[Системные требования 16](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228386)

[Подробное руководство пользователя 17](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228387)

[Заключение 22](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228388)

[Литература 23](file:///C:\универ\3%20семестр\курсач\моя%20курсовая%20записка.docx#_Toc326228389)

***Задание***

В данном курсовом проекте требуется разработать проект игры «Xonix» и реализовать его. Смысл игры — игрок перемещается по помлю, поле условно разделено на «сушу» и «море», цель игрока – отсечь куски «моря», превращая их в «сушу». Враги, управляемые компьютером, должны всячески воспрепятствовать игроку в этом — они разрушают «сушу», заполняя её «морем», «убивают» игрока и т.д. Также есть ограничение по времени, за которое игрок должен осушить ~85% воды.

Целями данного курсового проекта являются:

- изучение принципов разработки объектно-ориентированных проектов

- изучение основ проектирования и разработки игр

- практика работы с графическими библиотеками

***Введение***

Все, кто имеет дело с компьютером, так или иначе сталкивались с компьютерными играми, и подавляющее большинство может сходу назвать несколько игр, которые им особенно понравились. Те, кто уже совсем наигрался, почти наигрался или еше не наигрался, но в процессе общения с компьютером уже начал совмещать игры с чем-нибудь более полезным, возможно, хотели бы придумать какие-нибудь свои, не похожие ни на какие другие игры.

Многое захватывает в таком творчестве. И не сам процесс игры, а разработка игровой вселенной, ее проектирование и реализация. Когда можно слить воедино сценарий, графику, музыку, искусно задуманный и умело запрограммированный алгоритм - создать единый фантастический мир, живущий по законам, которые ты же для него и придумал.

В данной курсовой работе речь пойдет о создании игры «Xonix». Xonix (Ксоникс) — компьютерная игра. Первая версия была создана в 1984 году для платформы PC как клон игры Qix, появившейся ранее на аркадных автоматах. Авторы: Илан Рав (Ilan Rav) и Дани Катц (Dani Katz). Была популярна в 80-е и 90-е годы, существует много версий для разных платформ.

Игровое поле представляет собой сетку из квадратных или прямоугольных ячеек. Ячейки могут быть двух типов: условно «суша» и «море». По полю движутся игрок и управляемые программой враги. Игрок может двигаться по вертикали и горизонтали, враги — в любых направлениях. Враги бывают «сухопутными» и «морскими», то есть движутся или только по «суше», или только по «морю», отскакивая от разделяющей их границы.

Игрок может свободно передвигаться по «суше», где он уязвим для «сухопутных» врагов. Выходя в «море», он оставляет за собой след, уязвимый для «морских» врагов. Как только игрок снова оказывается на «суше», след его превращается в новую «сушу». Если при этом в «море» появилась замкнутая область, не содержащая врагов, то вся эта область также превращается в «сушу».

Проще говоря, игрок пытается отсечь куски «моря», превращая их в «сушу», а враги ему в этом мешают.

***Теория***

При разработке программы применялся [объектно-ориентированный](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](http://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) C#, использующий технологию .Net Framework, а также набор системных библиотек XNA Framework.

**Язык программирования C#:**

C# — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Java, Delphi, Модула и Smalltalk — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR. Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем. (Однако эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET.) CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

## XNA Framework

Платформа XNA Framework – это большой набор системных библиотек, построенных на базе .NET и направленных на улучшение и упрощение создания игр для всех продуктов семейства Microsoft. Создавая игру для PC на базе платформы XNA Framework, вы можете быть уверены, что эта игра будет работать на приставке Xbox 360 и в скором времени на новой системе Windows Mobile. Исключения в этом случае могут составлять некоторые системные классы, которые необходимы только для работы с той или иной платформой. Например, приставка Xbox 360 не имеет мыши, поэтому в программах для консоли нельзя использовать методы и классы, направленные на работу с мышью. В свою очередь, для компьютерных игр можно задавать различные разрешения экрана, тогда как Xbox 360 использует в качестве монитора телевизор, который имеет свою специфику вывода изображения на экран. Эти и другие исключения составляют мышиную долю (примерно 3–5%) от всей библиотеки XNA Framework. При правильном подходе в использовании библиотечных классов можно написать программу, которая на 100% будет работать как на консоли, так и на компьютере и, надеемся, на Windows Mobile тоже. Еще одной важной особенностью платформы XNA Framework является значительное упрощение в подходе реализации игр для всех платформ. Прежде чем начать писать именно исходный код игр с использованием DirectX SDK, необходимо создать большую кучу различных объектов, создать окно, настроить видеоадаптер и т. д. Применяя платформу XNA Framework, вы все перечисленное и даже больше создадите за пару строк исходного кода, при этом эти строки сформирует за вас сам компилятор и XNA Framework! Работать стало проще, быстрее, и главное – нет борьбы и драк с DirectX… Роль DirectX в этом деле – это низко уровневая прослойка API, к которой вы не касаетесь никоим образом. Все манипуляции производятся только через библиотеку XNA Framework.

Условно все компоненты XNA Framework можно разделить на 4 уровня абстракции:

* Platform (Платформа) - самый нижний уровень, содержащий платформо-зависимые API, такие как неуправляемый DirectX. В подавляющем большинстве случаев приложение может ничего не знать о существовании этого уровня, используя компоненты более высоких уровней абстракции. Более того, прямое обращение уровню Platform неминуемо сузит диапазон платформ, поддерживаемых приложением: не исключено, что Microsoft в будущем добавит поддержку XNA Framework и в операционные системы для карманных устройств.
* Core Framework (Основной Каркас) - нижний платформо-независимый уровень XNA, обеспечивающий базовую функциональность. Размещается в сборке Microsoft.Xna.Framework.dll и содержит 5 компонентов: Graphics (работа с графикой), Audio (работа со звуком), Input (работа с устройствами ввода-вывода), Math (математические расчеты), Storage (работа с файловой системой). Классы и структуры каждого компонента сгруппированы в пространства имен ([таблица 1.3](http://www.intuit.ru/department/se/intxna/1/3.html#table.1.3)). На платформе Windows первые три компонента ( Graphics, Audio, Input ) являются надстройками над DirectX, а компонент Storage - надстройкой над классами .NET Framework для работы с файловой системой. Однако следует всегда помнить о том, что на других платформах всe может обстоять совершенно иначе.
* Extended Framework (расширенный каркас) - набор высокоуровневых классов, решающих типовые задачи, встающие перед разработчиком игр: инициализация графического устройства, организация цикла обработки сообщений, экспорт моделей и текстур из графических редакторов. По сути Extended Framework можно считать универсальным игровым движком ( Game Engine ) начального уровня. Размещается в сборкеMicrosoft.Xna.Framework.Game.dll.
* Game - собственно приложение пользователя, то есть наши с вами программы. К слову, в комплект XNA входит несколько простых игр ( Starter Kits ), которые можно использовать в качестве заготовок для своих приложений.

В данном проекте используется XNA Game Studio 4.0 интегрированный в Visual Studio 2010.

# *Практика*

## Основные концепции

Игровое поле представляет собой сетку из квадратных или прямоугольных ячеек. Ячейки могут быть двух типов: условно «суша» и «море». По полю движутся игрок и управляемые программой враги. Игрок может двигаться по вертикали и горизонтали, враги — в любых направлениях. Враги бывают «сухопутными» и «морскими», то есть движутся или только по «суше», или только по «морю», отскакивая от разделяющей их границы.

Игрок может свободно передвигаться по «суше», где он уязвим для «сухопутных» врагов. Выходя в «море», он оставляет за собой след, уязвимый для «морских» врагов. Как только игрок снова оказывается на «суше», след его превращается в новую «сушу». Если при этом в «море» появилась замкнутая область, не содержащая врагов, то вся эта область также превращается в «сушу».

Необходимо реализовать:

* Физику движения игрока
* Физику движения врагов
* Логику взаимодействия врагов между собой и с игроком
* Логику взаимодействия игрового поля с игроком и врагами
* Систему бонусов, жизней
* Реализовать систему уровней
* Систему ограничения прохождения одного уровня по времени

По сути, игра представляет собой меняющиеся по мере прохождения уровни. Игра прекращается, как только пользователь потерял все жизни или прошёл все уровни.

## Описание объектов

При проектировании игры была использована методология объектно-ориентированного программирования.

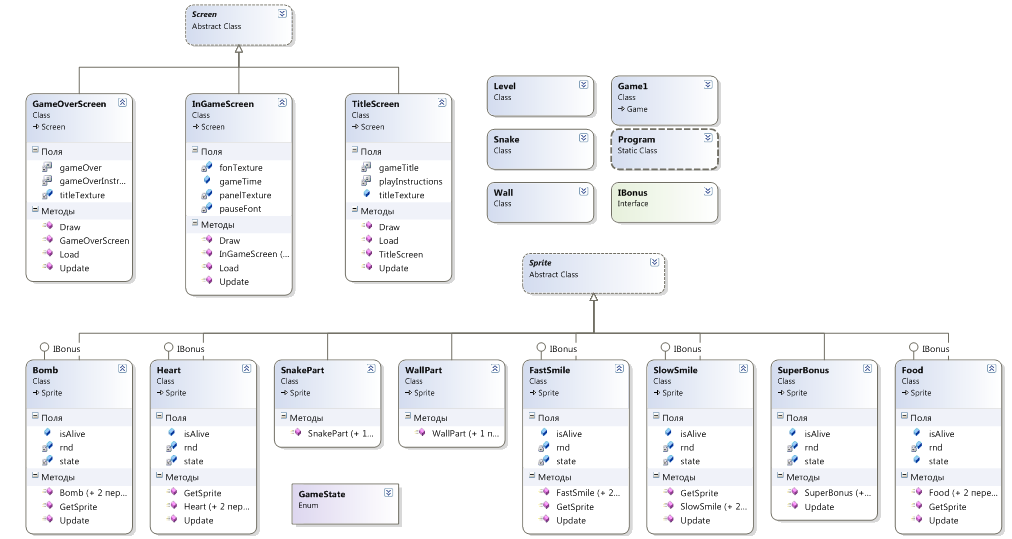
Базовым для всех игровых компонентов является абстрактный класс **GameObject**. Он реализует интерфейс IUpdatable (содержит метод Update, указывающий на возможность обновления состояние объекта; свойство Enabled, указывающее на необходимость обновлять обновления состояния объекта), а также интерфейс IDisposable, содержащий метод Dispose, вызов которого уничтожает объект, заставляя программу прекратить обновления состояния объекта.

Следующим в иерархии классов является абстрактный класс **DrawableGameObject**, являющийся наследником **GameObject**. Он к функционалу GameObject добавляет возможно отрисовки объекта на экране (метод Drawб, свойство Visible, Position и т.д.).

Абстрактный класс **InteractiveGameObject**, наследник **DrawableGameObject**, добавляет возможность взаимодействия интерактивных объектов между собой, определения столкновений.

Далее абстрактный класс **MoveableGameObjetc**, наследующий **InteractiveGameObject**, добавляет возможность передвижения объекта по экрану (методы MoveTo, Stop, свойства Velocity, Moving).

Объект, управляемый игроком представлен классом **MainDevice**, являющийся наследником MoveableGameObjetc. Игровое поле представлено классом **Playground**, наследующим InteractiveGameObject.

**Диаграмма классов**

**Game** – авто сгенерированный класс с помощью XNA.

**Program** – стандартный класс С# с точкой входа в приложение.

**Level** – класс, реализующий игровые уровни.

**GameOverScreen** – класс для отображения экрана в конце игры.

**InGameScreen** – класс для отображения экрана во время игры.

**TitleScreen** – класс для отображения экрана в начале игры с игровым меню.

**Screen** – базовый класс для экранов игры.

**Bomb** – класс для представления бомбочки.

**FastSmile** – класс для представления смайлика-ускорителя.

**Food** – класс для представления клевера.

**Heart** – класс для представления бонуса дополнительной жизни.

**SlowSmile** – класс для представления смайлика-замедлителя.

**Snake** – класс для представления змейки (содержит методы взаимодействия с бонусами).

**SnakePart** – класс для представления одного сегмента тела змейки.

**Sprite** – базовый класс, содержащий методы Draw и Save.

**SuperBonus** – класс для представления бонуса, меняющего свои свойства.

**Wall** – класс для представления массива стен на игровом поле.

**WallPart** – класс для представления одной стены.

# *Основные методы и алгоритмы*

У класса Snake есть метод Move, который реализует передвижение змейки по игровому полю в зависимости от заданного пользователем направления. (Змейка не может поменять движение на противоположное: например, если она движется вправо, а пользователь нажал «влево».) Змейка перемещается следующим образом: начиная с последнего сегмента тела змейки, он перемещается на позицию предыдущего, а голова змейки перемещается в зависимости от направления движения.

public void Move(GameTime gameTime)

{

//получаем текущее состояние клавиатуры

KeyboardState kbState = Keyboard.GetState();

//если пользователь нажал "влево" и змейка не движется вправо

if (kbState.IsKeyDown(Keys.Left) && move != direction.right)

{

move = direction.left;

}

//если пользователь нажал "вправо" и змейка не движется влево

else if (kbState.IsKeyDown(Keys.Right) && move != direction.left)

{

move = direction.right;

}

//если пользователь нажал "вверх" и змейка не движется вниз

else if (kbState.IsKeyDown(Keys.Up) && move != direction.down)

{

move = direction.up;

}

//если пользователь нажал "вниз" и змейка не движется вверх

else if (kbState.IsKeyDown(Keys.Down) && move != direction.up)

{

move = direction.down;

}

timeElapsed += gameTime.ElapsedGameTime.Milliseconds;

if (timeElapsed > millisecondsPerMove)

{

timeElapsed = 0;

//перемещаем каждый сегмент хвоста змеи на позицию предыдущего сегмента

int i = array.Length - 1;

while (i > 0)

{

array[i].rectangle = array[i - 1].rectangle;

i--;

}

//перемешаем голову змеи в зависимости от направления

if (move == direction.right)

{

array[0].rectangle.X += array[0].texture.Width;

}

else if (move == direction.left)

{

array[0].rectangle.X -= array[0].texture.Width;

}

else if (move == direction.up)

{

array[0].rectangle.Y -= array[0].texture.Height;

}

else if (move == direction.down)

{

array[0].rectangle.Y += array[0].texture.Height;

}

}

}

Метод Collision в классе Snake отвечает за взаимодействие объекта класса Snake и объекта класса Food. При их пересечении размер змейки увеличивается на один сегмент, увеличивается количество очков, объект класса Food как бы «умертвляется», чтобы сгенерироваться в другом месте.

public void Collision()

{

//если пересеклись змейка и еда

if (array[0].rectangle.Intersects(game.food.rectangle))

{

//объект еды "умертвляем"

game.food.isAlive = false;

score += scorePlus; //добавляем очки

SnakePart[] buffer = array;

snakeSize++; //увеличиваем размер змейки на один сегмент

array = new SnakePart[snakeSize];

for (int i = 0; i < buffer.Length; i++)

{

array[i] = buffer[i];

}

//добавим сегмент тела змейки

if (array[array.Length - 2].rectangle.X - array[array.Length - 3].rectangle.X == 0)

{

//если у двух сегментов тела змейки совпадают иксовые координаты

array[array.Length - 1] = new SnakePart(texture, new Rectangle(array[array.Length - 2].rectangle.X,

array[array.Length - 2].rectangle.Y + texture.Height, texture.Width, texture.Height));

}

else if (array[array.Length - 2].rectangle.Y - array[array.Length - 3].rectangle.Y == 0)

{

//если у двух сегментов тела змейки совпадают игриковые координаты

array[array.Length - 1] = new SnakePart(texture,

new Rectangle(array[array.Length - 2].rectangle.X + texture.Width,

array[array.Length - 2].rectangle.Y, texture.Width, texture.Height));

}

}

}

Многие классы содержат метод Update для обновления состояния объекта на игровом поле. В классе Food он реализован следующим образом.

public void Update(Game1 game)

{

//если состояние объекта isAlife = false, то генерируем новое место расположения на игровом поле

if (!isAlive)

{

state = false;

while (!state)

{

//произвольным образом генерируем координаты от 0 до ширины/высоты экрана

int x = (rnd.Next(0, game.gameScreenWidth / texture.Width)) \* texture.Width;

int y = (rnd.Next(0, game.gameScreenHeight / texture.Height)) \* texture.Height;

rectangle = new Rectangle(x, y, texture.Width, texture.Height);

state = true;

//проверяем, занимает ли эти координаты змейка

for (int i = 0; i < game.snake.array.Length; i++)

{

if (game.snake.array[i].rectangle.Intersects(rectangle))

{

state = false;

}

}

//проверяем, не заняты ли эти координаты уже другими объектами игры

if (game.bomb.rectangle.Intersects(rectangle) || game.heart.rectangle.Intersects(rectangle)

|| game.slowSmile.rectangle.Intersects(rectangle) || game.fastSmile.rectangle.Intersects(rectangle)

|| game.superBonus.rectangle.Intersects(rectangle))

{

state = false;

}

//проверяем заняты ли выбранные координаты стенами

if (game.wall.state)

{

for (int i = 0; i < game.wall.wallArray.Length; i++)

{

if (game.wall.wallArray[i].rectangle.Intersects(rectangle))

{

state = false;

}

}

}

}

//если удалось найти координаты, которые не заняты другими объектами, то состоянию объекта isAlive присваимваем значение true

isAlive = true;

}

}

# *Руководство пользователя*

## Системные требования

**Операционная система**: Microsoft Windows XP, Vista, 7;

**Видеокарта**: DirectX – совместимая видеокарта, поддерживающая разрешение не менее 800х600 пикселей;

**Жесткий диск**: 10 мб свободного места;

**Оперативная память**: 128 мб;

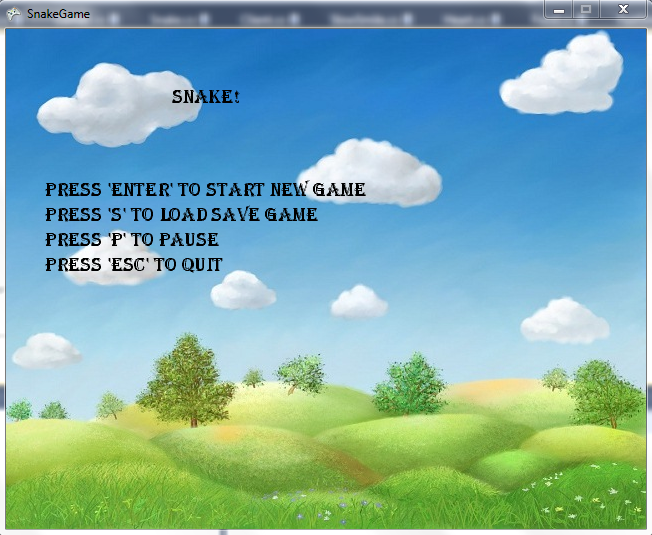
**Клавиатура;**

**Программное обеспечение**: для работы приложения требуется Microsoft .**NET Framework** версии не ниже **4.0**. Для работы с исходным кодом требуется **Microsoft XNA 4.0** (продукты доступны для бесплатного скачивания, http://www.microsoft.com/downloads)

## Подробное руководство пользователя

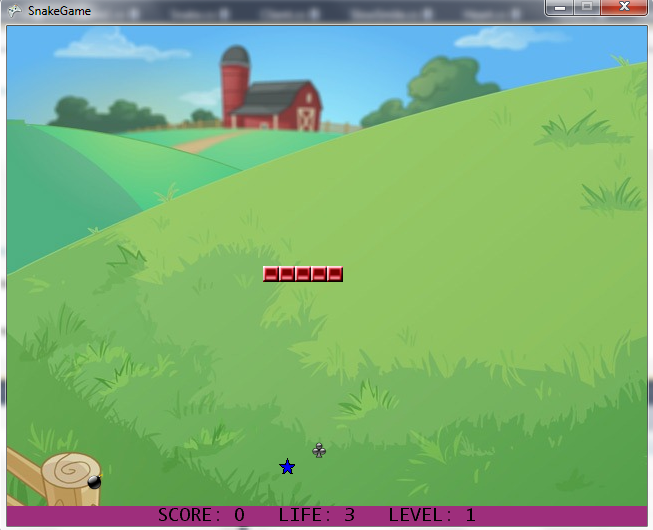
Перед запуском игры удостоверьтесь, что ваш компьютер удовлетворяет системным требованиям игры, в противном случае разработчик не несет никакой ответственности за некорректную работу продукта.

При запуске программы вы попадаете в главное меню. При нажатии “ESC” будет произведет выход из игры.



При нажатии “ENTER” будет начата новая игра, а при нажатии “S” запустится последняя сохраненная игра.

Внизу поля отображается текущее количество очков, количество жизней и текущей уровень. На первом уровне имеются следующие бонусы: клевер, бомбочка и звездочка. Чтобы перейти на следующий уровень, нужно набрать определенное количество очков, которые даются за клевер.

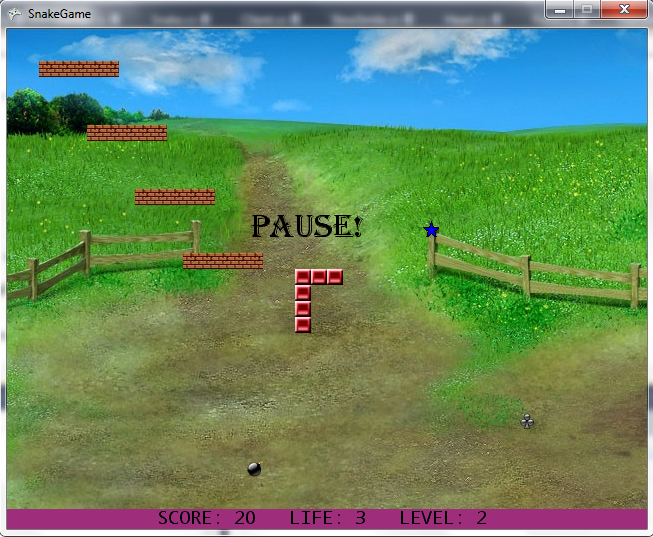


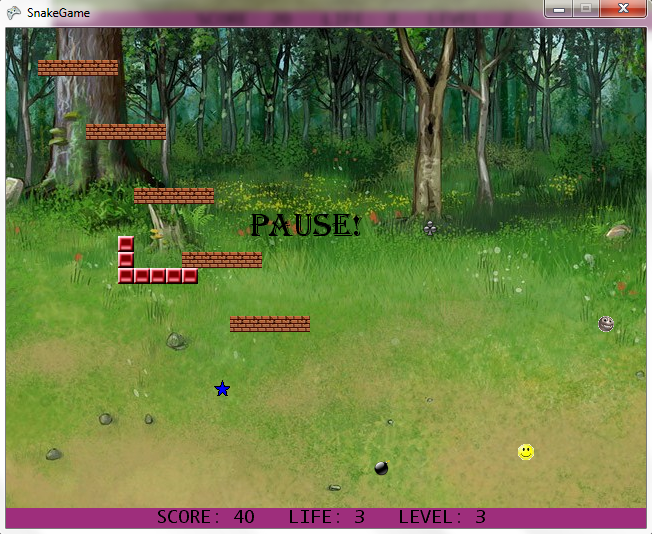
При нажатии на “P” игра ставится на паузу.

При переходе на следующий уровень игра автоматически ставится на паузу. За каждый новый уровень игроку дается дополнительная жизнь.

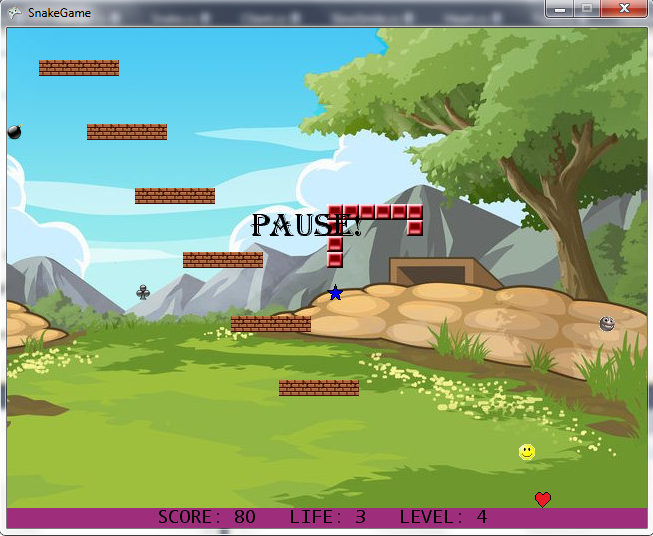
Перейдя на второй уровень, мы видим, что у нас изменилась фоновая картинка игры, а также появились кирпичные стены.

Чтобы выйти из паузы нужно снова нажать “P”.





На третьем уровне появляются дополнительные бонусы: серый смайлик ускоряет нашу змейку, а желтый смайлик – замедляет, а также увеличилось количество кирпичных стен. Фоновая картинка снова поменялась.

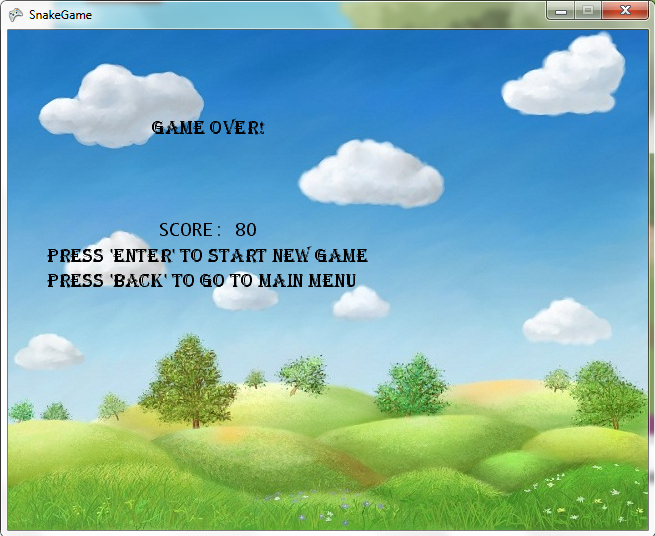


На четвертом уровне количество стен еще увеличилось и появился еще один бонус – сердечко (дополнительная жизнь).

Когда количество жизней равно нулю, игра заканчивается.

На экране Вы сможете увидеть количество набранных очков. При нажатии “ENTER” начнется новая игра, а при нажатии “BACK” Вы попадете в начальное меню.

При закрытии игрового окна во время игры происходит ее автоматическое сохранение.



# *Заключение*

При выполнении данной работы, потребовались высокие знание языка программирования. Мною было разработано полноценное игровое приложение, использующее двухмерную графику. Были изучены современные технологии разработки игровых приложений, в частности Microsoft XNA.

Я получила достаточный опыт в разработке игр, научилась применять знания, полученные в ходе изучения курса ООП.

Возможные применения данного курсового проекта: как и любая игра, он создан для приятного время препровождения.

На данном примере мы видим преимущества объектно-ориентированного программирования и использования современных средств разработки, таких как XNA Game Studio и Visual Studio.

# *Литература*

1. Э. Троелсен. Язык С# и платформа .NET 4.0. изд., 2011
2. С. Горнаков «Программирование компьютерных игр под Windows в XNA Game Studio». 2008
3. Сайт <http://www.cyberforum.ru>
4. Сайт Wikipedia.org
5. Герберт Шилдт "C# 3.0, 4.0. Полное руководство"