Министерство образования Республики Беларусь Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей Кафедра информатики

Курсовой проект по дисциплине:

«Программирование»

Пояснительная записка к курсовой работе

Тема работы:

«Игра "Жизнь"

(Conway's Game of Life)»

Исполнитель

студентка гр. 653501 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Никитинская А. С.

(подпись дата)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Козуб В.Н.

(подпись дата)

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

(оценка)

Минск

2017 год

**Содержание**

1. Введение………………………………………………………………3
2. Обзор аналогов……………………………………………………….4
3. Что представляет собой Microsoft XNA framework………………..6
4. Разработка приложения……………………………………………...7
5. Описание работы программы………………………………………15
6. Пользовательский интерфейс………………………………………16
7. Заключение…………………………………………………………..18
8. Список используемой литературы…………………………………19

**Введение**

Иногда даже простой набор элементарных правил может дать очень интересные результаты.

Игра "Жизнь" — клеточный автомат, разработанный в 1970-х годах британским математиком Джоном Конвеем.

Место действия этой игры — "вселенная" — это размеченная на клетки поверхность или плоскость. Каждая клетка на этой поверхности может находиться в двух состояниях: быть "живой" или быть "мертвой". Нам необходима двумерная матрица, у которой имеется лишь два типа клеток и набор правил, определяющий их следующее состояние, чтобы наблюдать огромное разнообразие форм, которые могут возникнуть в процессе игры.

Игра "Жизнь" — это не совсем игра — это скорее механизм (в основном потому, что в ней нет игрока и нет цели), который развивается на основе его первоначальных условий.

Об увлекательности игры свидетельствуют результаты множества интересных исследований и многочисленных компьютерных реализаций. При этом она имеет непосредственное отношение к перспективной в настоящее время области математики — теории клеточных автоматов.

Таким образом, целью данной курсовой работы является создание игры "Жизнь", включающей в себя следующие функциональные особенности: игровое меню, игровой интерфейс с возможностью ручного заполнения поля.

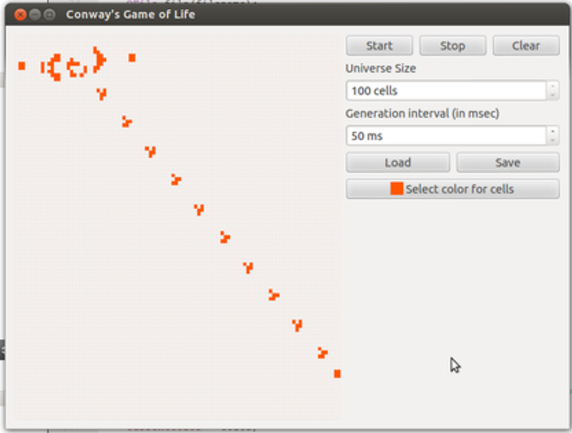
Для достижения цели необходимо решить следующие задачи:

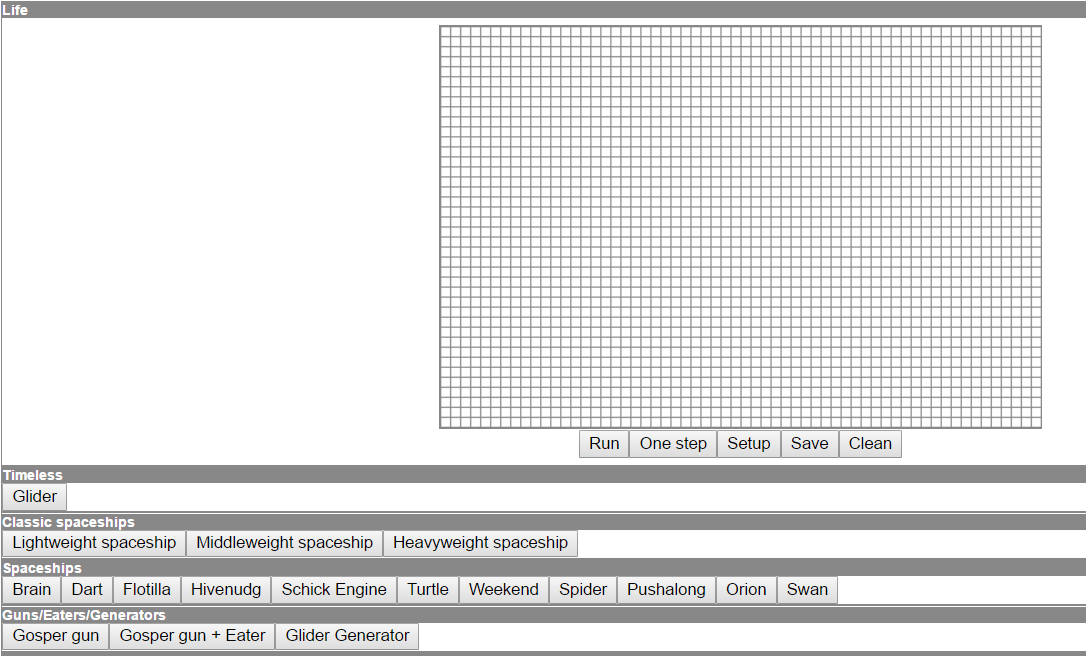
* научиться работать на платформе XNA Game Studio 4.0, которая является расширением для Visual Studio от компании Microsoft для разработки игр на XNA;
* углубить и расширить знания в языке программирования С#;
* реализовать основные принципы игры "жизнь";
* разработать игру с удобным интерфейсом на платформе Microsoft XNA Game Studio 4.0.

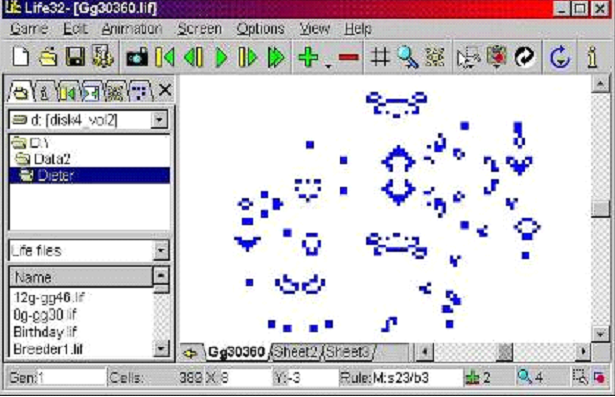
**Обзор аналогов**

Игра "Жизнь" имеет много аналогов, реализованных на различных языках программирования.

Ниже приведено несколько примеров:







Проанализировав несколько игр, разработанных по правилам игры "Жизнь", я выявила недочеты, которых нужно было избежать в собственной игре.

Недочеты:

* Большое количество редкоиспользуемых кнопок. Некоторые игры обладают большим количеством различных кнопок, что значительно увеличивает время, которое потребуется пользователю, чтобы освоить приложения.
* Невзрачный, простой дизайн.

Пути решения:

* Создать интуитивное игровое меню, понятное начинающему пользователю.
* Улучшить дизайн.
* Добавить звуковое оформление.

**Что представляет собой Microsoft XNA framework**

Microsoft XNA framework — это управляемая библиотека для работы с трёхмерной графикой, звуком, устройствами ввода (клавиатура, мышь, джойстик) и тд.   
Тут стоит подчеркнуть слово «управляемая». Дело в том, что XNA framework — это библиотека, которая основана на платформе .NET.

XNA — это надстройка над неуправляемым (классическим) DirectX. Во всяком случае такую картину можно наблюдать для операционной системы Windows. Дело в том, что XNA является кроссплатформенной библиотекой.  
Библиотека разрабатывалась корпорацией Microsoft с целью создания единого инструмента для разработки игр на двух не совместимых платформах, принадлежащих той же компании. А именно: [Microsoft Windows](http://ru.wikipedia.org/wiki/Windows) и [Microsoft XBOX360](http://ru.wikipedia.org/wiki/Xbox_360).

**Разработка приложения**

Самый основной элемент в игре "Жизнь" — клетки, которые являются "жизненными формами", составляющими основу всего моделирования. Каждая клетка может находиться в одном из двух состояний: быть "живой" или "мертвой".

Клетки не двигаются, они просто воздействуют на своих соседей, основываясь на их текущем состоянии.

Клетки должны выполнять следующие задачи:

* Они должны отслеживать свое положение, границы и состояние, чтобы нажатием мыши пользователь смог правильно нарисовать то, что он захочет.
* Они должны переключаться между живыми и мертвыми левым/правым нажатием мыши, что позволит пользователю делать интересные "формы жизни".
* Они должны быть нарисованы белыми, если они "мертвы", или черными, если они "живы".

Все вышеперечисленные задачи успешно выполняются в классе Cell.

**class** Cell

{

**public** Point Position { **get**; **private** **set**; }

**public** Rectangle Bounds { **get**; **private** **set**; }

**public** **bool** IsAlive { **get**; **set**; }

**public** Cell(Point position)

{

Position = position;

Bounds = **new** Rectangle(Position.X \* Game1.CellSize, Position.Y \* Game1.CellSize, Game1.CellSize, Game1.CellSize);

IsAlive = **false**;

}

**public** **void** Update(MouseState mouseState)

{

**if** (Bounds.Contains(**new** Point(mouseState.X, mouseState.Y)))

{

// Создает клетку левым нажатием мыши и

// удаляет правым нажатием.

**if** (mouseState.LeftButton == ButtonState.Pressed)

IsAlive = **true**;

**else** **if** (mouseState.RightButton == ButtonState.Pressed)

IsAlive = **false**;

}

}

**public** **void** Draw(SpriteBatch spriteBatch)

{

**if** (IsAlive)

spriteBatch.Draw(Game1.Pixel, Bounds, Color.Black);

}

}

Теперь, когда каждая ячейка ведет себя правильно, необходимо иметь сетку, которая будет содержать все клетки, и реализовать логику, которая сообщает каждой клетке, должна ли она ожить, остаться в "живых", остаться в "мертвых" или умереть. Правила, по которым все это реализовывается довольно просты:

* Любая "живая" клетка, у которой менее двух "живых" соседей, умирает от "одиночества".
* Любая "живая" клетка, у которой два или три "живых" соседа, продолжает жить.
* Любая "живая" клетка, у которой более трех "живых" соседа, умирает от "перенаселения".
* Любая "мертвая" клетка, у которой ровно три "живых" соседа, оживает.

Таким образом, пока игра будет запускать обновление через постоянные промежутки времени, сетка будет проверять каждое из этих правил для всех ячеек в сетке. Это достигнуто в классе Grid.

for (var i = 0; i < Size.X; i++)

{

for (var j = 0; j < Size.Y; j++)

{

// Проверяет текущее состояние клетки и считает ее живых

// соседей.

bool living = \_cells[i, j].IsAlive;

int count = GetLivingNeighbors(i, j);

bool result = false;

// Применяет правила и устанавливает следующее состояние

// клетки.

if (living && count < 2)

result = false;

if (living && (count == 2 || count == 3))

result = true;

if (living && count > 3)

result = false;

if (!living && count == 3)

result = true;

\_cells[i, j].IsAlive = result;

}

}

Метод GetLivingNeighbors подсчитывает, сколько из соседей текущей ячейки в настоящий момент живы. В данной функции первый оператор if каждой пары проверяет, находится ли клетка на краю сетки. Если бы этой проверки не было, то мы столкнулись бы с некоторыми исключениями из-за превышения границы массива.

public int GetLivingNeighbors(int x, int y)

{

int count = 0;

// Проверяет клетку справа.

if (x != Size.X - 1)

if (\_cells[x + 1, y].IsAlive)

count++;

// Проверяет клетку в правом нижнем углу.

if (x != Size.X - 1 && y != Size.Y - 1)

if (\_cells[x + 1, y + 1].IsAlive)

count++;

// Проверяет клетку внизу.

if (y != Size.Y - 1)

if (\_cells[x, y + 1].IsAlive)

count++;

// Проверяет клетку в левом нижнем углу.

if (x != 0 && y != Size.Y - 1)

if (\_cells[x - 1, y + 1].IsAlive)

count++;

// Проверяет клетку слева.

if (x != 0)

if (\_cells[x - 1, y].IsAlive)

count++;

// Проверяет клетку в левом верхнем углу.

if (x != 0 && y != 0)

if (\_cells[x - 1, y - 1].IsAlive)

count++;

// Проверяет клетку вверху.

if (y != 0)

if (\_cells[x, y - 1].IsAlive)

count++;

// Проверяет клетку в правом верхнем углу.

if (x != Size.X - 1 && y != 0)

if (\_cells[x + 1, y - 1].IsAlive)

count++;

return count;

}

До сих пор вся логика, которая была реализована, разумна, но она не будет вести себя правильно, если не учесть один момент: все наши ячейки должны обновляться в одно и то же время, чтобы правильно выполнялись правила игры.

Например, цикл выше проверяет все ячейки слева направо, поэтому, если клетка слева, которая была только что проверена, ожила, это изменило бы счетчик для ячейки в середине, которую надо проверять на следующем шаге.

Необходимо иметь возможность проверять ячейки в произвольном порядке, и следующий шаг всегда должен быть идентичным.

Для того чтобы это реализовать, нужно иметь вторую сетку ячеек в памяти для их следующего состояния. Т. е. каждый раз, когда определяется следующее состояние клетки, оно сохраняется во второй сетке для следующего состояния всей системы. Затем, когда будут найдены следующие состояния для каждой клетки, их можно будет применить одновременно.

public void SetNextState()

{

for (var i = 0; i < Size.X; i++)

for (var j = 0; j < Size.Y; j++)

\_cells[i, j].IsAlive = \_nextCellStates[i, j];

}

Имеющаяся сетка уже готова к использованию, однако она станет намного яснее для пользователя, если добавить некоторые линии. Это позволит пользователю более легко видеть границы ячеек.

public void Draw(SpriteBatch spriteBatch)

{

foreach (Cell cell in \_cells)

cell.Draw(spriteBatch);

// Рисует вертикальные линии.

for (var i = 0; i < Size.X; i++)

spriteBatch.Draw(Game1.Pixel, new Rectangle(i \* Game1.CellSize - 1, 0, 1, Size.Y \* Game1.CellSize), Color.DarkGray);

// Рисует горизонтальные линии.

for (var j = 0; j < Size.Y; j++)

spriteBatch.Draw(Game1.Pixel, new Rectangle(0, j \* Game1.CellSize - 1, Size.X \* Game1.CellSize, 1), Color.DarkGray);

}

Проблема, с которой можно столкнуться на этом этапе разработки, заключается в том, что щелчки мыши не всегда регистрируются, когда пользователь пытается обновить ячейку. Например, когда пользователь нажимает и перетаскивает курсор мыши по сетке, позади остается пунктирная, а не сплошная линия. Это происходит потому, что скорость, с которой обновляются ячейки, является также скоростью, с которой проверяется мышь. Это слишком медленно. Поэтому необходимо просто отделить скорость, с которой игра обновляется, и скорость, с которой считывается ввод.

// Обновления в секунду

Public const int UPS = 20;

Public const int FPS = 100;

Чтобы управлять состояниями игры(главное меню; сама игра; отображение информации о игре; отображение информации, как пользоваться игрой) необходимо иметь объект enum в классе Game1.

private enum GameState

{

Menu,

InGame,

AboutGame,

HowToPlay

};

// Устанавливает текущее (первоначальное) состояние игры: Меню

private GameState currentGameState = GameState.Menu;

Далее, в методе Draw(GameTime gameTime) для каждого состояния игры прорисовываются элементы, которые необходимы в текущий момент.

Так, например, если текущее состояние игры будет "Меню", прорисуется следующее:

switch (currentGameState)

{

case GameState.Menu:

GraphicsDevice.Clear(Color.White);

spriteBatch.Begin();

// Прорисовка фонового изображения.

spriteBatch.Draw(backgroundTexture,

new Rectangle(0, 0, Window.ClientBounds.Width,

Window.ClientBounds.Height), null,

Color.White, 0, Vector2.Zero,

SpriteEffects.None, 0);

// Прорисовка рамки с названием игры.

spriteBatch.Draw(frameTexture,

new Rectangle(0, 0, Window.ClientBounds.Width,

Window.ClientBounds.Height), null,

Color.White, 0, Vector2.Zero,

SpriteEffects.None, 0);

// Прорисовка необходимых кнопок: Начать, Об игре, Выход

for (var i = 0; i < NumberOfMenuButtons; i++)

spriteBatch.Draw(ButtonTexture[i], ButtonRectangle[i], ButtonColor[i]);

spriteBatch.End();

break;

…

}

**Описание работы программы**

Глобальная переменная GraphicsDeviceManager graphics позволяет получить доступ к графическому устройству компьютера. Другой глобальный объект — SpriteBatch spriteBatch служит для отрисовки спрайтов — изображений, которые используются в игре.

Первым вызывается конструктор класса Game1. Объект GraphicsDeviceManager, который создаётся в этом классе, является проводником между игрой и видеокартой. Вся отрисовка в игре будет проходить через этот объект.

Затем вызывается метод Initialize(). Здесь создаётся объект класса Grid, отвечающий за главный компонент всей игры — сетку, на которой и будет разворачиваться "жизнь". А также инициализируются кнопки (указываются их координаты и параметры).

Далее вызывается метод LoadContent(). В нем загружаются все необходимые ресурсы для игры, такие как задний фон(backgroundTexture = Content.Load<Texture2D>(@"Images\font")), изображения всех кнопок(ButtonTexture[StartButtonIndex] = Content.Load<Texture2D>(@"images/start"); ...), необходимые аудиофайлы (audioEngine = new AudioEngine(@"Content\Audio\GameAudio.xgs"); …) и т.д..

Затем начинается игровой цикл, который состоит из двух методов: Update() и Draw(). Первый предназначен для обработки игровой логики, пользовательского ввода, изменения положения объектов. Метод Draw() предназначен для отрисовки на экране всех объектов.

В методе Update() проверяются нажатые пользователем клавиши. Напрмер, если пользователь нажал клавишу Esc на клавиатуре, то срабатывает метод Exit(), который завершает выполнение программы.

if (keyboardState.IsKeyDown(Keys.Escape) && lastKeyboardState.IsKeyUp(Keys.Escape))

{

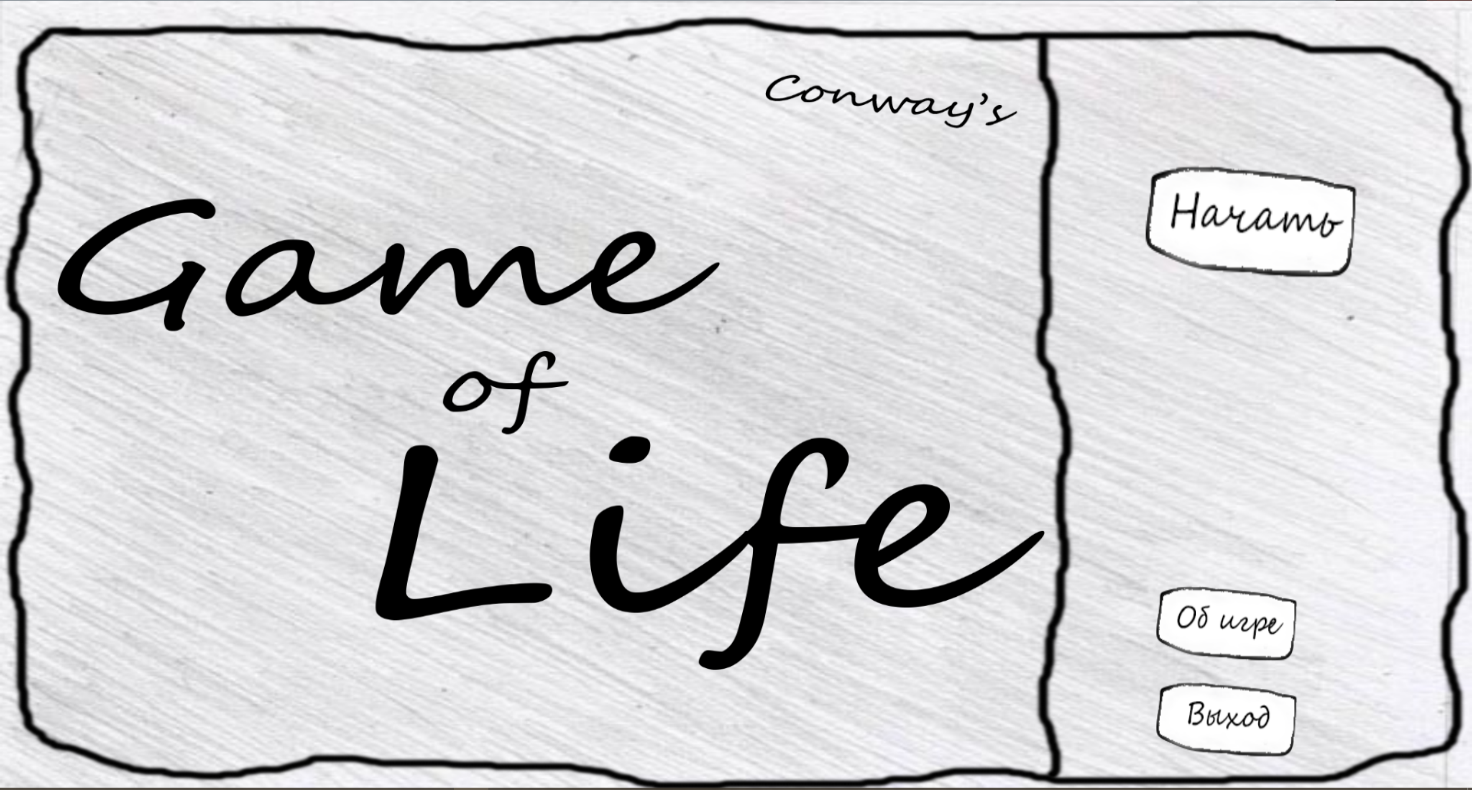
Exit();

}

В методе Draw(), как указывалось ранее, отрисовка на экране всех объектов зависит от текущего состояния игры.

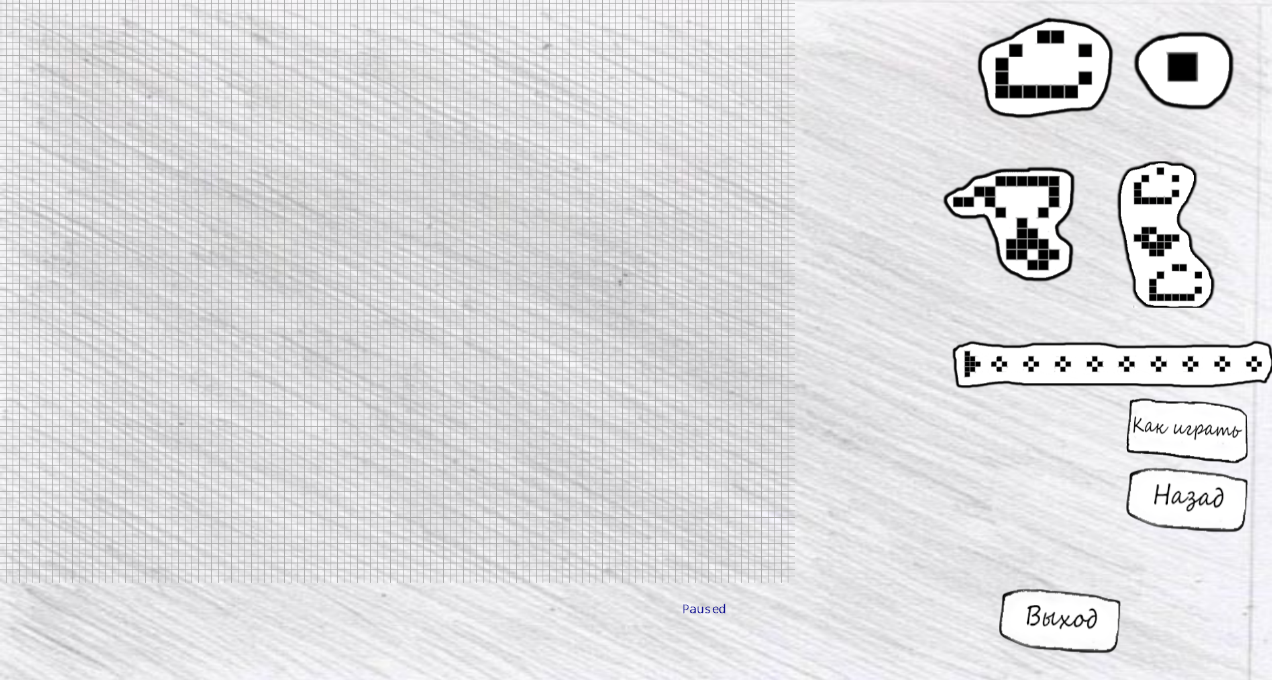
**Пользовательский интерфейс**

После запуска игры открывается главное меню:

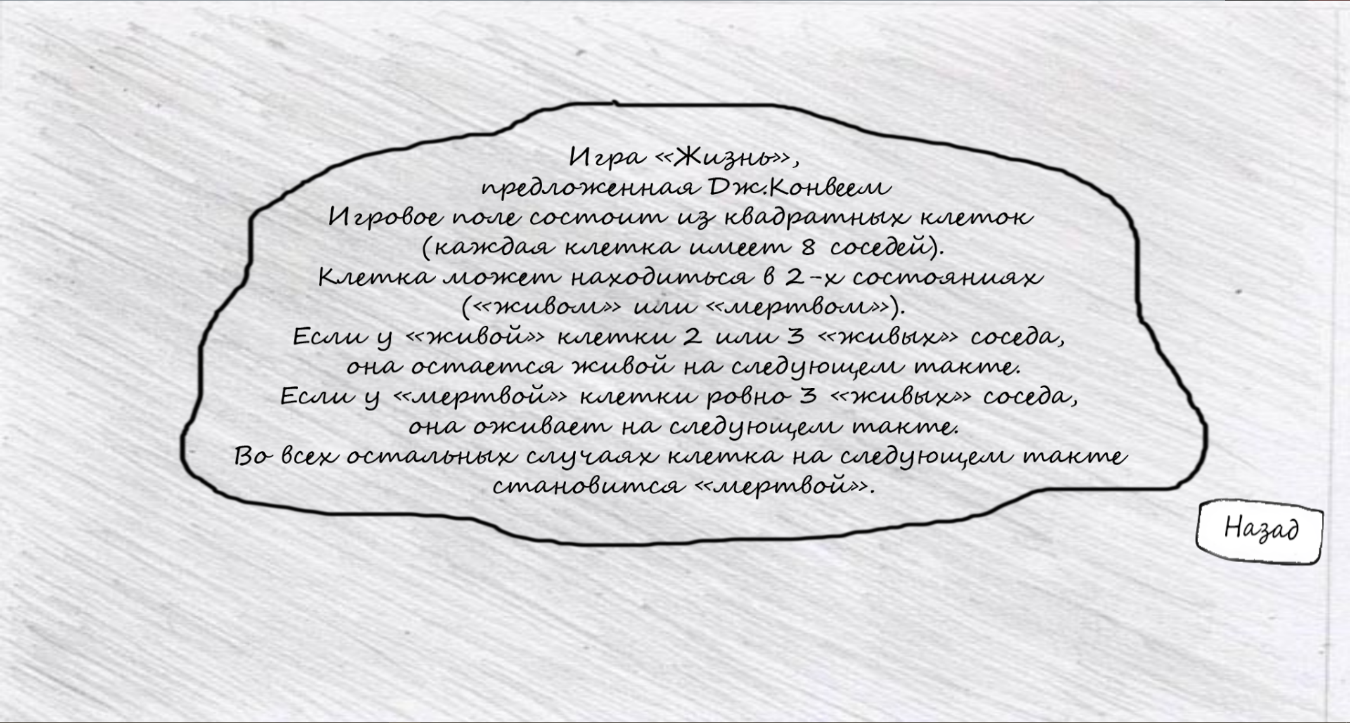


Меню состоит из названия игры("Conway's Game of Life") и трех кнопок:

* Начать: Открывает игровое поле.

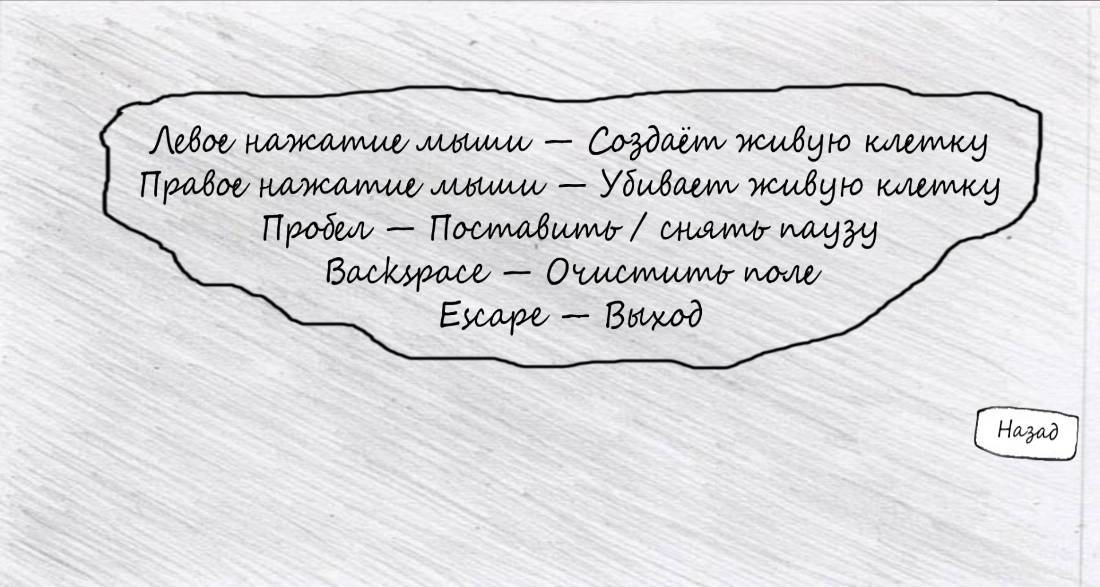


* Об игре: Дает пользователю информацию о данной игре.



* Выход: Завершает выполнение программы.

В главном состоянии игры, где пользователь может создавать "формы жизни", имеется 8 кнопок, 5 из которых отвечают за форму изображения клеток на сетке. "Выход", как и в главном меню, отвечает за завершение программы. "Назад" возвращает пользователя в меню. А нажав кнопку "Как играть", пользователь может ознакомится с особенностями управления данной игрой.



**Заключение**

Целью данной курсовой работе являлась разработки игры "Жизнь" Джона Конвея с использованием XNA Game Studio 4.0. Игра должна включать в себя игровое меню, звуковое сопровождение и сетку (с возможностью ручного заполнения клеток), на которой будет разворачиваться "жизнь".

Можно сделать вывод, что эта цель достигнута.

В ходе данной работы я научилась создавать игры на платформе XNA Game Studio 4.0, расширила знания о возможностях использования языка программирования С#.

Хотя игра "Жизнь" состоит из нескольких простых правил, тем не менее она привлекает пристальное внимание ученых. Эта игра и ее модификации повлияла на многие разделы таких точных наук, как математика, информатика, физика. Кроме того, многие закономерности, обнаруженные в игре, имеют свои аналогии в других (иногда совершенно "нематематических") дисциплинах. Эти факты подчеркивают актуальность изучения этой темы.

На данном этапе разработку игры можно продолжать, например, добавляя в нее различное множество интересных фигур. Также сетку можно сделать замкнутой, что перенесет игру на другой уровень, потому что можно будет получать совершенно другие "формы жизни".

**Список используемой литературы**

1. <https://ru.wikipedia.org/wiki/Жизнь_(игра>)

Википедия. Свободная энциклопедия

2. http://brtrg.by/blog/post/51

3. Рид А. Изучаем XNA 4.0. / Издательство: O'Reilly Media, 2010