ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ
Юго-Западный административный округ
Государственное бюджетное общеобразовательное учреждение города Москвы
«Школа № 1533 «ЛИТ»

ВЫПУСКНОЙ ПРОЕКТ

Учащегося группы 10.5
Барымова Ильи Александровича
(Специальность: программирование)

КЛЕТОЧНЫЙ АВТОМАТ НА ГРАНЯХ ДОДЕКАЭДРА

Заказчик и руководитель Гиглавый Александр Владимирович

Содержание

Введение

В XX веке венгеро-американский математик Джош фон Нейман работал над проблемой самовоспроизводящихся систем. Первоначальная концепция фон Неймана основывалась на идее робота, собирающего другого робота. Разработав эту модель, фон Нейман осознал сложность создания самовоспроизводящегося робота и, в частности, обеспечения необходимого «запаса частей», из которого должен строиться робот. В то же время Станислав Улам изучал рост кристаллов, используя простую решёточную модель. Улам предложил фон Нейману использовать более абстрактную математическую модель, подобную той, что Улам использовал для изучения роста кристаллов. Так и появился первый в истории клеточный автомат.

Клеточные автоматы - дискретные модели, идеально подходящие для решения множества задач в разных областях науки. Такие модели изучаются в математике, теоретической биологии и физике.

Одним из самых известных клеточных автоматов является игра «Жизнь». Его в 1970-ом году придумал британский математик Джон Хортон Конвей. В его игре клеточный автомат определен на поле, состоящем из квадратов, которые являются клетками. Каждая клетка может иметь два состояния: живая или мертвая. Живая клетка окрашена в черный цвет, а мертвая в белый. Клетки могут изменять свои состояние в зависимости от условий, которые придумал Конвей. Он смог подобрать правила игры «Жизнь» так, что начальные конфигурации даже из небольшого количества клеток развиваются зачастую совершенно непредсказуемо. Также он обнаружил, что на поле игры могут существовать фигуры, который способны сохранять форму, стабильно перемещаться или стабильно размножаться.

Игра «Жизнь» была создана в двумерном пространстве, ограниченном с четырех сторон. Поэтому автору в качестве выпускной работы было предложено перенести игру Конвея на правильный многогранник.

Постановка задачи

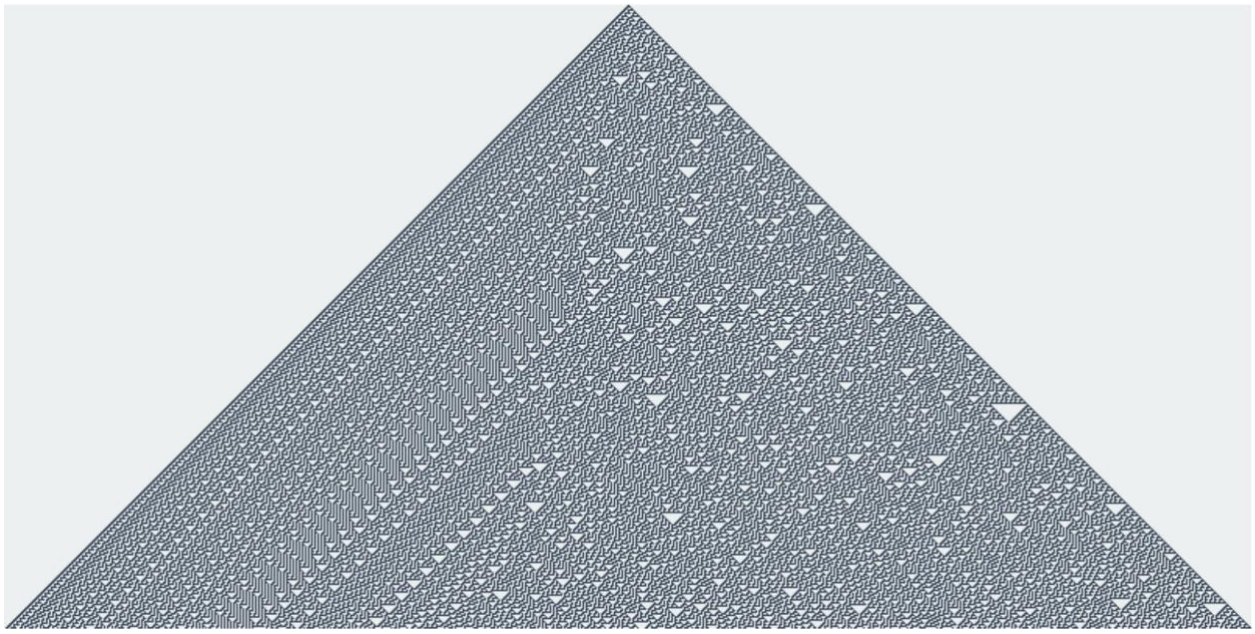
Автору было предложено создать игру «Жизнь» на гранях додекаэдра, т.е. создать клеточный автомат на каждой из грани правильного многогранника, учитывая то, что все грани соединены между собой и соседи клетки не обязаны лежать с ней в одной плоскости.

Также было необходимо изучить получившуюся программу и найти в клеточном автомате фигуры, которые способны закономерно изменять свою форму и расположение, подобно тем, что нашел Конвей.

Актуальность

Забегая вперед, скажу, что простейшие клеточные автоматы используются в криптографии, моделировании физических процессов, поведения людей, в биологии, и в целой куче других важных и интересных сфер. Клеточные автоматы могу, например, помочь повысить эффективность работы с базами данных. Клеточные автоматы можно встретить и в жизни.

Это – Текстильный конус, одно из самых опасных созданий на земле. В Ноттингемском университете внимательно рассмотрели его панцирь и обнаружили, что рисунок образован по принципу клеточного автомата.



Так выглядит раскраска Текстильного конуса в клеточном автомате. Она была образована по правилу под номером 30.

Правило 30

Текущее состояние	111	110	101	100	011	010	001	000
Новое состояние центральной клетки	0	0	0	1	1	1	1	0

Одним словом, клеточные автоматы встречаются и широко применимы в абсолютно разных сферах. За счет того, что созданный нами клеточный автомат является замкнутой фигурой, то есть не имеет границ, его можно применять для изучения эволюции живых существ в замкнутых системах.

Обзор предшествующих решений

Существует огромное множество различных клеточных автоматов, так как в настоящее время клеточно-автоматная теория очень популярна для решения всевозможных задач. Они могут быть одномерными, двумерными, трёхмерными и т.д. В данном проекте рассматривается двумерная модель КА.

Наверное, самым популярным двумерным клеточным автоматом является раннее упомянутая игра «Жизнь». В доступных в интернете примерах игры жизнь пользователь выставляет клетки на поле и наблюдает за эволюцией. Ему дается возможность выставлять на поле ранее заготовленные фигуры или распределять живые клетки случайным образом. Однако я считаю, что это игре сильно не хватает возможности выбора условий, по которым клетки меняют свои состояния. Если добавить эту функцию, то возможности клеточного автомата, разновидности эволюций и количество интересных конфигураций возрастут в разы.

Наша программа работает по принципу игры Конвея, но есть несколько преимуществ.

Во-первых, у пользователя есть возможность выбора условия изменения состояния клетки. Пользователю предлагается 3 варианта «жизни и смерти» клетки: сложный, средний и легкий. При каждом из вариантов выбранная конфигурация будет вести себя по-разному.

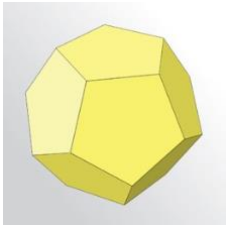
Во-вторых, поле клеточного автомата является замкнутым, т.е. у него нет границ. В игре Конвея клетки, расположенные у границ поля, изменяют свое состояние, не учитывая соседей, расположенных на за границами поля. В нашей программе у каждой клетки есть сосед, не лежащий в одной плоскости с данной клеткой. При изменении состояния клеток такие соседи учитываются и влияют на эволюцию.

В-третьих, наша программа была создана, для переноса игры Конвея на объемную фигуру – додекаэдр, то есть наш автомат можно перенести на правильны многогранник, чтобы можно было наблюдать за эволюцией клеточного автомата на поверхности объемного тела.

Анализ предметной области

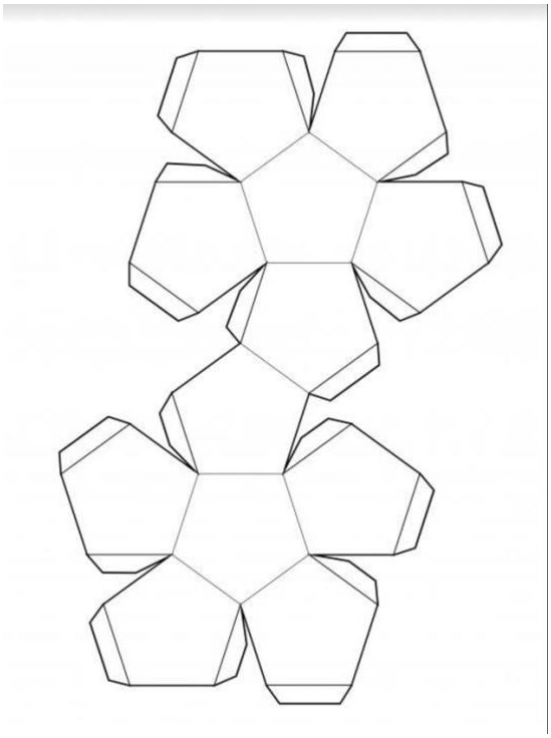
Автором реализована игра «Жизнь» на поверхности правильного многогранника– додекаэдра.

Правильный многогранник - это выпуклый многогранник, состоящий из одинаковых правильных многоугольников и обладающий пространственной симметрией. Такие фигуры также принято называть «Платоновыми телами». Правильные многогранники упоминаются в философии Платона, в честь которого и получили название «Платоновы тела». Платон писал о них в своём трактате «Тимей» (360г до н. э.), где сопоставил каждую из четырёх стихий (землю, воздух, воду и огонь) определённому правильному многограннику. Земля сопоставлялась кубу, воздух — октаэдру, вода — икосаэдру, а огонь — тетраэдру. По поводу пятого элемента, додекаэдра, Платон сделал такое замечание: «…его бог определил для Вселенной и прибегнул к нему в качестве образца». Додекаэдр состоит из двенадцати правильных пятиугольников, которые являются его гранями. Он имеет 12 граней (пятиугольных), 30 рёбер и 20 вершин (в каждой сходятся 3 ребра).



Для наглядности визуализации автор реализовал работу клеточного автомата на развертке додекаэдра.

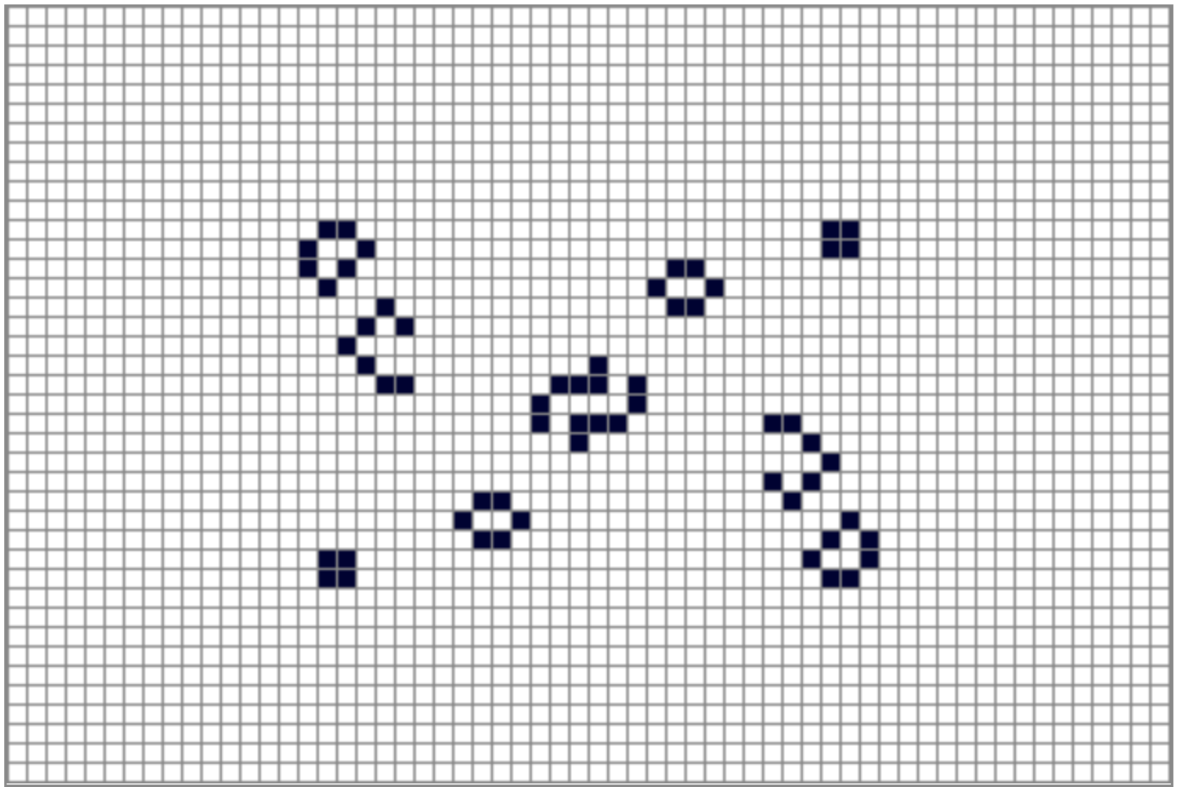
Развёртка поверхности — фигура, получающаяся в плоскости при таком совмещении точек данной поверхности с этой плоскостью, при котором длины линий остаются неизменными.



Как мы видим, развертка додекаэдра состоит из 12-ти правильных питяугольников.

Игра «Жизнь» - клеточный автомат, который был придуман в 1970 году английским математиком Джоном Конвеем.

При запуске игры перед игроком появляется поле, размеченное сеткой. Каждое отделение сетки называется «клеткой». Клетка может иметь два состояния: живая или мертвая. У каждой клетки имеется 8 соседей. Игрок распределяет живые клетки на пустом поле, то есть задает первое поколение. Следующие поколения рассчитываются относительно количества «живых» соседей у каждой клетки. Если у клетки есть ровно 3 «живых соседа», то в ней зарождается жизнь, если клетка является живой и у нее есть 2-3 живых соседа, то жизнь в ней продолжает существовать, иначе клетка будет мертвой.



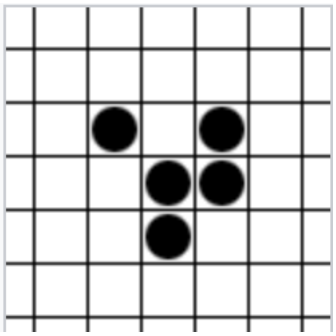
После опубликования правил было обнаружено несколько интересных шаблонов.

Шаблоны (фигуры) - вариантов расстановки живых клеток в первом поколении. Некоторые такие шаблоны остаются неизменными во всех последующих поколениях, состояние других периодически повторяется, в некоторых случаях со смещением всей фигуры. Сейчас существует классификация таких фигур:

1. Устойчивые фигуры - фигуры, которые остаются неизменными;
2. Долгожители - фигуры, которые долго меняются, прежде чем стабилизироваться;
3. Периодические фигуры - фигуры, у которых состояние повторяется через некоторое число поколений, превышающее одно;
4. Двигающиеся фигуры - фигуры, у которых состояние повторяется, но с некоторым смещением;
5. Ружья - фигуры с повторяющимися состояниями, дополнительно создающие движущиеся фигуры;
6. Паровозы - двигающиеся фигуры с повторяющимися состояниями, которые оставляют за собой другие фигуры в качестве следов;
7. Пожиратели - устойчивые фигуры, которые могут пережить столкновения с некоторыми двигающимися фигурами, уничтожив их;
8. Отражатели - устойчивые или периодические фигуры, способные при столкновении с ними движущихся фигур поменять их направление;

9. Размножители - конфигурации, количество живых клеток в которых растёт как квадрат количества шагов;

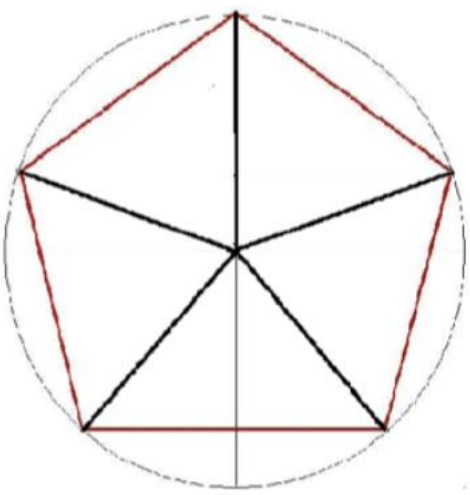
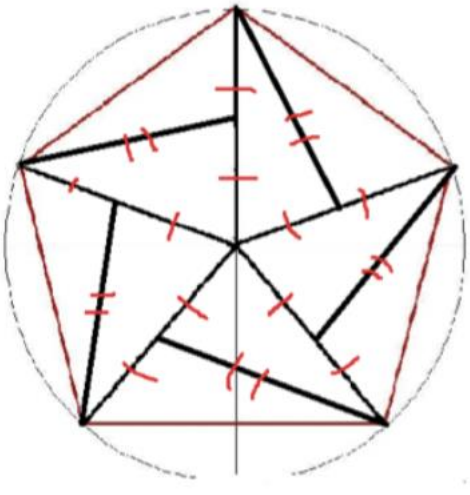
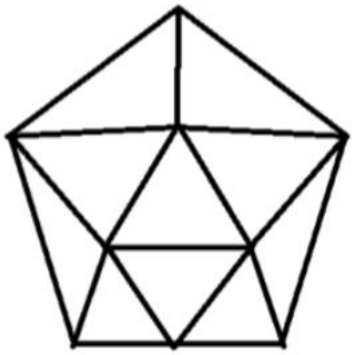
Одним из самых интересных фигур является «планер». Данный шаблон относится к классификации «Движущиеся фигуры». Он был придуман группой из Массачусетского технологического института.



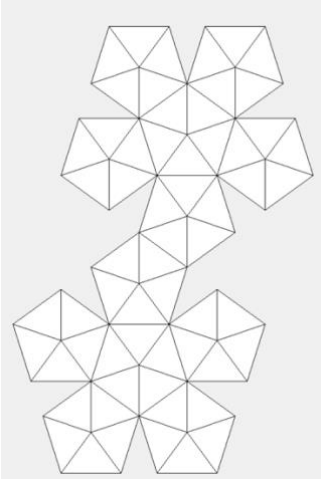
Реализация

Задача была реализована в программе Microsoft Visual Studio Windows Forms на языке программирования C#.

Сам клеточный автомат был перенесен на развертку додекаэдра. Как и было сказано ранее, развертка додекаэдра состоит из 12 правильных пятиугольников. Чтобы увеличить количество клеток, тем самым разнообразить варианты начальных конфигураций и эволюций, каждый из пятиугольников был разбит на треугольники. Автору было предложено на выбор четыре вида разбиения.


Для реализации был выбран первый вариант разбиения, так как он состоит из пяти правильных и равных треугольников, которые имеют общую вершину в центре пятиугольника. Такое разбиение является наиболее подходящим, потому что все клетки на развертке буду выглядеть одинаково и пользователю будет проще проследить за эволюцией. Развертка в программе выглядит следующим образом:



Чтобы нарисовать поле клеточного автомата, был написан класс Edge.cs. Для работы с клетками был написан класс Triangle.cs, содержащий все необходимые свойства для реализации эволюции.

В данной задачей рассматривается клеточный автомат, состоящий из 60 клеток, каждая из которых имеет по три соседа, то есть на будущее состояние фокусной клетки будут влиять состояния трех соседей с каждой стороны. Пользователь может выбрать условия, по которому будут изменяться состояния клеток. Всего есть 3 варианта условий:

- «Hard» - в клетке зарождается и сохраняется жизнь, если у нее есть ровно 2 живых соседа.

- «Medium» - в клетке зарождается и сохраняется жизнь, если у нее есть 1 или 2 живых соседа.

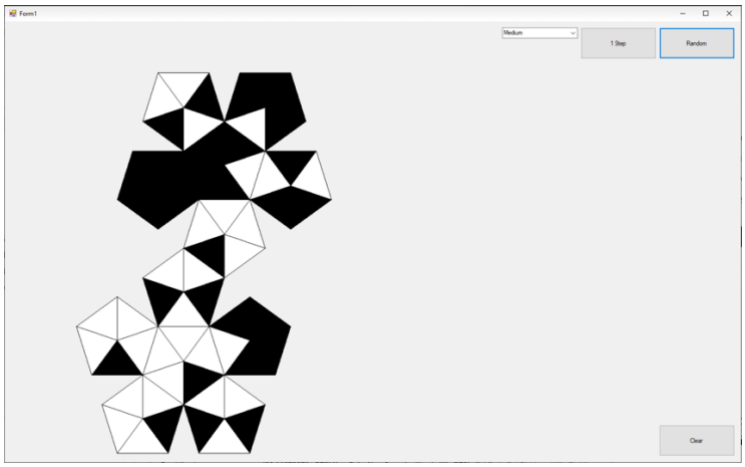
- «Easy» - в клетке зарождается и сохраняется жизнь, если у нее есть хотя бы один живой сосед.

Также у пользователя есть возможность очистить все поле от живых клеток или распределить ровно 30 живых клеток на поле в случайном порядке с помощью кнопок.

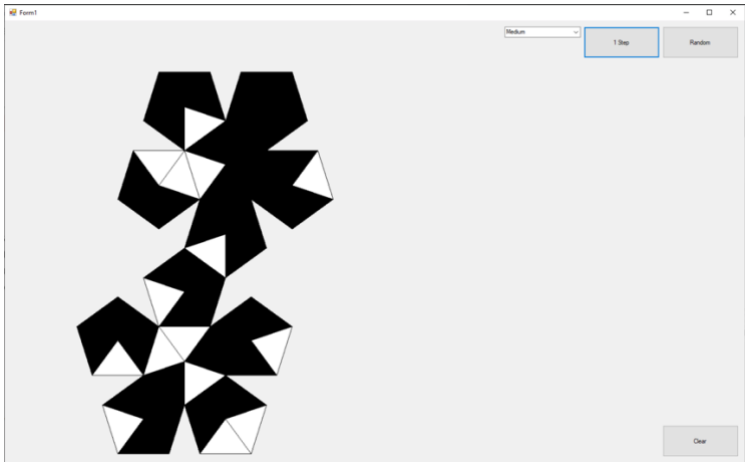
Пример работы программы

Чтобы показать работу программы на примере, я расставил 30 живых клеток на поле, с помощью кнопки «Random», и зафиксировал 3 шага революции, чтобы продемонстрировать их.

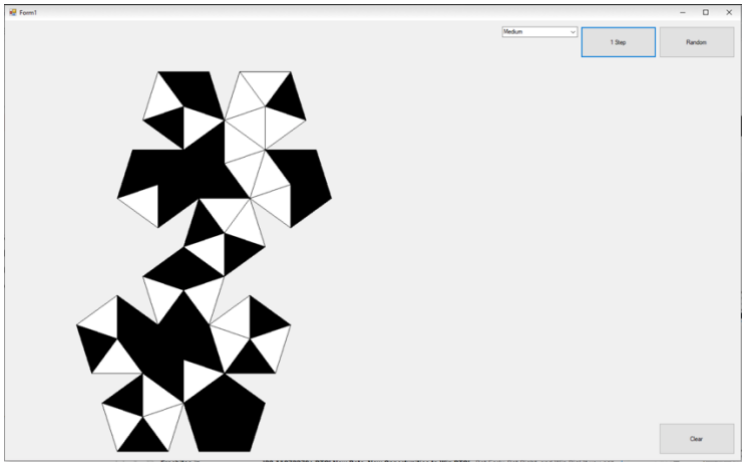
Первый шаг:



Второй шаг:



Третий шаг:



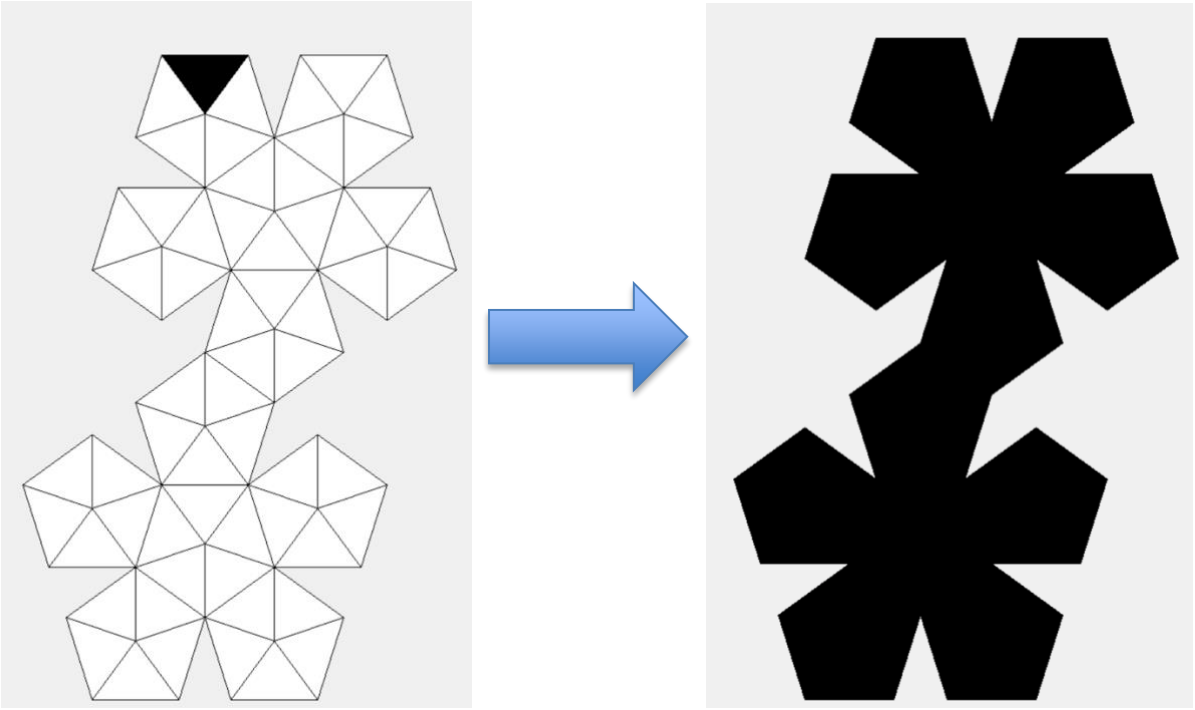
Результат

Написав программу и протестировав ее, мы обнаружили несколько интересных конфигураций при разных «условиях жизни и смерти».

«Easy»:

Мы заметили, что если выбрать вариант «условия жизни и смерти» под названием «Easy», то все поле закрасится в черный цвет, если на нем есть хотя бы одна живая клетка. От количества живых клеток в начальной конфигурации зависит только количество ходов, нужное чтобы все поле стало черным.

Пример:

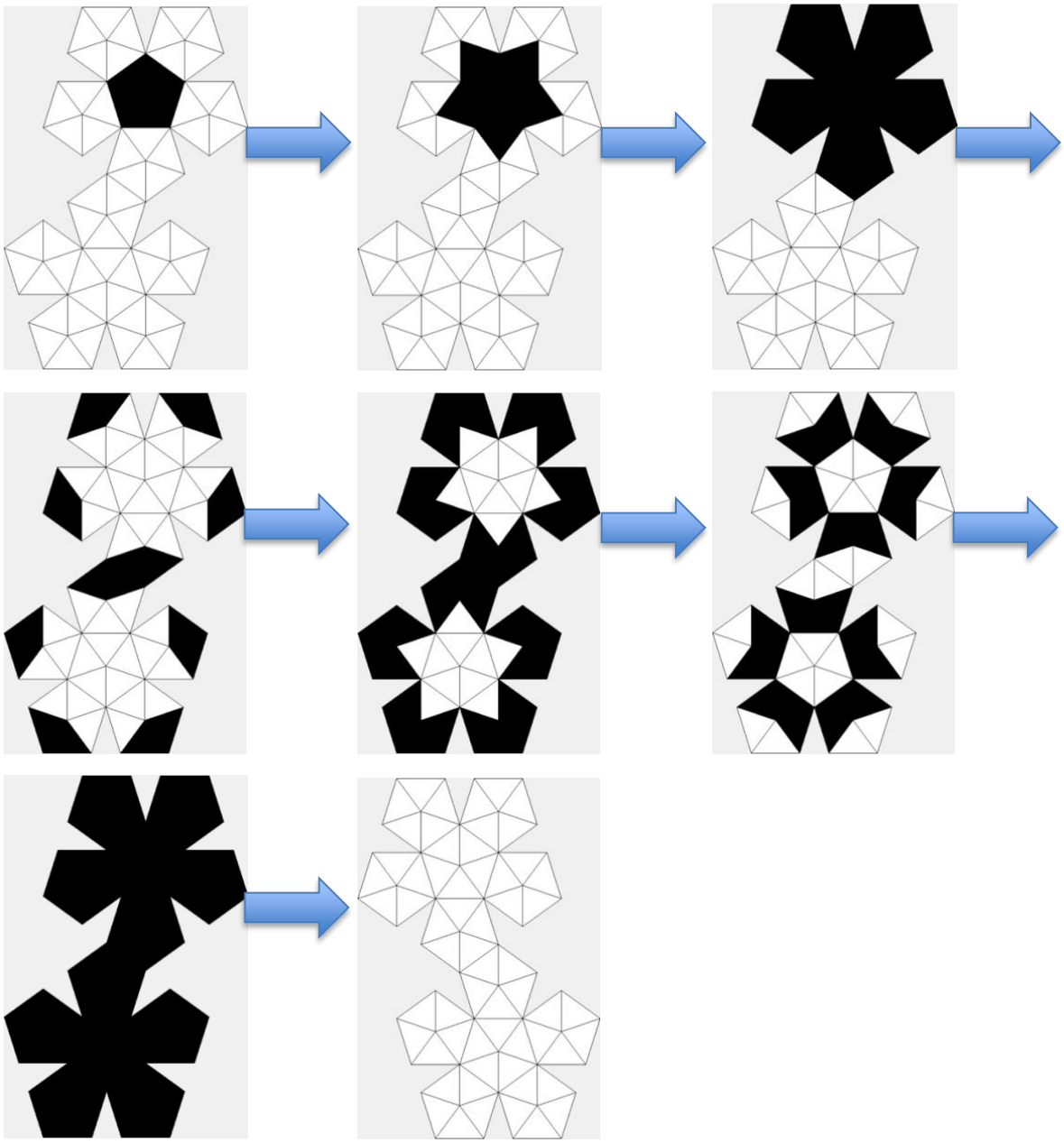


«Medium»:

Самым интересным «условием жизни и смерти» является «Medium». При нем клеточный автомат имеет наибольшее количество вариантов эволюции. При таком условии было найдено несколько интересных конфигураций.

Одно из самых, на наш взгляд, красивых начальных конфигураций является так называемая «Звезда». Чтобы задать такую конфигурацию, нужно закрасить все клетки верхней или нижней грани додекаэдра. В таком случае все поле окрасится в черный цвет после нескольких ходов, а потом опустеет. На каждом ходу будут появлятся симметричные фигуры.

Пример:

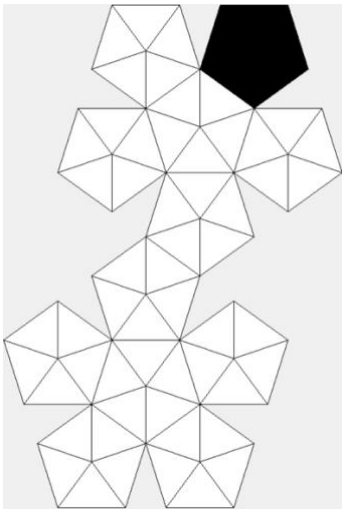


«Hard»:

При «условии жизни и смерти», которое называется «Hard», поле почти всегда будет пустым в итоге. Однако, есть некоторые интересные конфигурации, которые заслуживают внимания.

Одной из таких является «Точка». Она задается так же, как и «Звезда» при условии «Medium», результат будет другим. Точка остается неизменной на протяжении бесконечного количества ходов. Такая конфигурация не меняет ни своего расположения, ни своей формы.

Пример:



Заключение

В заключении хочется сказать, клеточные автоматы будут оставаться актуальными, так как они упрощают сложные математические модели, могут привести к решению разных вопросов в окружающем мире. Например, с помощью клеточного автомата «Жизнь» возможно усовершенствование и проведение исследований по "Искусственной жизни" (Artificial Life), а это уже разработка в дальнейшем искусственного интеллекта.

Однако, что касается нашей разработки, остается только усложнять данную программу, увеличивая количество клеток, находить интересные конфигурации, переносить клеточный автомат в трехмерную реальность. Усложненная программа будет применима в больших сферах.

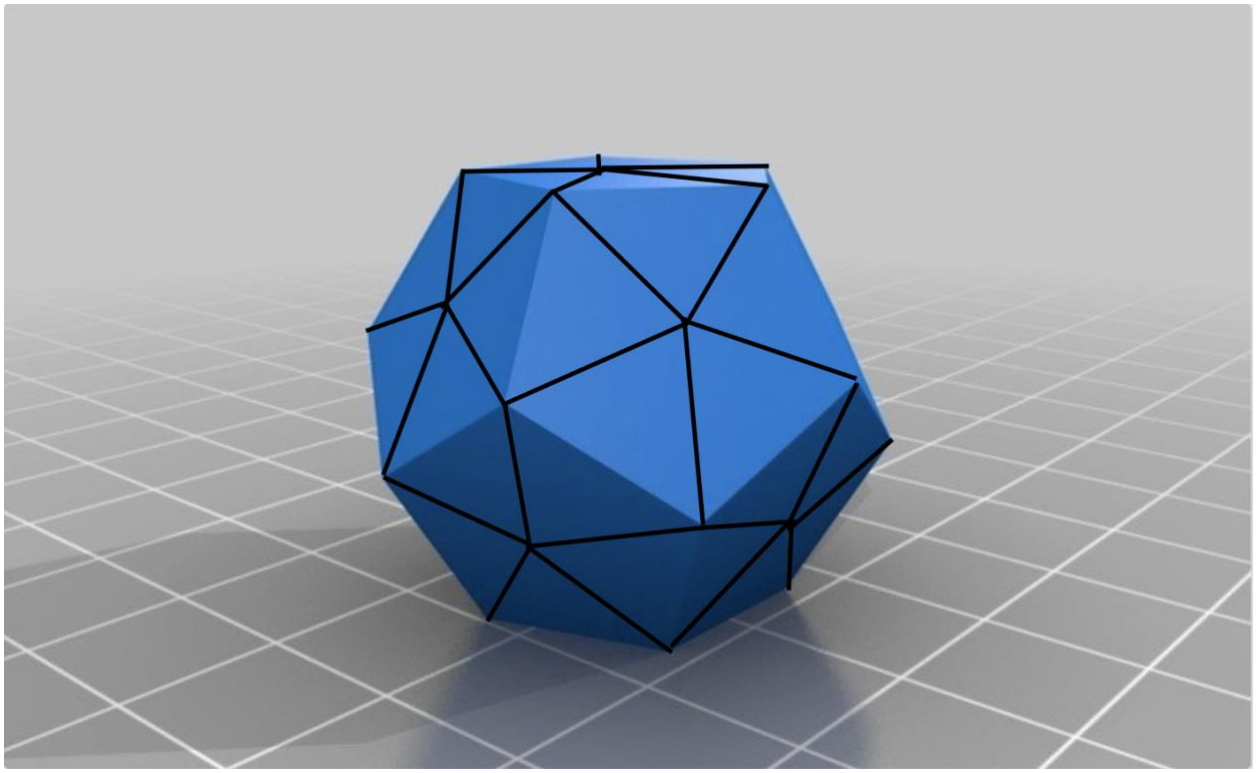
Перспективы

Одно из направлений дальнейших разработок - усложнение уже существующих моделей. Возможно увеличить количество клеток вдвое, если провести в каждом из треугольников высоты, выходящую из центра пятиугольника. В таком случае поле будет состоять из 120 равных прямоугольных треугольников.

Также можно перенести клеточный автомат из развертки в трехмерное пространство с помощью 3D редакторов.

Одним из сложнейших направления дальнейших разработок будет перенос клеточного автомата в реальную жизнь. Если внутрь додекаэдра вставить светодиоды, которые будут загораться и погасать в зависимости от состояния клетки.

Пример 3D модели, на поверхность которой можно перенести автомат:



Список литературы

- Википедия: Конвей, Джон Хортон - https://ru.wikipedia.org/wiki/Конвей,_Джон_Хортон#Клеточные_автоматы
- Википедия: Игра «Жизнь» - https://ru.wikipedia.org/wiki/Игра_«Жизнь»
- Хабр: Простейшие клеточные автоматы и их практическое применение - https://habr.com/ru/post/273393/
- Хабр: Поиграем в жизнь - https://habr.com/ru/post/63848/
- Википедия: Правильный многогранник - https://ru.wikipedia.org/wiki/Правильный_многогранник

Приложения

Класс Triangle.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Drawing;
using System.Text;
using System.Threading.Tasks;

namespace Life
{
   class Triangle
   {
      public Point pt1;
      public Point pt2;
      public Point pt3;
      public bool flag;
      public bool type;
      public Color c;
      public int num;
      public int[] neighbors;
      public int[] N(int a, int b, int c)
      {
         int[] N = new int[3];
         N[0] = a;
         N[1] = b;
         N[2] = c;
         return N;
      }
      public Triangle(Point pt1, Point pt2, Point pt3, int num)
      {

         this.pt1 = pt1;
         this.pt2 = pt2;
         this.pt3 = pt3;
         flag = false;
         type = false;
         this.num = num;
         if (num == 1)
         {
            this.neighbors = N(2, 5, 54);
         }
         if (num == 2)
         {
            this.neighbors = N(1, 3, 10);
         }
         if (num == 3)
         {
            this.neighbors = N(2, 4, 15);
         }
         if (num == 4)
         {
            this.neighbors = N(5, 3, 16);
         }
         if (num == 5)
         {
            this.neighbors = N(1, 4, 44);
         }
         if (num == 6)
         {
            this.neighbors = N(7, 10, 53);
         }
         if (num == 7)
```

```
{
   this.neighbors = N(6, 8, 58);
}
if (num == 8)
{
   this.neighbors = N(7, 9, 21);
}
if (num == 9)
{
   this.neighbors = N(8, 9, 11);
}
if (num == 10)
{
   this.neighbors = N(2, 6, 9);
}
if (num == 11)
{
   this.neighbors = N(9, 15, 12);
}
if (num == 12)
{
   this.neighbors = N(11, 13, 25);
}
if (num == 13)
{
   this.neighbors = N(12, 14, 26);
}
if (num == 14)
{
   this.neighbors = N(13, 15, 17);
}
if (num == 15)
{
   this.neighbors = N(3, 11, 14);
}
if (num == 16)
{
   this.neighbors = N(4, 17, 20);
}
if (num == 17)
{
   this.neighbors = N(14, 16, 18);
}
if (num == 18)
{
   this.neighbors = N(17, 19, 30);
}
if (num == 19)
{
   this.neighbors = N(18, 20, 35);
}
if (num == 20)
{
   this.neighbors = N(16, 19, 45);
}
if (num == 21)
{
   this.neighbors = N(8, 22, 25);
}
if (num == 22)
{
   this.neighbors = N(21, 23, 57);
}
if (num == 23)
{
   this.neighbors = N(22, 24, 47);
}
if (num == 24)
```

```
{
   this.neighbors = N(23, 25, 27);
}
if (num == 25)
{
   this.neighbors = N(12, 21, 24);
}
if (num == 26)
{
   this.neighbors = N(13, 27, 30);
}
if (num == 27)
{
   this.neighbors = N(24, 26, 28);
}
if (num == 28)
{
   this.neighbors = N(27, 29, 46);
}
if (num == 29)
{
   this.neighbors = N(28, 30, 31);
}
if (num == 30)
{
   this.neighbors = N(18, 26, 29);
}
if (num == 31)
{
   this.neighbors = N(29, 32, 35);
}
if (num == 32)
{
   this.neighbors = N(31, 33, 50);
}
if (num == 33)
{
   this.neighbors = N(32, 34, 36);
}
if (num == 34)
{
   this.neighbors = N(33, 35, 41);
}
if (num == 35)
{
   this.neighbors = N(19, 31, 34);
}
if (num == 36)
{
   this.neighbors = N(33, 37, 40);
}
if (num == 37)
{
   this.neighbors = N(36, 38, 49);
}
if (num == 38)
{
   this.neighbors = N(37, 39, 60);
}
if (num == 39)
{
   this.neighbors = N(38, 40, 51);
}
if (num == 40)
{
   this.neighbors = N(36, 39, 42);
}
if (num == 41)
```

```
{
   this.neighbors = N(34, 42, 45);
}
if (num == 42)
{
   this.neighbors = N(40, 41, 43);
}
if (num == 43)
{
   this.neighbors = N(42, 44, 55);
}
if (num == 44)
{
   this.neighbors = N(43, 45, 5);
}
if (num == 45)
{
   this.neighbors = N(41, 44, 20);
}
if (num == 46)
{
   this.neighbors = N(28, 47, 50);
}
if (num == 47)
{
   this.neighbors = N(23, 46, 48);
}
if (num == 48)
{
   this.neighbors = N(47, 49, 56);
}
if (num == 49)
{
   this.neighbors = N(37, 48, 50);
}
if (num == 50)
{
   this.neighbors = N(46, 49, 32);
}
if (num == 51)
{
   this.neighbors = N(39, 52, 55);
}
if (num == 52)
{
   this.neighbors = N(51, 53, 59);
}
if (num == 53)
{
   this.neighbors = N(6, 52, 54);
}
if (num == 54)
{
   this.neighbors = N(53, 55, 1);
}
if (num == 55)
{
   this.neighbors = N(43, 51, 54);
}
if (num == 56)
{
   this.neighbors = N(48, 57, 60);
}
if (num == 57)
{
   this.neighbors = N(22, 56, 58);
}
if (num == 58)
```

```
            {
                this.neighbors = N(7, 57, 59);
            }
            if (num == 59)
            {
                this.neighbors = N(52, 58, 60);
            }
            if (num == 60)
            {
                this.neighbors = N(38, 56, 59);
            }
        }

        public void Fill(Graphics e, Brush color)
        {
            Point[] pts = new Point[3];
            pts[0] = pt1;
            pts[1] = pt2;
            pts[2] = pt3;
            e.FillPolygon(color, pts);
        }
        public void Draw(Graphics e, Pen color)
        {
            Point[] pts = new Point[3];
            pts[0] = pt1;
            pts[1] = pt2;
            pts[2] = pt3;
            e.DrawPolygon(color, pts);
        }
        public void Check(int x, int y)
        {
            int x1 = pt1.X;
            int y1 = pt1.Y;
            int x2 = pt2.X;
            int y2 = pt2.Y;
            int x3 = pt3.X;
            int y3 = pt3.Y;
            if (y1 == y2)
            {
                if (y3 > y1)
                {
                    if (y >= y1 && y <= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y <= x * (y3 - y2) /
(x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
                    {
                        if (type == true)
                        {
                            type = false;
                            flag = false;
                        }
                        else
                        {
                            type = true;
                            flag = true;
                        }
                    }
                }
                else
                {
                    if (y <= y1 && y >= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y >= x * (y3 - y2) /
(x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
                    {
                        if (type == true)
                        {
                            type = false;
                            flag = false;
                        }
                        else
                        {
```

```
                    type = true;
                    flag = true;
                }
            }
        }
    }
    else if (x3 == x1)
    {
        if (x2 < x3)
        {
            if (x <= x3 && y <= x * (y2 - y1) / (x2 - x1) - ((x1 * y2 - x2 * y1) / (x2 - x1)) && y >= x * (y3 - y2) /
(x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
            {
                if (type == true)
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
        }
        else
        {
            if (x >= x3 && y >= x * (y2 - y1) / (x2 - x1) - ((x1 * y2 - x2 * y1) / (x2 - x1)) && y <= x * (y3 - y2) /
(x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
            {
                if (type == true)
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
        }
    }
    else if(x3 == x2)
    {
        if (x1 > x3)
        {
            if (x >= x3 && y >= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y <= x * (y1 - y2) /
(x1 - x2) - ((x2 * y1 - x1 * y2) / (x1 - x2)))
            {
                if (type == true)
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
        }
        else
        {
            if (x <= x3 && y <= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y >= x * (y1 - y2) /
(x1 - x2) - ((x2 * y1 - x1 * y2) / (x1 - x2)))
            {
                if (type == true)
```

```
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
        }
    }
    //
    else if(y3 > y2 && y3 < y1)
    {
        if(x2 > x1)
        {
            if(y <= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y >= x * (y1 - y2) / (x1 - x2) - ((x2
* y1 - x1 * y2) / (x1 - x2)) && y >= x * (y3 - y2) / (x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
            {
                if (type == true)
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
        }
        else
        {
            if (y <= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y <= x * (y1 - y2) / (x1 - x2) -
((x2 * y1 - x1 * y2) / (x1 - x2)) && y >= x * (y3 - y2) / (x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
            {
                if (type == true)
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
        }
    }
    else if(y3 > y1 && y3 < y2)
    {
        if(x2 > x1)
        {
            if(y >= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y >= x * (y1 - y2) / (x1 - x2) - ((x2
* y1 - x1 * y2) / (x1 - x2)) && y <= x * (y3 - y2) / (x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
            {
                if (type == true)
                {
                    type = false;
                    flag = false;
                }
                else
                {
                    type = true;
                    flag = true;
                }
            }
```

```
            }
            else
            {
                if (y >= x * (y3 - y1) / (x3 - x1) - ((x1 * y3 - x3 * y1) / (x3 - x1)) && y <= x * (y1 - y2) / (x1 - x2) -
((x2 * y1 - x1 * y2) / (x1 - x2)) && y <= x * (y3 - y2) / (x3 - x2) - ((x2 * y3 - x3 * y2) / (x3 - x2)))
                {
                    if (type == true)
                    {
                        type = false;
                        flag = false;
                    }
                    else
                    {
                        type = true;
                        flag = true;
                    }
                }
            }
        }
    }
    public int Nbrs(Triangle[] trs, Triangle tr)
    {
        int nbrs = 0;
        for(int i = 0; i < 3; i++)
        {
            if(trs[tr.neighbors[i] - 1].type == true)
            {
                nbrs++;
            }
        }
        return nbrs;
    }
    public bool Step_H(int nbrs)
    {
        if (nbrs == 1 || nbrs == 3 || nbrs == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }

    public bool Step_M(int nbrs)
    {
        if ( nbrs == 3 || nbrs == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    public bool Step_E(int nbrs)
    {
        if (nbrs == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
```

```
    }
}
```

Класс Edge.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Life
{
    class Edge
    {
        public int a;
        private int x;
        private int y;
        private int type;
        public Point pt1;
        public Point pt2;
        public Point pt3;
        public Point pt4;
        public Point pt5;
        public Point cpt;
        public Point[] edgpts;
        public Edge(int a, int x, int y, int type)
        {
            this.a = a;
            this.x = x;
            this.y = y;
            this.type = type;
            if (type == 1)
            {
                pt1 = new Point(x, y);
                pt2 = new Point(x + (int)(a * Math.Sin(Convert.ToDouble(54 / 180.0 * 3.14))), y + (int)(a *
Math.Cos(Convert.ToDouble(54 / 180.0 * 3.14))));
                pt3 = new Point(pt2.X - (int)(a * Math.Sin(Convert.ToDouble(18 / 180.0 * 3.14))), pt2.Y + (int)(a *
Math.Cos(Convert.ToDouble(18 / 180.0 * 3.14))));
                pt4 = new Point(pt3.X - a, pt3.Y);
                pt5 = new Point(pt4.X - (int)(a * Math.Sin(Convert.ToDouble(18 / 180.0 * 3.14))), pt4.Y - (int)(a *
Math.Cos(Convert.ToDouble(18 / 180.0 * 3.14))));
                cpt = new Point(pt1.X, pt1.Y + (int)(a * (Math.Sqrt(10) * Math.Sqrt(5 + Math.Sqrt(5)) / 10)));
            }
            else
            {
                pt1 = new Point(x, y);
                pt2 = new Point(pt1.X + a, pt1.Y);
                pt3 = new Point(pt2.X + (int)(a * Math.Sin(Convert.ToDouble(18 / 180.0 * 3.14))), pt2.Y + (int)(a *
Math.Cos(Convert.ToDouble(18 / 180.0 * 3.14))));
                pt4 = new Point(pt3.X - (int)(a * Math.Sin(Convert.ToDouble(54 / 180.0 * 3.14))), pt3.Y + (int)(a *
Math.Cos(Convert.ToDouble(54 / 180.0 * 3.14))));
                pt5 = new Point(pt4.X - (int)(a * Math.Sin(Convert.ToDouble(54 / 180.0 * 3.14))), pt4.Y - (int)(a *
Math.Cos(Convert.ToDouble(54 / 180.0 * 3.14))));
                cpt = new Point(pt4.X, pt4.Y - (int)(a * (Math.Sqrt(10) * Math.Sqrt(5 + Math.Sqrt(5)) / 10)));
            }
            edgpts = new Point[7];
            edgpts[0] = pt1;
            edgpts[1] = pt2;
            edgpts[2] = pt3;
            edgpts[3] = pt4;
            edgpts[4] = pt5;
            edgpts[5] = pt1;
            edgpts[6] = cpt;
        }
        public int X
        {
            get { return x; }
```

```
        }
        public void Draw(Graphics e)
        {
            e.DrawLine(Pens.Black, pt1, pt2);
            e.DrawLine(Pens.Black, pt2, pt3);
            e.DrawLine(Pens.Black, pt3, pt4);
            e.DrawLine(Pens.Black, pt4, pt5);
            e.DrawLine(Pens.Black, pt5, pt1);
            e.DrawLine(Pens.Black, pt1, cpt);
            e.DrawLine(Pens.Black, pt2, cpt);
            e.DrawLine(Pens.Black, pt3, cpt);
            e.DrawLine(Pens.Black, pt4, cpt);
            e.DrawLine(Pens.Black, pt5, cpt);
          /*List<Point> p = new List<Point>();

          for (int i = 0; i < 360; i += 72)
          {


            double rad = (double)i / 180.0 * 3.14;
            int x = 800 + (int)(R * Math.Cos(rad));
            int y = (int)(R * Math.Sin(rad));

            p.Add(new Point(x + R, y + R));
          }
          e.DrawPolygon(Pens.Black, p.ToArray());*/
        }
    }
}
```

Класс Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Life
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        Edge edg1;
        Edge edg2;
        Edge edg3;
        Edge edg4;
        Edge edg5;
        Edge edg6;
        Edge edg7;
        Edge edg8;
        Edge edg9;
        Edge edg10;
        Edge edg11;
        Edge edg12;
        Edge[] edgs;
        Triangle[] trs;
        private void Form1_Load(object sender, EventArgs e)
        {
            DoubleBuffered = true;

            edgs = new Edge[12];
```

```
        edg1 = new Edge(100, 300, 100, 2);
        edgs[0] = edg1;
        edg2 = new Edge(100, edg1.pt2.X + 2 * (int)(edg1.a * Math.Sin(Convert.ToDouble(18 / 180.0 * 3.14))) ,
100, 2);
        edgs[1] = edg2;
        edg3 = new Edge(100, edg1.pt3.X , edg1.pt4.Y - (int)(edg1.a * Math.Sin(Convert.ToDouble(36 / 180.0 *
3.14))), 1);
        edgs[2] = edg3;
        edg4 = new Edge(100, edg3.pt1.X - (int)(edg1.a * Math.Cos(Convert.ToDouble(36 / 180.0 * 3.14))) -
edg1.a, edg3.pt5.Y, 2);
        edgs[3] = edg4;
        edg5 = new Edge(100, edg3.pt2.X, edg4.pt1.Y, 2);
        edgs[4] = edg5;
        edg6 = new Edge(100, edg3.pt4.X, edg3.pt4.Y, 2);
        edgs[5] = edg6;
        edg7 = new Edge(edg1.a, edg6.pt5.X, edg6.pt5.Y, 1);
        edgs[6] = edg7;
        edg8 = new Edge(edg1.a, edg7.pt4.X, edg7.pt4.Y, 2);
        edgs[7] = edg8;
        edg9 = new Edge(edg1.a, edg8.pt1.X - (int)(edg1.a * Math.Sin(Convert.ToDouble(54 / 180.0 * 3.14))),
edg8.pt1.Y - (int)(edg1.a * Math.Sin(Convert.ToDouble(36 / 180.0 * 3.14))), 1);
        edgs[8] = edg9;
        edg10 = new Edge(edg1.a, edg8.pt2.X + (int)(edg1.a * Math.Sin(Convert.ToDouble(54 / 180.0 * 3.14))),
edg9.pt1.Y, 1);
        edgs[9] = edg10;
        edg11 = new Edge(edg1.a, edg8.pt5.X, edg8.pt5.Y, 1);
        edgs[10] = edg11;
        edg12 = new Edge(edg1.a, edg8.pt3.X, edg8.pt3.Y, 1);
        edgs[11] = edg12;
        trs = new Triangle[60];

        int t = 0;
        for(int i = 0; i < 12; i++)
        {

            for(int y = 0; y < 5; y++)
            {
                trs[t] = new Triangle(edgs[i].edgpts[y], edgs[i].edgpts[y + 1], edgs[i].cpt, t + 1);
                t++;
            }
        }

    }
    protected override void OnPaint(PaintEventArgs e)
    {
        edg1.Draw(e.Graphics);
        edg2.Draw(e.Graphics);
        edg3.Draw(e.Graphics);
        edg4.Draw(e.Graphics);
        edg5.Draw(e.Graphics);
        edg6.Draw(e.Graphics);
        edg7.Draw(e.Graphics);
        edg8.Draw(e.Graphics);
        edg9.Draw(e.Graphics);
        edg10.Draw(e.Graphics);
        edg11.Draw(e.Graphics);
        edg12.Draw(e.Graphics);
        for (int i = 0; i < 60; i++)
        {
            if (trs[i].flag == true)
            {
                trs[i].Fill(e.Graphics, Brushes.Black);
            }
            else if(trs[i].flag == false)
            {
                trs[i].Fill(e.Graphics, Brushes.White);
                trs[i].Draw(e.Graphics, Pens.Black);
            }
```

```
    }
}

private void Form1_MouseClick(object sender, MouseEventArgs e)
{

}

private void Form1_Paint(object sender, PaintEventArgs e)
{

}

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    for (int i = 0; i < 60; i++)
    {
    trs[i].Check(e.X, e.Y);

    }
    Refresh();
}

private void button1_Click(object sender, EventArgs e)
{
    for(int i = 0; i < 60; i++)
    {
        trs[i].type = false;
        trs[i].flag = false;
    }
    Random rnd = new Random();
    int y;
    for(int i = 0; i < 30; i++)
    {
        y = rnd.Next(0,59);
        if(trs[y].type == false)
        {
            trs[y].type = true;
            trs[y].flag = true;
        }
        else
        {
            while(trs[y].flag != false)
            {
                y = rnd.Next(0, 59);
            }
            trs[y].type = true;
            trs[y].flag = true;
        }
    }
    Refresh();
}

private void button2_Click(object sender, EventArgs e)
{
    for (int i = 0; i < 60; i++)
    {
        trs[i].type = false;
        trs[i].flag = false;
    }
    Refresh();
}

private void button3_Click(object sender, EventArgs e)
{
    int y;
    bool[] temp = new bool[60];
    for (int i = 0; i < 60; i++)
```

```
            {
                temp[i] = trs[i].type;
            }
            if (comboBox1.Text == "Hard")
            {
                for (int i = 0; i < 60; i++)
                {
                    temp[i] = trs[i].Step_H(trs[i].Nbrs(trs, trs[i]));
                }
            }
            else if(comboBox1.Text == "Medium")
            {
                for (int i = 0; i < 60; i++)
                {
                    temp[i] = trs[i].Step_M(trs[i].Nbrs(trs, trs[i]));
                }
            }
            else if(comboBox1.Text == "Easy")
            {
                for (int i = 0; i < 60; i++)
                {
                    temp[i] = trs[i].Step_E(trs[i].Nbrs(trs, trs[i]));
                }
            }
            for (int i = 0; i < 60; i++)
            {
                trs[i].type = temp[i];
                trs[i].flag = temp[i];
            }
            Refresh();
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {

        }
    }
}
```