Региональная конференция по научно-техническому творчеству школьников «Лабиринты науки»



Государственное общеобразовательное учреждение «Средняя школа «Провинциальный колледж»

Игра «Жизнь»: программирование математической игры на гексагональной решётке с помощью средств языка Python

Исследовательская работа

Выполнена учениками 11 информационно-технологического класса Государственного общеобразовательного учреждения «Средняя школа «Провинциальный колледж» **Игнатьевым Максимом Юрьевичем и Кудряшовой Анной Александровной**

Научный руководитель – старший преподаватель кафедры сетевых и информационных технологий ЯрГУ им. П. Г. Демидова, **Легков Николай Васильевич**

Ярославль, 2021

Оглавление

ВВЕДЕНИЕ **3**

ГЛАВА 1. ПОНЯТИЕ И ПРАВИЛА ИГРЫ «ЖИЗНЬ» КОНВЕЯ **6**

1.1. История создания игры 6

1.2. Правила игры на клетчатом поле 7

1.3. Правила игры на гексагональной решётке 7

1.4. Виды игрового поля 8

1.3.1. Ограниченное поле *8*

1.3.2. Тороид *9*

1.3.3. Цилиндр *9*

1.4. Понятие форм «Жизни» 9

ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ **11**

2.1. Описание реализованной программы 11

2.2. Тестирование алгоритма 11

ЗАКЛЮЧЕНИЕ **13**

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ **14**

ПРИЛОЖЕНИЯ **15**

Приложение 1 15

Приложение 2 20

Введение

Игра «Жизнь» – клеточный автомат, придуманный английским математиком Джоном Конвеем в 1970 году.[1] Эта игра стала известной задачей как для математиков, так и непосредственно для программистов, которые ищут всё новые и новые алгоритмы её реализации. «Жизнь» Конвея пережила немало преображений и сейчас существует множество вариаций правил игры, которые влияют на свойства задачи в значительной степени.

Простейшие правила, заложенные в принцип игры, повлияли на развитие многих наук, таких как математика, информатика и даже биология. Модели, конфигурации на различных типах карт в жизни поражают своим разнообразием. В интернете есть много информации о стандартной версии «Жизни» на клеточном поле, однако на других типах карт, таких как гексагональная решётка, эту задачу изучают значительно реже. Важно, чтобы варианты реализации кода продолжали пополняться. В малоизученности игры «Жизнь» на гексагональной решётке заключается **актуальность** работы. Итак, ставится **проблема** в поиске алгоритма для игры «Жизнь» на гексагональной решётке и в изучении его свойств.

Цель исследовательской работы – запрограммировать математическую игру «Жизнь» Конвея на гексагональной решётке на языке Python.

Задачи исследования:

1. Выявить принципиальные различия игры «Жизнь» на клетчатом поле и на гексагональной решётке.
2. Изучить и проанализировать особенности программирования игры «Жизнь» на языке Python.
3. Продумать и реализовать графику игры.

[1] Википедия – свободная энциклопедия:[Электронный ресурс] // Игра «Жизнь». URL: https://ru.wikipedia.org/wiki/Игра_«Жизнь» (Дата обращения: 10.12.2020).

4. Запрограммировать алгоритм игры на гексагональной решётке.

Объект исследования – программирование на языке Python.

Предмет исследования – игра «Жизнь» Конвея.

Методы исследования:

- Анализ (для изучения информации об особенностях языка Python и выявления основных характеристик программы);
- Сравнение (для выявления общих черт и различий между картами);
- Описание (для выявления и фиксирования информации об особенностях игры «Жизнь»);
- Эксперимент (для выявления ошибок в программе);
- Обобщение (для формулировки вывода на основе анализа).

Несмотря на то, что данная игра очень известна, она продолжает удивлять всё новыми и новыми возможностями, которые открывают конфигурации «Жизни». Таким образом, **степень изученности темы** достаточно высока, но эта характеристика больше относится к классической вариации «Жизни» на стандартном клеточном поле.

В нашей работе обозначено программирование математической игры «Жизнь» на нестандартном поле, состоящем из шестиугольников. Появление новых единиц и правил игры неизбежно повлияет на фигуры в ней, они будут отличаться от стандартных. **Новизна работы** заключается в реализации собственного алгоритма для нестандартной формулировки «Жизни» Конвея.

Личный вклад автора работы заключается в написании версии программы «Жизни» Конвея на гексагональной решётке на языке Python, которая по результатам анализа может оказаться проще для реализации, чем более распространённые варианты кода.

Практическая значимость исследования заключается в реализации игры «Жизнь», которая будет взаимодействовать с пользователем, то есть процесс самой «жизни» будет контролируемым.

Глава 1. Понятие и правила игры «Жизнь» Конвея

1.1. История создания игры

В 1940-х годах Джоном фон Нейманом была выдвинута идея создания машины, которая могла бы воспроизводить сама себя. Ему удалось создать математическую модель такой машины, однако она работала по чересчур сложным правилам. Удалось упростить этот автомат британскому математику Джону Хортону Конвею, который и известен как создатель игры «Жизнь». Впервые правила этой задачи-игры были опубликованы в 1970 году в выпуске журнала Scientific American.

Джон Конвей придумал не столько игру, сколько математическую задачу. В каждый момент времени поле «Жизни» изменяется по особым законам: первоначальная расстановка единиц игры и события, происходящие в квадрате 3 на 3 (клетка и её 8 соседи), определяют дальнейшее развитие клеточной «жизни». Тем не менее простые правила не помешали задаче стать столь популярной среди математиков, а даже наоборот подпитывали их надежды найти новую конфигурацию[2] в игре и попробовать смоделировать с её помощью процесс из естественного мира.

Мир вокруг нас так же строится по простым правилам, хотя нам кажется, что все эти процессы несопоставимы со вселенной «Жизни». Однако это не совсем так. Игра получила распространение не только среди математиков, но и перекочевала в некоторые другие дисциплины. Оказалось, что с помощью клеточного автомата можно моделировать огромное количество процессов. Например, в биологии «Жизнь» использовалась для моделирования процесса развития примитивных организмов, а в бактериологии применялась для воссоздания точной копии процесса размножения бактерий.

[2] Взаимное расположение предметов или их частей (Толковый словарь Ожегова)

1.2. Правила игры на клетчатом поле

Представьте себе листок бумаги в клетку. На этом листке произвольным образом закрашены некоторые из клеток. Назовём закрашенную клетку живой, а нетронутую – мёртвой. Теперь рассмотрим соседей каждой клеточки. Всего их 8. Далее следуем алгоритму:

1. Если клетка пустая, то есть является мёртвой, и при это ровно три её соседа живые, то в следующем поколении эта клетка родится. Соответственно, её мы закрасим.
2. Если клетка живая и имеет две или три закрашенные клетки, то в следующий момент времени она продолжить жить.
3. Если соседей у живой клетки меньше двух, то она умрёт «от одиночества».
4. Если у закрашенной клетки более трёх живых соседей, то она умирает «от перенаселения».

1.3. Правила игры на гексагональной решётке

Имеется n строк, образующих соты. Длина строк m > 1. Считается, что на каждую ячейку в гексагональной решётке влияют двенадцать соседей. Они влияют по-разному в зависимости от того, какую степень близости имеет сосед: первую или вторую (см. рис. 1).

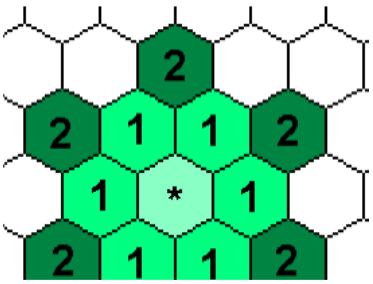


Рис.1. Ячейка и её соседи двух степеней близости

Введём величину IMPACT, обозначающую влияние соседей на ячейку. Она будет зависеть от количества соседей первого и второго порядка, а ещё от их «влияния», заданного определёнными константами. Расчёт влияние осуществляется по следующей формуле

IMPACT = FST_IMPACT * FST_COUNT + SND_IMACT * SND_COUNT, где

FST_IMPACT – влияние одного соседа 1-го порядка, FST_COUNT – количество соседей первого порядка, SND_IMPACT – влияние одного соседа 2-го порядка, SND_COUNT – количество соседей второго порядка.

Далее нам понадобятся обозначение констант, отвечающих за порог изменения состояния каждой ячейки. BIRTH_BEGIN и BIRTH_END задают границы, в которые должно войти значение IMPACT, что клетка родилась на следующем этапе игры. LIVE_BEGIN и LIVE_END отвечают за границы, в которые должно поместиться значение IMPACT, чтобы клетка продолжила жизнь на следующем этапе. Если IMPACT < LIVE_BEGIN или IMPACT > LIVE_END, то клетка умирает на следующем этапе жизни.

Константы задаются следующим образом:

- LIVE_BEGIN = 2.0
- LIVE_END = 3.3
- BIRTH_BEGIN = 2.3
- BIRTH_END = 2.9
- FST_ IMPACT = 1.0
- SND_ IMPACT = 0.3.

1.4. Виды игрового поля

Игровое поле «Жизни» в классическом варианте представляет собой прямоугольную матрицу. Однако в процессе игры живые клетки могут появиться на краю поля. А что с ними происходит дальше? Это зависит от того, какой тип карты реализован в программе. Существует несколько видов игрового поля:

1.3.1. Ограниченное поле

Клетки, которые могли бы родиться за пределами поля, если бы у него было продолжение, никуда не переносятся. Соседи для каждой клетки считаются, как обычно, например, у клетки угла карты будут проверены только 3 её соседа, других у неё нет.

1.3.2. Тороид[3]

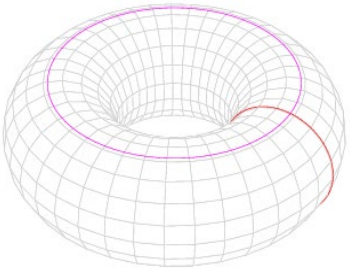


Рис.2. Поверхность тороида

Иногда такую карту называют «бублик». Такое поле обладает свойствами поверхности тороида (см. рис.2). Верхняя и нижняя, правая и левая границы попарно условно соединены. Поле будет всё так же иметь вид прямоугольной матрицы, однако все крайние клетки, которые будут рождаться за пределами поля будут переноситься начало противоположной стороны карты.

1.3.3. Цилиндр

Карта имеет свойства боковой грани цилиндра, т.е., в отличие от тороида, условно соединены только две противоположные грани: верхняя и нижняя или правая и левая. Таким образом перенос клеток будет осуществляться только в случае выхода за границу «связанной» грани.

1.4. Понятие форм «Жизни»

Действия законов реального мира привело к появлению многообразных существ и явлений. Так же и в клеточном автомате: через какое-то количество времени в игре наблюдается определённая закономерность. Клетки объединяются в формы, некоторые из которых становятся устойчивыми. Одни «клеточные существа» через время исчезают, некоторые становятся долгожителями, так как прежде чем стабилизироваться, они долгое время живут и изменяются.

Уже не одно десятилетие математики открывают новые и новые конфигурации, при этом каждое из них имеет своё название, которое в большинстве случаев взято из реального мира, жизни. Например, в игре

[3] Тороид (тор) — поверхность вращения, получаемая вращением образующей окружности вокруг оси, лежащей в плоскости этой окружности и не пересекающей её.

Тор(поверхность): [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/wiki/Тор_(поверхность)

существуют так называемый «планер». Это осциллятор, то есть конфигурация клеточного автомата, которая после конечного числа поколений повторяется в изначальном виде и положении. Однако помимо это она не стоит на месте и подобно самолёту перемещается в по карте игры по определённой траектории.

Существуют же и такие конфигурации, период изменения которых равен единице, то есть эта фигура не изменяется совсем. Однако это продолжается лишь до тех пор, пока на неё не повлияет другая соседняя конфигурация или даже отдельно взятая клетка, вошедшая в зону устойчивой фигуры.

Глава 2. Практическая часть

2.1. Описание реализованной программы

Графика была реализована на языке Python 3.9 при помощи библиотеки pygame. Полный код «Жизни» с комментариями указан в Приложении 1.

Сразу после запуска программы пользователь видит гексагональную решётку с равным количеством строк и столбцов. Живые клетки окрашиваются в зелёный цвет, мёртвые – в чёрный (см. Приложение 2).

В самом начале игры все клетки пустые. Пользователю предлагается самостоятельно сгенерировать первое поколение «Жизни», оживив некоторые клетки, нажимая по ним левой кнопкой мыши. Чтобы закончить создание первого поколения и начать развитие, нужно нажать на пробел или на правую кнопку мыши. После этого следующие поколения будут генерироваться с некоторой скоростью. Её можно изменять, прокручивая колёсико мыши вверх (для ускорения) или вниз (для замедления).

В любой момент можно остановить процесс генерации поколений, нажав на пробел или на правую кнопку мыши. Затем можно вновь изменять текущее поколение, оживляя и убивая клетки нажатием левой кнопки мыши.

2.2. Тестирование алгоритма

В ходе тестирования алгоритма мы пришли к выводу, что конфигурации ячеек гексагональной сетки «Жизни» более долговечные, чем клетки на прямоугольной карте. Можно сделать предположение, что это обосновано наличием соседей первого и второго порядка. На гексагональной решётке у каждой ячейки на 4 соседа больше, чем в классическом автомате Конвея.

Помимо этого, были выявлены одна устойчивая конфигурация (см. рис. 4) и один планер (см. рис. 3), подобные тем, что встречаются в классическом варианте игры. Стоит отметить, что найденная устойчивая фигура очень похожа на конфигурацию клеточной «Жизни», которой дали название «улей» (см. рис. 5), однако в нашем же варианте мы бы назвали эту фигуру «соты».

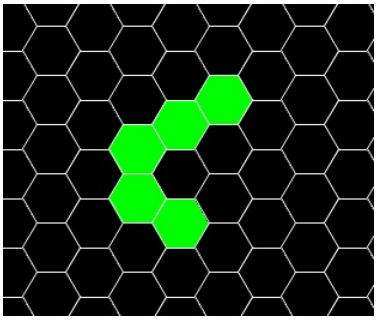


Рис. 3. Планер

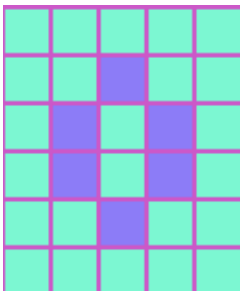


Рис. 5. «Улей»

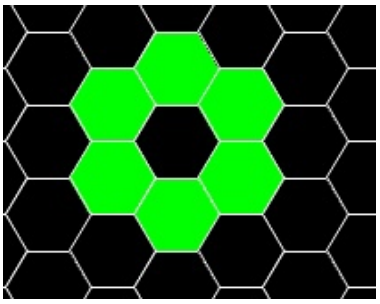


Рис. 4. «Соты»

Заключение

В ходе работы была написана математическая игра «Жизнь» на гексагональной решётке с 2D-графикой. Были выявлены и проанализированы особенности реализации программы на языке Python.

В ходе анализа алгоритма игры были выявлены некоторые отличия от классического варианта «Жизни» Конвея. Оказалось, что конфигурации шестиугольной «Жизни» более устойчивы, чем фигуры клеточного автомата. Были найдены и определены по своему типу две устойчивые конфигурации.

Так как игра на гексагональной решётке – малоизвестный вариант «Жизни» Конвея, перспективно и полезно для науки будет создание библиотеки конфигураций в игре, специфичной исключительно для преобразованных правил «Жизни».

Список использованной литературы и источников

1. Игра «Жизнь»: [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/wiki/Игра_«Жизнь» (Дата обращения: 10.12.2020).
2. Козлов Д.С. Задача Life – жизнь Конвея: канд. техн. наук. НГУ. Новосибирск, 2017. 9 с.
3. Словари и энциклопедии: [Электронный ресурс] // Толковый словарь Ожегова. URL: http://endic.ru/ozhegov/default.htm (Дата обращения: 30.03.2021).
4. Тор (поверхность): [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/wiki/Тор_(поверхность) (Дата обращения 06.04.2021).

Приложения

Приложение 1

```
import pygame
from copy import deepcopy
from math import sqrt, floor

di = [[-1, 1, 0, 0, 1, 1, 0, 0, -1, -1, 2, 2], [-1, 1, 0, 0, -1, -1, 0, 0, 1, 1, -2, -2]]
# Списки относительных координат соседей клетки, сначала для четных рядов, потом для нечётных
dj = [[0, 0, -1, 1, -1, 1, -2, 2, -1, 1, -1, 1], [0, 0, -1, 1, 1, -1, -2, 2, -1, 1, -1, 1]]
# Под одинаковыми индексами в двух массивах хранятся относительные координаты одной ячейки-соседа

FST_IMPACT = 1.0
SND_IMPACT = 0.3
BIRTH_BEGIN = 2.3
BIRTH_END = 2.9
LIVE_BEGIN = 2.0
LIVE_END = 3.3

def count_wh(side_len):
return side_len * 2, int(2 * sqrt(side_len **2** - (side_len / 2) 2))

def draw_hexagon(left, top, width, height, color, type):
x1, y1 = (left, top + height // 2)
x2, y2 = (left + width // 4, top)
x3, y3 = (left + 3 * width // 4, top)
x4, y4 = (left + width, top + height / 2)
x5, y5 = (left + 3 * width // 4, top + height)
x6, y6 = (left + width // 4, top + height)

if type:
pygame.draw.aalines(screen, color, True, [[x1, y1], [x2, y2], [x3, y3], [x4, y4], [x5, y5], [x6, y6]])
else:
pygame.draw.polygon(screen, color, [[x1, y1], [x2, y2], [x3, y3], [x4, y4], [x5, y5], [x6, y6]])

class Board: # Класс, объект которого является шестиугольной доской
def *init*(self, width, height): # Инициализация объекта
self.width = width
self.height = height
self.board = [[0] * width for _ in range(height + 1)]
self.left = 10
self.top = 10
self.side_len = 100
```

```
self.cell_width, self.cell_height = count_wh(self.side_len)

def set_view(self, left, top, side_len): # Изменение левой верхней координаты поля и длины стороны ячейки
self.left = left
self.top = top
self.side_len = side_len
self.cell_width, self.cell_height = count_wh(side_len)

def render(self): # Вывод поля на экран
N = self.width
M = self.height

left = self.left
top = self.top

for i in range(M):
for j in range(N):
if not self.board[i][j]:
draw_hexagon(left, top + (j % 2) * (-self.cell_height // 2), self.cell_width, self.cell_height,
(0, 0, 0), 0)
else:
draw_hexagon(left, top + (j % 2) * (-self.cell_height // 2), self.cell_width, self.cell_height,
(0, 255, 0), 0)
draw_hexagon(left, top + (j % 2) * (-self.cell_height // 2), self.cell_width, self.cell_height,
(255, 255, 255), 1)
left += self.cell_width * 3 // 4
top += self.cell_height
left = self.left

def get_click(self, mouse_pos): # При нажатии левой кнопки мыши
cell = self.get_cell(mouse_pos)
self.on_click(cell)

def get_cell(self, pos): # Определяет в какую клетку поля было произведено нажатие
x, y = pos
row, col = 0, 0
x, y = x - self.left, y - self.top
x_period = int(self.cell_width * 1.5)
y_period = self.cell_height

xrem = x % x_period
yrem = y % y_period

x_part = xrem // (x_period // 6)
y_part = yrem // (y_period // 4)

if x_part == 1 or x_part == 2:
```

```
row = x // (self.cell_width * 3 // 4)
col = y // self.cell_height

if x_part == 4 or x_part == 5:
if y_part < 2:
row = x // (self.cell_width * 3 // 4)
col = y // self.cell_height
elif y_part > 1:
row = x // (self.cell_width * 3 // 4)
col = y // self.cell_height + 1
        if x_part == 0:
if y_part == 0:
row = x // (self.cell_width * 3 // 4) - 1
col = y // self.cell_height
elif y_part == 1 or y_part == 2:
row = x // (self.cell_width * 3 // 4)
col = y // self.cell_height
else:
row = x // (self.cell_width * 3 // 4) - 1
col = y // self.cell_height + 1

if x_part == 3:
if y_part == 0:
row = x // (self.cell_width * 3 // 4)
col = y // self.cell_height
elif y_part == 1 or y_part == 2:
row = x // (self.cell_width * 3 // 4) - 1
col = y // self.cell_height
else:
row = x // (self.cell_width * 3 // 4)
col = y // self.cell_height + 1

return row, col

def on_click(self, cell): # Меняет состояние клетки, на которую нажали
x, y = cell
if (0 <= x <= self.width - 1) and (0 <= y <= self.height - 1):
if self.board[y][x]:
self.board[y][x] = 0
else:
self.board[y][x] = 1

class Life(Board): # Дочерний класс класса Board - шестиугольная доска для игры Жизнь
def init(self, width, height): # Инициализация
super().init(width, height)
self.cases = [self.board]
self.over = 0
```

```
def count_neighbours1(self, i, j): # Подсчёт соседей первого порядка
cnt = 0
for k in range(6):
cnt += self.board[(i + di[j % 2][k]) % self.height][(j + dj[j % 2][k]) % self.width]
return cnt

def count_neighbours2(self, i, j): # Подсчёт соседей второго порядка
cnt = 0
for k in range(6, 12):
cnt += self.board[(i + di[j % 2][k]) % self.height][(j + dj[j % 2][k]) % self.width]
return cnt

def next_move(self): # Генерация следующего поколения
new_field = [[0] * self.width for _ in range(self.height + 1)]
pp = 0
for i in range(self.height):
for j in range(self.width):
impact = self.count_neighbours1(i, j) * FST_IMPACT + self.count_neighbours2(i, j) *
SND_IMPACT
if self.board[i][j] == 0 and (BIRTH_BEGIN <= impact <= BIRTH_END) or self.board[i][j] == 1
and (LIVE_BEGIN <= impact <= LIVE_END):
new_field[i][j] = 1
else:
new_field[i][j] = 0
if pp == 0:
self.over = 1
self.board = deepcopy(new_field)
self.cases.append(self.board)

size = width, height = 590, 678
screen = pygame.display.set_mode(size)
pygame.display.set_caption('Игра "Жизнь" на шестиугольном поле')
board = Life(16, 16)
start = 0
board.set_view(0, 20, 24)
clock = pygame.time.Clock()
time = 50
slow = 10
fast = 50
running = True
while running: # Бесконечный цикл программы, обрабатывающий всеразличные нажатия
for event in pygame.event.get():
if event.type == pygame.QUIT:
running = False
elif event.type == pygame.MOUSEBUTTONDOWN:
if event.button == 1:
if not start:
board.get_click(event.pos)
```

```
elif event.button == 3:
if not start:
start = 1
time = slow
board.next_move()
else:
start = 0
time = slow
if event.button == 4:
if slow < 40:
slow += 1
time = slow
if event.button == 5:
if slow > 1:
slow -= 1
time = slow
elif event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
if not start:
start = 1
time = slow
board.next_move()
else:
start = 0
time = slow

screen.fill((0, 0, 0))
if start:
board.next_move()
board.render()
pygame.display.flip()
clock.tick(time)
pygame.quit()
```

Приложение 2

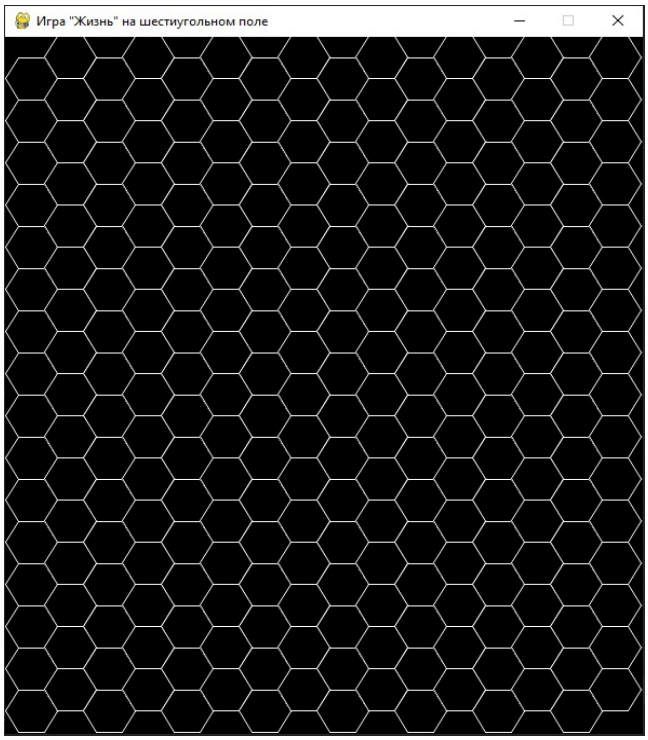


Рисунок 1. Карта игры при открытии приложения

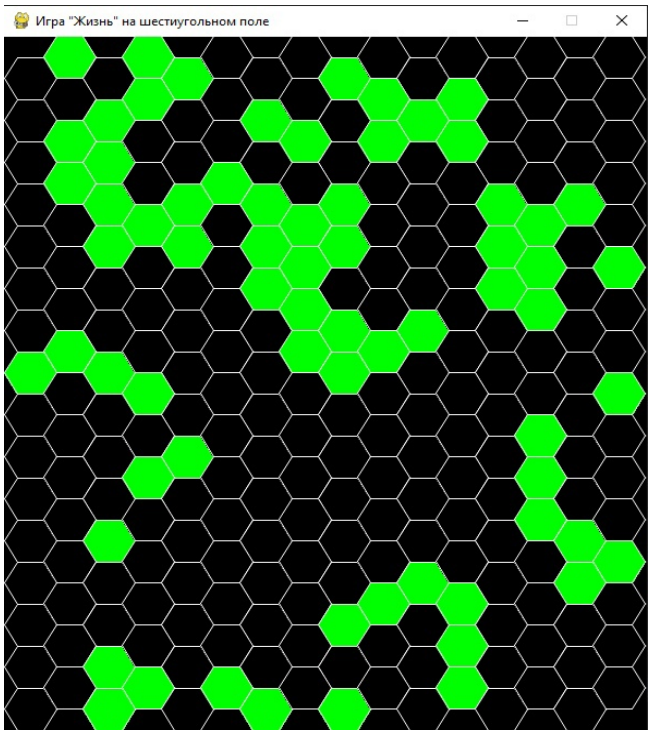


Рисунок 2. Процесс игры