Министерство науки и высшего образования РФ  
ФГАОУ ВПО

Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

**КУРСОВАЯ РАБОТА**

по дисциплине «Технологии программирования»

на тему «ИГРА ЗМЕЯ»

Исполнители:

|  |  |  |
| --- | --- | --- |
| Раздел | Фамилия И.О. | Группа |
| Формулирование требований | Занг Кхак Нгы | БИВТ 21-14 |
| Анализ предметной области | Илья Бирюков | ББИ-21-4 |
| Разработка алгоритмов | Ковальчук Анастасия Игоревна | ББИ-21-3 |
| Диаграммы переходов состояний | Занг Кхак Нгы | БИВТ 21-14 |
| Кодирование модулей |  |  |
| Структурное тестирование и отладка |  |  |
| Функциональное тестирование |  |  |

Проверил:   
ст. преп. каф. ИКТ

Карпишук А.В.

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

Москва, 2022

# **1.** **Формирование требований**

# **1.1. Описание**

Программа " ИГРА ЗМЕЯ " имитирует процесс роста змеи в дикой природе. Игрок будет управлять змеей в указанном направлении вверх, вниз, влево, вправо, чтобы собирать фрукты и избегать препятствия. Всего в игре пять видов фруктов: яблоки – способствуют росту змеи; груши – наделяют змею возможностью стрелять и уничтожать препятствия; киви – временное бессмертие; персик – ускорение; апельсин – укорачивает змею. В процессе сбора еды змея не должна кусать себя. Игра закончится, когда змея столкнется с препятствием или коснется себя. Счет будет сохранен в конце игры. Самые высокие баллы будут ранжированы (максимум 10 раз).

Очки будут складываться из количества съеденных яблок и персиков. 1 яблоко = 1 балл. 1 персик = 2 балла, если за время работы ускорения змея не врезалась.

# **1.2.** **Функции**

Программа должна реализовывать следующие возможности:

* Выбор цвета змеи;
* Генерация поля игры;
* Выкладка фруктов на поле и наделение змеи способностями;
* Контроль следования правилам;
* Вывод результатов;
* Pейтинг результатов;
* Bывод справки.

# **1.2.1. Выбор цвета змеи**

После открытия игры пользователь попадает на главную страницу, где ему предоставляется возможность выбора цвета змеи: зеленый, красный, черный.

# **1.2.2. Генерация поля игры**

Игровое поле имеет вид прямоугольника, по периметру которого хаотично располагаются препятствия. Если змея уничтожает препятствие, необходимо поставить новое в свободное место.

# **1.2.3. Выкладка фруктов на поле и наделение змеи способностями**

В процессе игры программе необходимо время от времени выкладывать на поле различные фрукты. В зависимости от того, какой фрукт съела змея, программа должна наделять ее определенными качествами.

# **1.2.4. Контроль следования правилам**

Необходимо следить за тем, чтобы змея не врезалась и не кусала себя.

# **1.2.5. Вывод результата**

Игра закончится, когда змея наткнется на препятствие или укусит себя. затем появится окно со словами «игра окончена» и «счет, который Вы получите:»

# **1.2.6. Рейтинг результатов**

После окончания игры, если пользователь набрал высокий балл, компьютер отобразит журнал результатов (может рассматриваться как игровой рекорд) с 10 самыми высокими результатами. Также в журнал пользователь может заглянуть самостоятельно, вызвав окошко нажатием кнопки.

# **1.2.7. Вывод справки**

По запросу пользователя программа должна выводить окно справки с yказанием правил.

# **1.3. Входные и выходные данные**

Входные и выходные данные зависят от каждого пункта раздела «Функции». Все данные сохраняются в текстовых файлах в специальной папке.

Для функции «Рейтинг результатов» входными данными будет пустая таблица с двумя столбцами: дата и количество очков. После каждой игры данные таблицы будут обновляться, они же и будут выходными.

Для функции «Вывод результатов» входных данных нет, зато на выходе игрок будет уведомлен о количестве полученных баллов за игру.

Все остальные функции выполняются автоматически.

* 1. **Пользовательский интерфейс**

Примерный внешний вид основных окон программы приведен ниже (Рисунок 1-Рисунок 3).

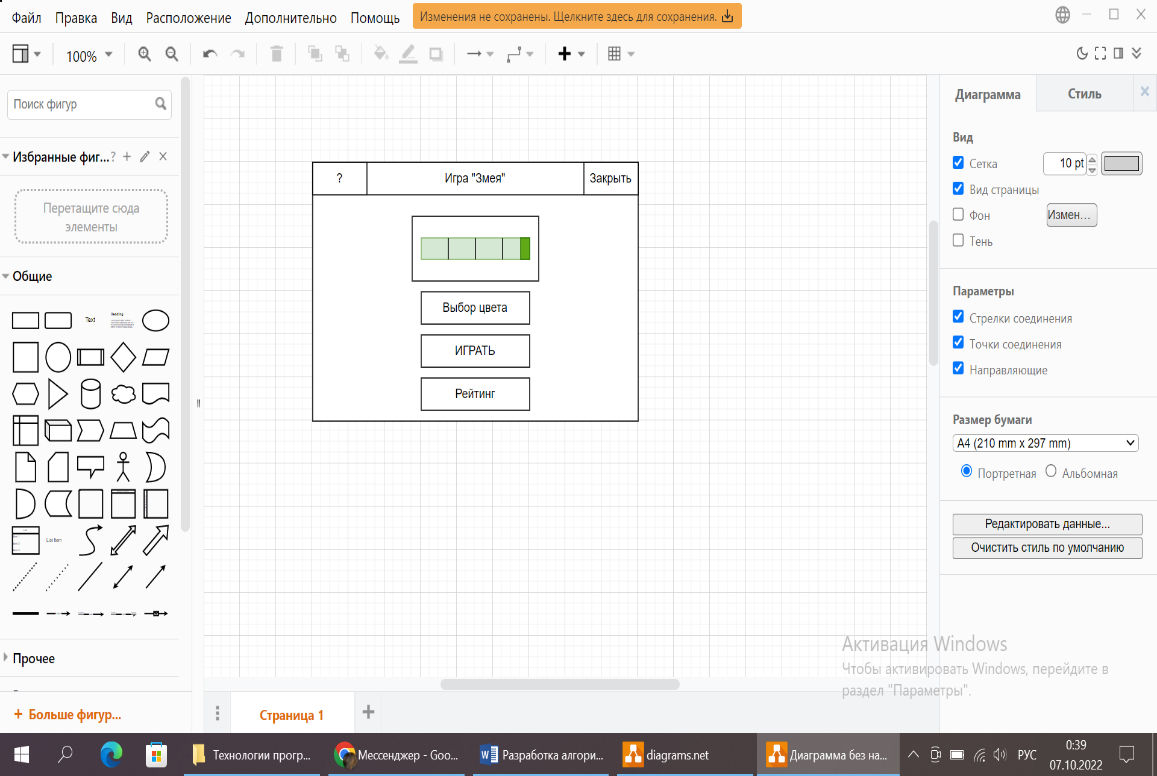


Рисунок 1 - Главная страница игры

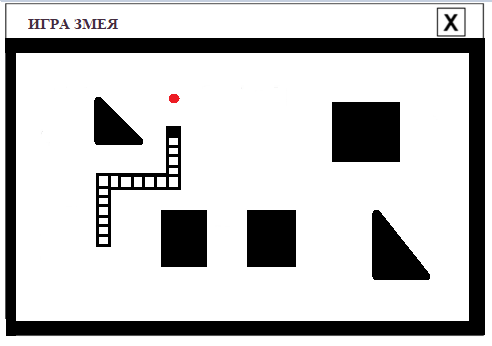


Рисунок 2 - Игровое поле

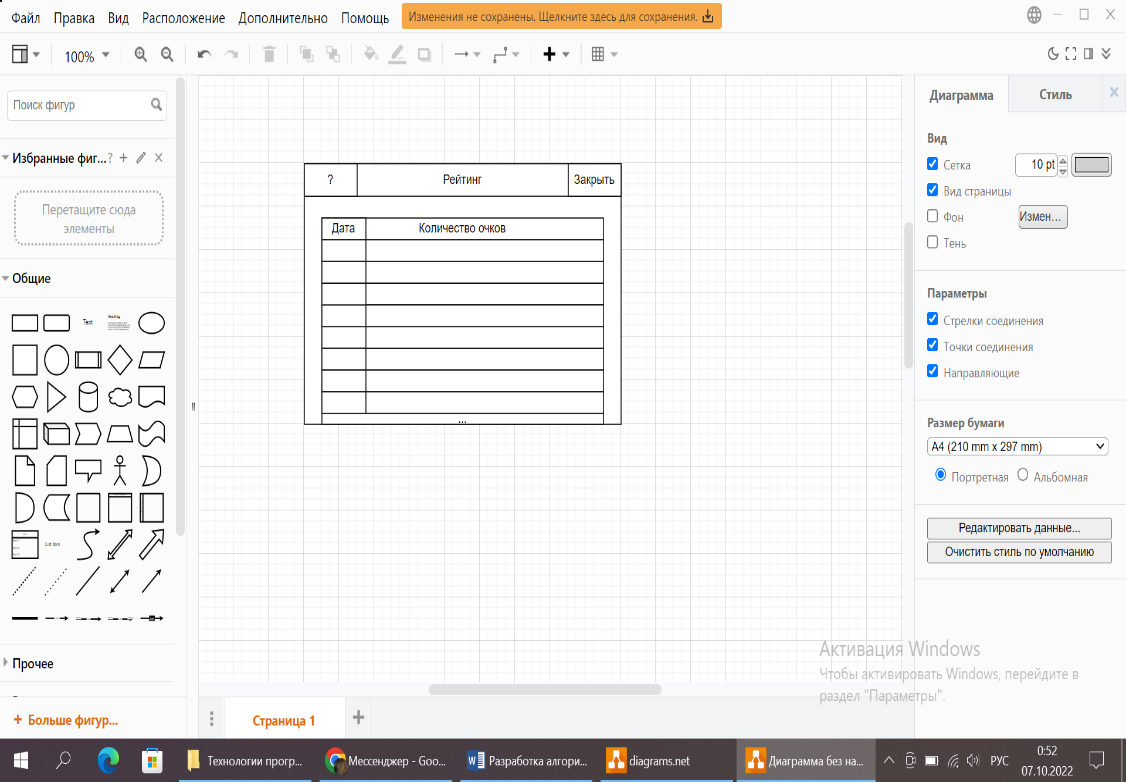


Рисунок 3 - Итоги игры

**2. Анализ предметной области**

Опираясь на описанные в первом разделе функциональные возможности будущей программы, можно выделить основные задачи, требующие предварительного исследования:

* Контроль процесса игры
* Анализ и исполнение программы

Разберем каждую задачу отдельно.

**2.1. Контроль процесса игры**

После попадания на игровое поле пользователю дается 3 секунды до старта [2] на изучение периметра с хаотично расположенными на нем препятствиями. В этот момент змейка находится в верхнем правом углу мордочкой вниз, ее длина – 4 клетки. Как-только игра начинается, змея может перейти в любое из состояний «Вверх», «Вниз», «Влево» и «Вправо» [1].

В процессе игры компьютеру необходимо периодически выкладывать различные фрукты на игровое поле [3] так, чтобы пользователь мог решить, нужны ли они ему (смотреть Рисунок 4). Например, змея начала движение, и через какое-то время недалеко между препятствиями появляется яблоко, которое она может забрать, пойдя к нему, а может избежать, свернув в другую сторону. Если фрукт остается нетронутым в течение 20 секунд, его стоит удалить с поля. Также более 10 фруктов на поле помещать нельзя.

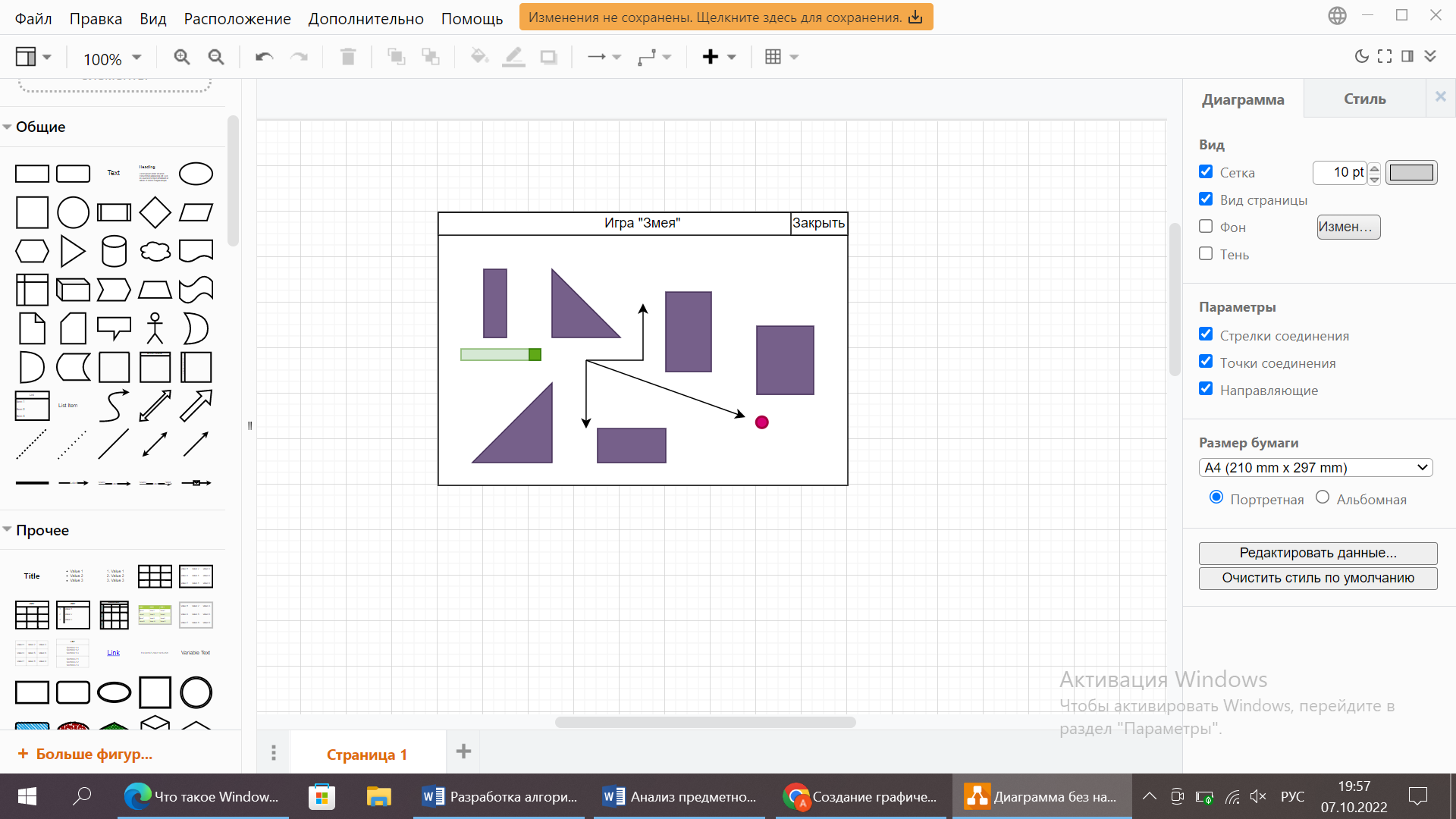


Рисунок 4 - Пример расположения яблока

В зависимости от фрукта, змею наделяют способностями или награждают. Таким образом: яблоко прибавляет 1 очко и удлиняет змею на одну клетку; груша позволяет уничтожать препятствие (после уничтожения препятствия следует расположить новое в любом свободном месте на расстоянии не менее 3 клеток от другого); киви наделяет бессмертием на 15 секунд, благодаря чему змея имеет право проходить «сквозь» препятствия; персик дает 2 очка, если за 20 секунд ускорения змея не врезалась и не укусила себя; апельсин укорачивает на 2 клетки [5].

Фрукты важно раздавать не в хаотическом порядке.

* Яблоко не зависит от других фруктов, поэтому его можно добавлять на поле каждые 8-12 секунд (на усмотрение разработчика). Но раз в минуту неплохо было бы устраивать змее «праздник», рассыпая n – яблок на поле в свободных местах (их количество зависит от того, сколько на поле фруктов уже имеется).
* Грушу лучше давать каждый раз, как змея вырастет на 15 клеток.
* Персик следует выкладывать, когда на поле появляются n яблок, чтобы за время ускорения змея успела собрать все фрукты.
* Как-только змея проглотит персик, программа должна выложить на поле киви. Таким образом, у змеи будет возможность собрать n яблок за короткое время и не врезаться в препятствия.
* Апельсин укорачивает змею на 2 клетки, не отнимая при этом очки. Данный фрукт может быть полезен, когда змея достигает больших размеров. Например, можно начинать периодически давать апельсин, когда длина змеи 25+.

Игра закончится, если змея врежется в препятствие [3] или укусит себя [4], после чего компьютеру будет необходимо объявить счет. Если змея перейдет одну из границ, следует переместить ее на противоположную сторону [4], обеспечив временное бессмертие.

**2.2. Анализ и исполнение программы**

В качестве среды разработки и написания скриптов были выбраны: среда разработки MS Visual Studio и язык программирования C# для платформы .NET Framework. Для анализа и выполнения программы могут использоваться различные методы [6] и интерфейсы [7] языка.

Простейшей технологией для создания графических приложений является Windows Forms [8]. Она очень проста в использовании и освоении. Другой популярной платформой является Windows Presentation Foundation [9], которая позволяет создавать настольные приложения с более насыщенной и богатой графикой [10].

**2.3. Список литературы**

1. Состояния змейки: <http://is.ifmo.ru/download/snake.pdf>

2. Таймер: <https://metanit.com/sharp/tutorial/11.9.php>

3. Выкладка фруктов и проверка столкновения: <https://upread.ru/art.php?id=746>

4. Перемещение: <http://www.opita.net/node/876>

5. Вспомогательный алгоритм реализации: <https://life-prog.ru/2_20819_algoritm-dlya-realizatsii-igri-zmeyka.html>

6. Методы: <https://learn.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/methods>

7. Интерфейсы:

<https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/types/interfaces>

8. Windows Forms: <https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/overview/?view=netdesktop-6.0>

9. Windows Presentation Foundation: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-6.0>

10. Графические редакторы C#: <https://metanit.com/sharp/forms.php>

# **3.** **Разработка алгоритмов**

Среди функций программы, перечисленных в пункте 1.2, а также сопутствующих задач, исследованных во втором разделе, алгоритмической проработки требуют следующие:

* Генерация поля игры
* Выкладка фруктов на поле и наделение змеи способностями

Функции «Выкладка фруктов на поле и наделение змеи способностями» и «Контроль следования правилам» напрямую связана с задачей «Контроль процесса игры», которую компьютер должен осуществлять автоматизировано, поэтому на данном этапе более предпочтительно, на мой взгляд, разобрать эти две функции отдельно, нежели разбирать одну большую задачу.

# **3.1. Генерация поля игры**

# **3.1.1.** **Варианты подходов**

# **3.1.1.1.** **Случайная генерация**

Этот подход заключается в том, что поле генерируется случайно, без каких-либо правил, после чего проверяется на «проходимость». Если поле является непроходимым, пересоздается (смотреть Рисунок 5).

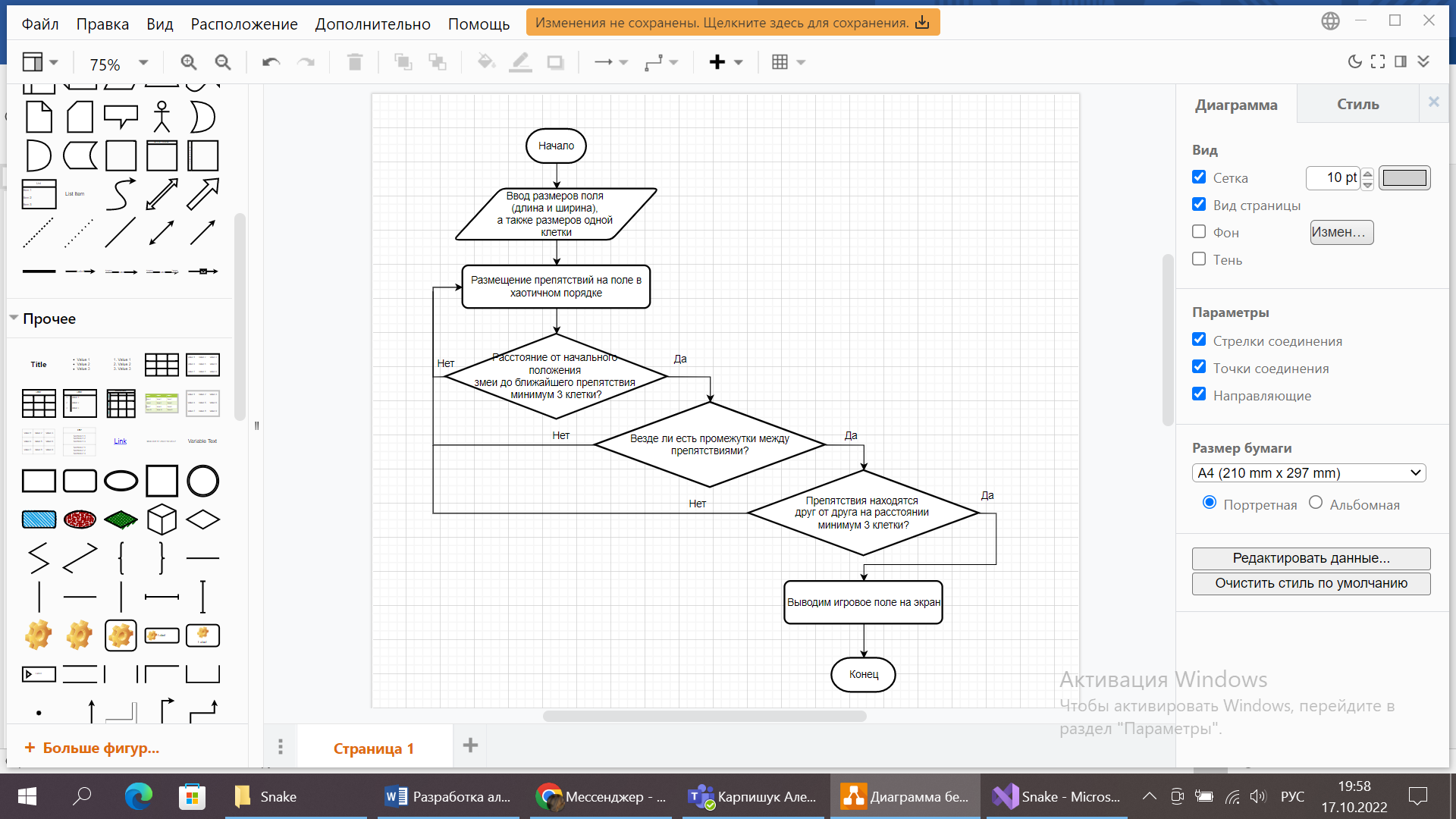


Рисунок 5 - Случайная генерация поля

# **3.1.1.2. Генерация с элементами случайности**

Поле генерируется случайно, но по определенным правилам. Благодаря правилам вероятность того, что поле окажется некорректным, невероятно мала. Тем не менее, частично случайную генерацию необходимо проверять (смотреть Рисунок 6).

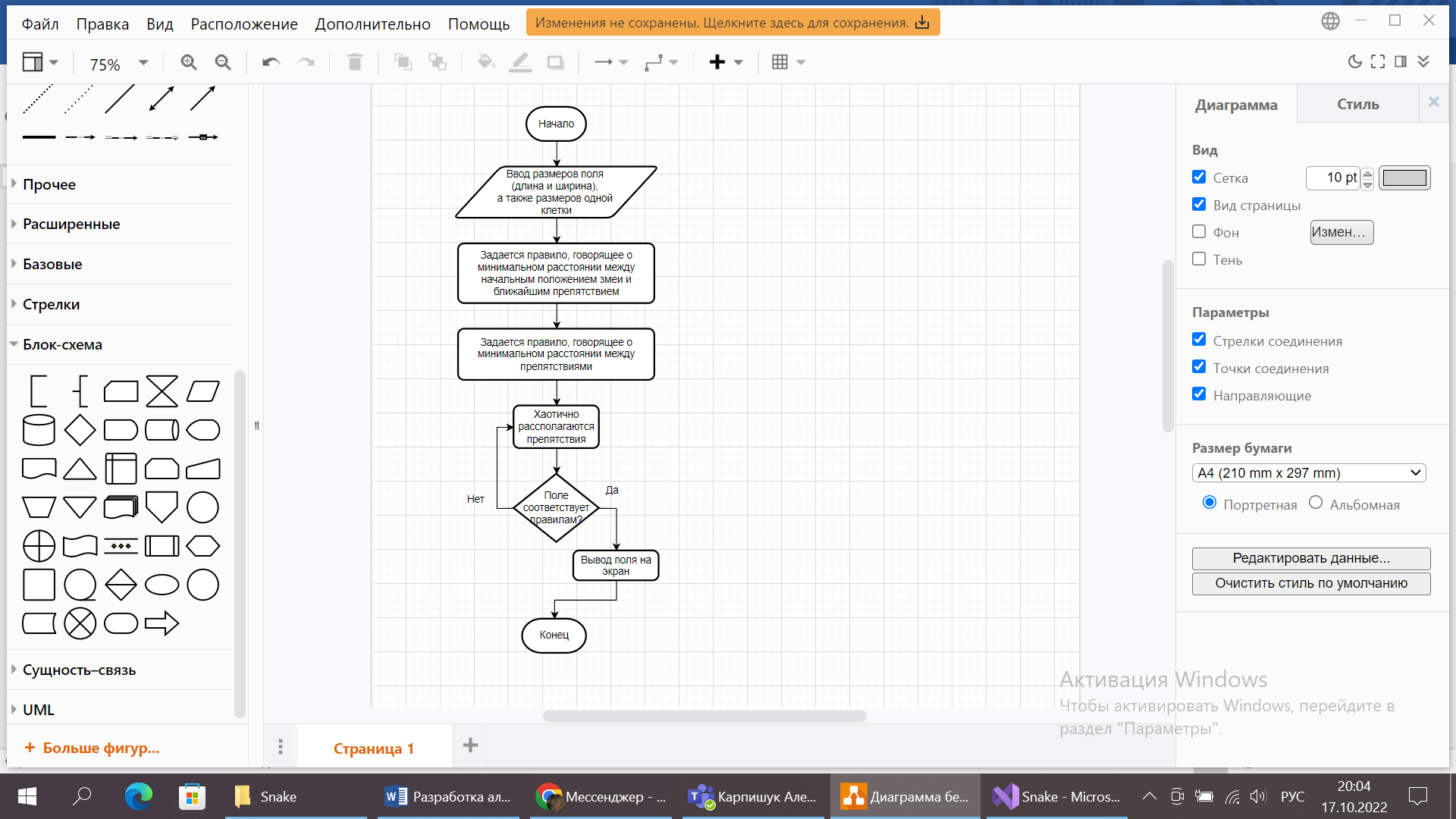


Рисунок 6 - Частичная случайная генерация

# **3.1.1.3. Шаблонная генерация поля**

Суть данного подхода в том, что у нас есть готовый шаблон, прописанный в коде (их может быть несколько), и при генерации программа его загружает (если шаблонов несколько, выбираем случайно). Какое количество шаблонов взять решает человек, который пишет код, так как прописывать их необходимо будет ему (смотреть Рисунок 7).

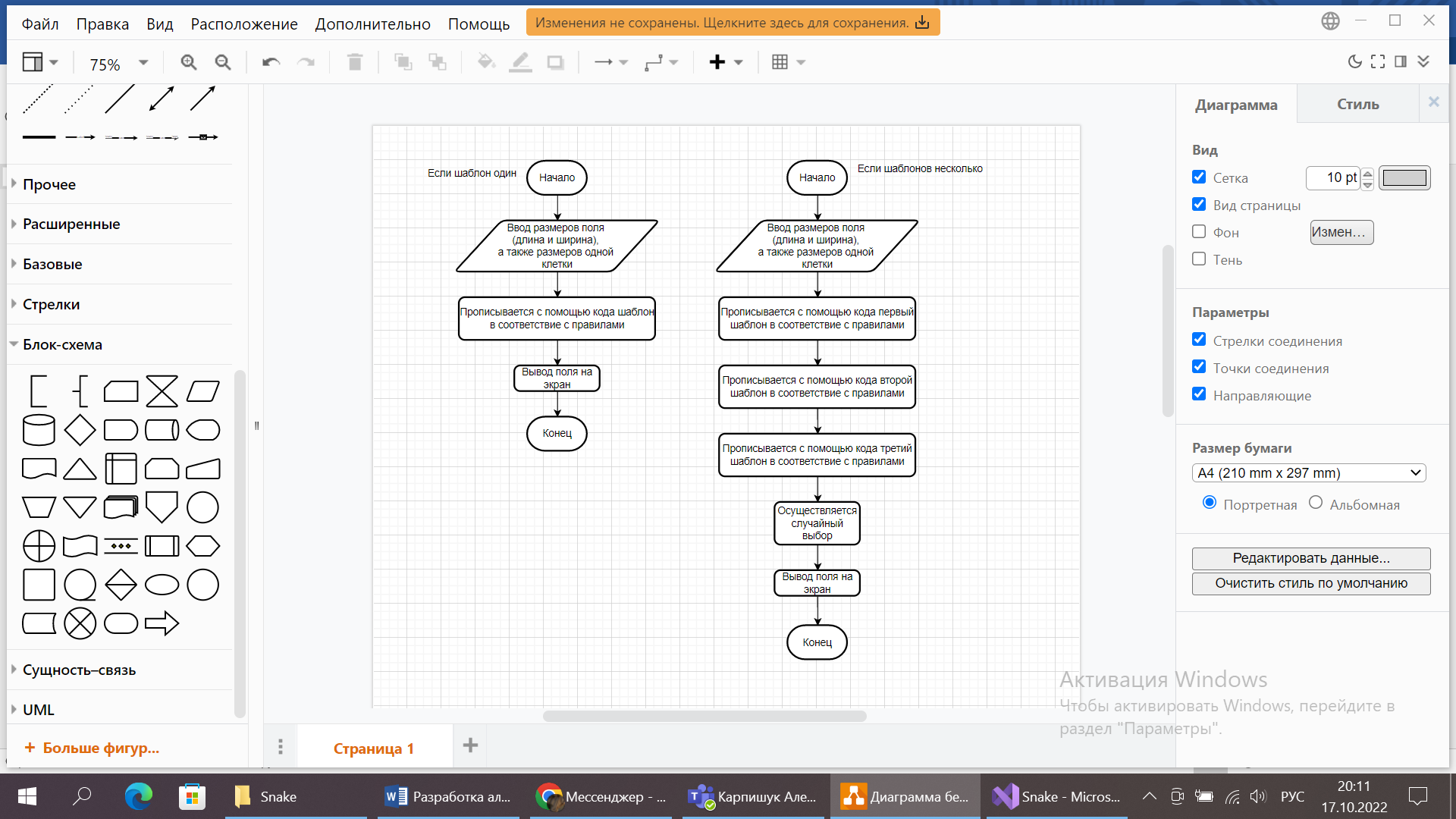


Рисунок 7 - Шаблонный метод

# **3.1.2. Анализ временной сложности алгоритмов**

Очевидно, что асимптотически время выполнения будет падать от алгоритма «Случайная генерация» до «Шаблонная генерация поля». Не смотря на рост пространственной сложности, предпочтительнее для реализации данной функции использовать третий алгоритм. Кроме того, последний описанный подход проще в реализации.

# **3.2. Выкладка фруктов на поле и наделение змеи способностями**

# **3.2.1. Варианты подходов**

# **3.2.1.1. Алгоритм «В лоб»**

Данный подход подразумевает совершенно не оптимизированный алгоритм, в котором присутствуют повторяющиеся элементы, усложняющие реализацию и увеличивающие асимптотику времени (смотреть Рисунок 8).

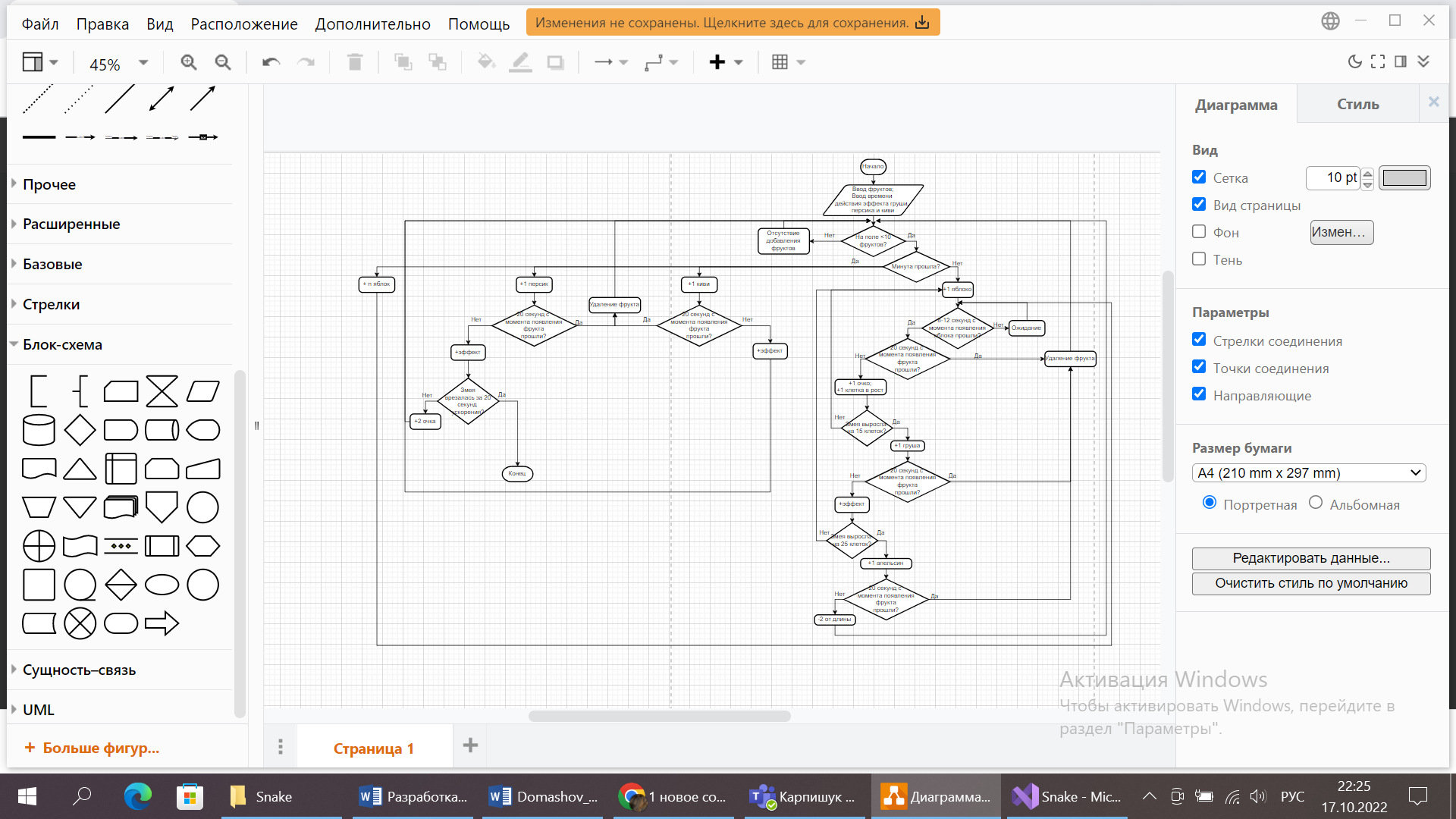


Рисунок 8 - Решение "В лоб"

# **3.2.1.2. Оптимизированный алгоритм**

Оптимизированный алгоритм подразумевает то, что при написании кода человек данные, которые касались времени, вынесет за циклы и обозначит их отдельной частью. Таким образом программа будет и выглядеть, и реализовываться гораздо проще (смотреть Рисунок 9).

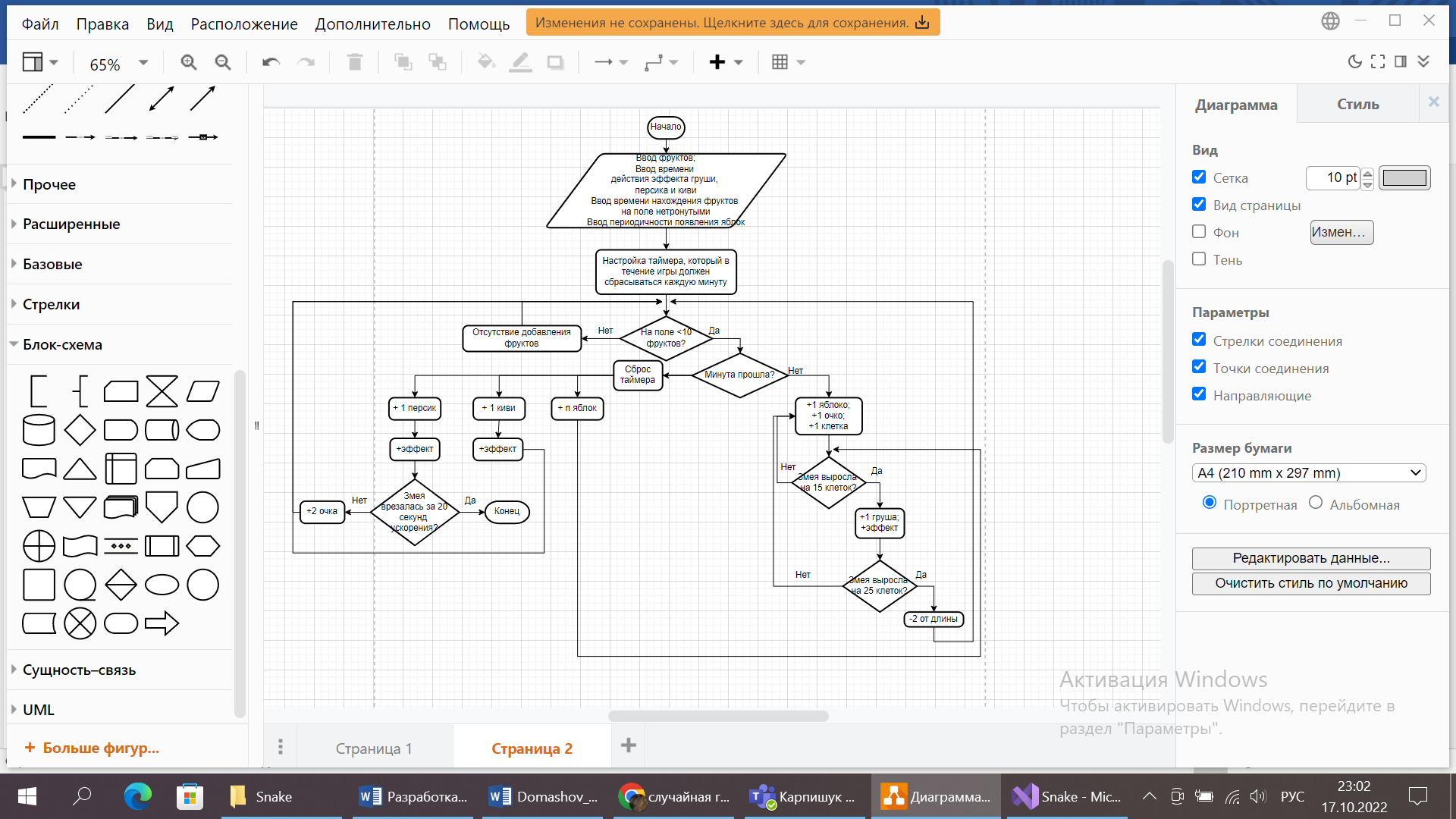


Рисунок 9 - Оптимизированный алгоритм

# **3.2.2. Анализ временной сложности алгоритмов**

Асимптотически время выполнения алгоритма «В лоб» будет гораздо больше, чем время оптимизированного варианта, поэтому для реализации функции «Выкладка фруктов на поле и наделение змеи способностями» предпочтительно использовать второй вариант. Кроме того, как было сказано ранее, второй алгоритм более простой в реализации.

**4.Диагарммы переходов состояния**

**4.1 Выделим сущности поведение которых можно описать диаграммой**

* Пользовательский интерфейса
* Поведение игрока-компьютера
* Поведение объектов

Чтобы оформить диаграмму переходов состояния я обратился к 1.1 «Описание поведение программы» и описал поведение игрока, интерфейса и объектов. Так как процесс хода цикличный я описал его в элементе составное состояние. Алгоритм действий компьютера был описан в пункте «Разработка алгоритмов», поэтому расписывать последовательность действий не обязательно.

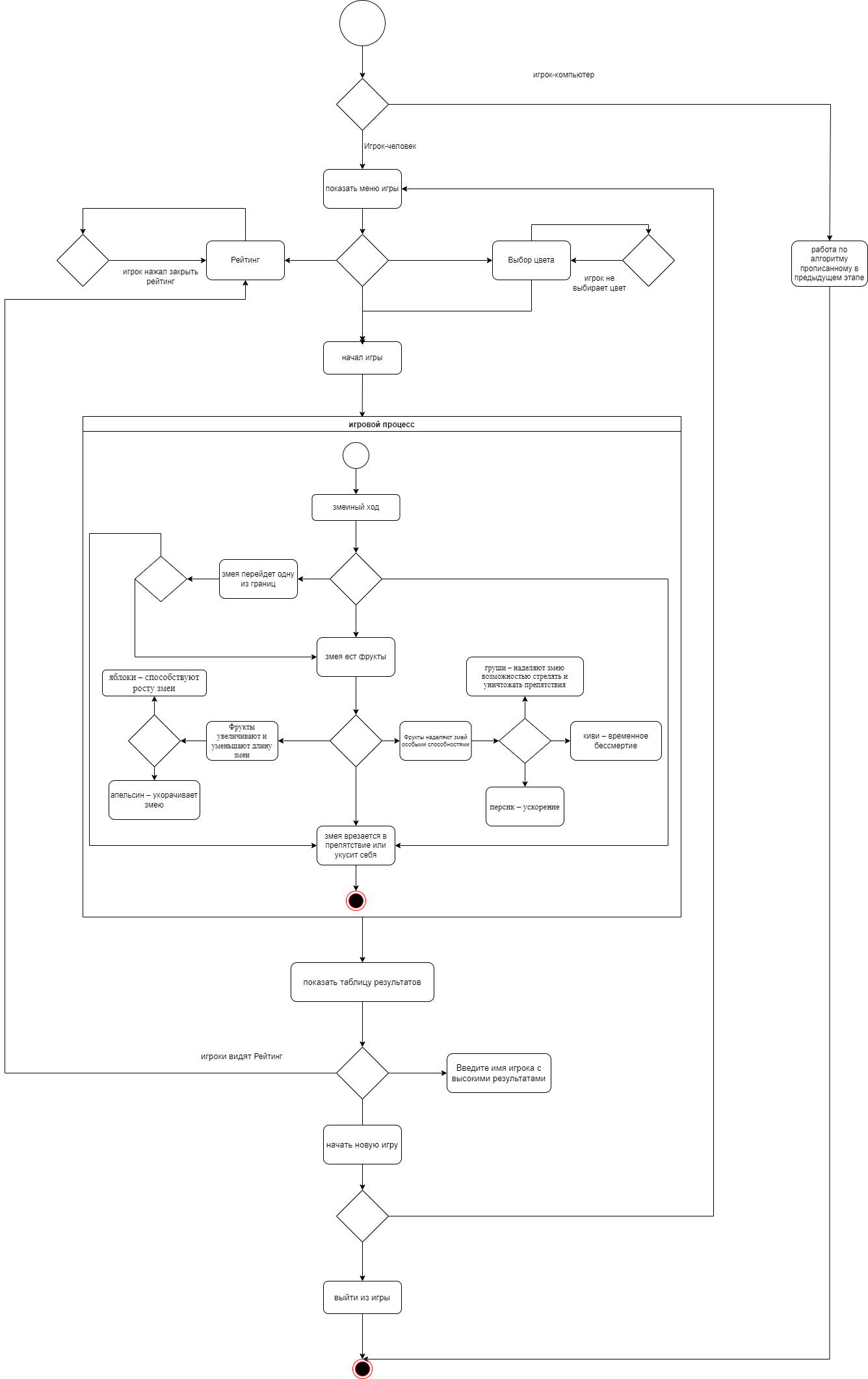


Рисунок 4.1.1-Диаграма переходов состояния программы «ИГРА ЗМЕЯ».