Таким образом, разработанное приложение представляет собой полноценную игровую реализацию Судоку с возможностью генерации новых задач и проверки решений. Дальнейшее развитие проекта может включать добавление уровней сложности с разным количеством заполненных ячеек, улучшение графического интерфейса с анимацией ввода, поддержку сохранения и загрузки игр, а также перенос на мобильные устройства. В результате получилось удобное и производительное приложение для игры в Судоку, в котором использованные методы генерации и проверки головоломок обеспечивают корректную работу программы, а графический интерфейс делает взаимодействие с пользователем интуитивным и удобным.

Список использованной литературы

1. Metanit [Electronic resource] // История создания Судоку – Mode of access: https://metanit.com/cpp/tutorial/1.1.php – Date of access: 26.03.2025.

2. Habr [Electronic resource] // Разработка логических игр – Mode of access: https://habr.com/ru/articles/454396/ – Date of access: 26.03.2025.

3. Geeksforgeeks [Electronic resource] // Алгоритмы решения Судоку – Mode of access: https://www.geeksforgeeks.org/sudoku-backtracking/ – Date of access: 26.03.2025.

ИСПОЛЬЗОВАНИЕ КЛЕТОЧНЫХ АВТОМАТОВ ДЛЯ СОЗДАНИЯ РАЗЛИЧНЫХ ФИЗИЧЕСКИХ МОДЕЛЕЙ НА ОСНОВЕ ИГРЫ «ЖИЗНЬ»

Пилипейко Александр (УО МГПУ им. И.П. Шамякина, г. Мозырь)
Научный руководитель – А.П. Сафронов, старший преподаватель

В данной работе будет показано конечное состояние «живых» клеток и будут разобраны некоторые нюансы, связанные с моделью, такие как правила игры и конечные состояния клеток.

Для начала нужно определить, что представляют собой клеточные автоматы, правила игры «Жизнь».

Клеточные автоматы представляют собой дискретные пространственно-временные системы, состоящие из ячеек, которые могут находиться в различных состояниях. Состояние каждой ячейки в следующий момент времени определяется ее собственным текущим состоянием и состояниями соседних ячеек в соответствии с заданными правилами перехода [1].

Правила игры «Жизнь» включают в себя условия жизни и смерти клеток в зависимости от количества соседей.

Игра моделирует жизнь простых организмов на двумерной решетке. Каждая ячейка решетки может находиться в одном из двух состояний: «живая» или «мертвая». Состояние ячейки в следующий момент времени определяется ее собственным текущим состоянием и состояниями восьми соседних ячеек в соответствии с простыми правилами: рождение – мертвая ячейка становится живой, если ровно три ее соседа живы; выживание – живая ячейка остается живой, если у нее ровно два или три живых соседа;

смерть – живая ячейка умирает от перенаселения (четыре или более живых соседей) или одиночества (менее двух живых соседей).

Конец игры наступает в случаях когда все клетки на поле оказываются пустыми; когда текущая конфигурация точно повторяет себя на более ранних шагах без изменений в расположении клеток; когда на следующем шаге ни одна клетка не меняет своего состояния.

Игра «Жизнь» не требует активного участия игрока – развитие поколений происходит автоматически на основе начального распределения живых и мёртвых клеток. По мере прохождения поколений, формируются различные структуры и фигуры, которые могут быть стабильными, колебательными или даже самореплицирующимися [2].

В начале игры на поле будут случайным образом расставлены живые и мертвые клетки. Затем каждое поколение будет обновляться в соответствии с правилами игры «Жизнь»: клетки будут рождаться, выживать или умирать в зависимости от количества соседей.

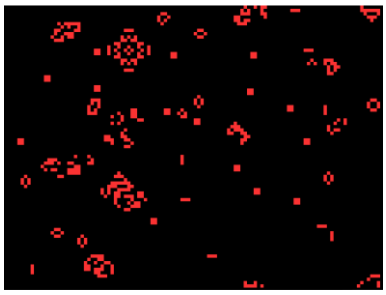

Рисунок 1 – Пример рабочего интерфейса модели

В ходе работы этой модели будут создаваться различные фигуры.

В игре «Жизнь» существует множество интересных конечных состояний, которые могут возникнуть в результате эволюции клеток. Одно из возможных завершений игры – стабильное состояние, когда все клетки остаются неподвижными и не изменяют своего положения. Другой вариант завершения игры – когда все клетки уничтожаются или формируют стабильные «конструкции», которые больше не меняются со временем.

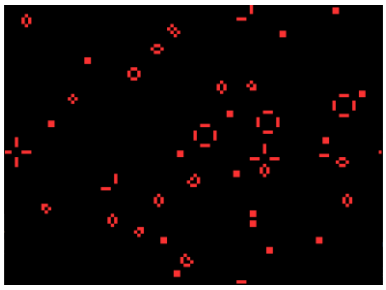

Рисунок 2 – Конечное состояние модели

В заключение, физические модели на основе клеточных автоматов представляют собой мощный инструмент для исследования сложных

физических систем и явлений. Клеточные автоматы позволяют моделировать эмерджентное поведение, когда сложное поведение системы возникает из простых правил взаимодействия между ее составляющими частями. Это открывает новые возможности для изучения самоорганизации, формирования структур, динамики процессов и других аспектов физических явлений. Их использование способствует лучшему пониманию сложных систем и помогает создавать новые методы анализа и прогнозирования поведения природных и искусственных объектов.

Список использованной литературы

1. Mathworld [Electronic resource] // Представление клеточных автоматов – Mode of access: https://mathworld.wolfram.com/ CellularAutomaton.html – Date of access: 12.05.2024.

2. Researchgate [Electronic resource] // Понятие о устойчивых структурах – Mode of access: https://www.researchgate.net/figure/Illustration-of-the-cellular-automata-model-Original-figure-was-retrieved-from_fig3_220136467 – Date of access: 12.05.2024.

ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ ВИРТУАЛЬНЫХ МЕТОДОВ И АБСТРАКТНЫХ КЛАССОВ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C++

Поливач Сергей (УО МГПУ им. И.П. Шамякина, г. Мозырь)
Научный руководитель – А.А. Голуб, канд. физ.-мат. наук, доцент

Объектно-ориентированное программирование в C++ характеризуется наличием ряда специфических особенностей, отличающих его от реализации этой парадигмы в таких языках программирования, как Java или C#. Например, используются виртуальные методы и абстрактные классы, которые являются одним из центральных механизмов, позволяющих реализовать гибкую работу с полиморфными объектами.

В C++ механизм виртуальных функций базируется на использовании специальной таблицы методов (vtable). Каждый объект класса, содержащего хотя бы один виртуальный метод, внутри себя хранит скрытый указатель на эту таблицу. Именно этот механизм обеспечивает динамический выбор реализации метода в момент выполнения программы [1].

Рассмотрим простейший пример реализации виртуального метода [2].

```
class Base {
public:
    virtual void show() {
      cout << "Base";
    }
};
class Derived : public Base {
public:
    void show() override {
      cout << "Derived";
   }
};
```