содержание

[Введение 3](#_Toc88900003)

[1 Исследовательский раздел 7](#_Toc88900004)

[2 Конструкторский раздел 10](#_Toc88900005)

[2.1 Математическая модель 10](#_Toc88900006)

[2.2 Разработка структуры компонента 11](#_Toc88900007)

[2.3 Разработка графического интерфейса компонента 14](#_Toc88900008)

[2.4 Описание входных и выходных данных компонента 15](#_Toc88900009)

[2.5 Разработка алгоритмов компонента 16](#_Toc88900010)

[3 Технологический раздел 31](#_Toc88900011)

[3.1 Отладка компонента 31](#_Toc88900012)

[3.2 Инструкция по установке компонента 32](#_Toc88900013)

[3.3 Инструкция по эксплуатации компонента 40](#_Toc88900014)

[4 Раздел охраны труда 41](#_Toc88900015)

[Заключение 46](#_Toc88900016)

[Список использованных источников 47](#_Toc88900017)

[Приложение А 48](#_Toc88900018)

Введение

Вычислительная техника является важнейшим компонентом процесса вычислений и обработки данных. Первыми приспособлениями для вычислений были, вероятно, всем известные счётные палочки, которые и сегодня используются в начальных классах многих школ для обучения счёту. Развиваясь, эти приспособления становились более сложными, например, такими как финикийские глиняные фигурки, также предназначаемые для наглядного представления количества считаемых предметов. Такими приспособлениями, пользовались торговцы и счетоводы того времени.

В современных компьютерах постоянно запускается и активно функционирует большое количество программного обеспечения с самым разным функционалом. Одни занимаются арифметическими расчетами, другие строят диаграммы, рисуют или помогают оставаться на линии связи с собеседниками через почту. Все они делятся на 3 основных вида:

* прикладное;
* системное;
* инструментальное.

Прикладное программное обеспечение – программы, предназначенные для выполнения определенных пользовательских задач и рассчитанные на непосредственное взаимодействие с пользователем. Смысл этой разновидности в выполнении четко поставленной задачи: рисовать, учитывать, открывать сетевые страницы, набирать текст. Если утилита нужна для конкретного выполнения действия, то она является прикладным программным обеспечением.

Системное программное обеспечение — это набор программ, которые управляют компонентами вычислительной системы, такими как процессор, коммуникационные и периферийные устройства, а также которые предназначены для обеспечения функционирования и работоспособности всей системы. В отличие от прикладного, системное программное обеспечение используется для обеспечения работы компьютера самого по себе и выполнения прикладных программ.

Инструментальное программное обеспечение представляют собой программы или целые пакеты программ, предназначенные для использования разработки новых программ. Современное инструментальное программное обеспечение – это целые интегрированные среды, или системы программных средств, используемых программистами для разработки программного обеспечения. Обычно среда разработки включает в себя текстовый редактор, компилятор и/или интерпретатор, средства автоматизации сборки и отладчик. Иногда также содержит систему управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Сюда входят компиляторы, отладчики, переводчики высокого уровня, редакторы, интерпретаторы и другие средства.

На первых компьютерах приходилось программировать двоичными машинными кодами. Однако программировать таким образом – достаточно трудоемкая и сложная задача. Для упрощения этой задачи стали появляться языки программирования низкого уровня, которые позволяли задавать машинные команды в более понятном для человека виде.

Язык программирования предназначен для написания компьютерных программ, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс, организовать управление различными объектами, и т. п. Язык программирования отличается от естественных языков тем, что предназначен для управления электронно-вычислительными машинами, в то время как естественные языки используются, прежде всего, для общения людей между собой. Большинство языков программирования использует специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Языки низкого уровня, как правило, используют для написания небольших системных программ, драйверов устройств, модулей стыков с нестандартным оборудованием, программирование специализированных микропроцессоров, когда важнейшими требованиями являются компактность, быстродействие и возможность прямого доступа к аппаратным ресурсам.

Примером языка низкого уровня является ассемблер. Ассемблер машинно-зависимый язык, т. е. он отражает особенности архитектуры конкретного типа компьютера. Исходная программа, написанная на ассемблере, состоит из одного или нескольких исходных модулей, а каждый модуль – из операторов. Написание программ на ассемблере является довольно сложной задачей, к тому же необходимы знания устройств компьютера. И тем не менее программы на ассемблере – самые эффективные и работоспособные.

Программист, создающий алгоритм для компьютера на языке низкого уровня, обращаются непосредственно к ресурсам компьютера: процессору, памяти, периферийным устройствам. Это обеспечивает высокую скорость работы программ, поскольку, в отличие от высокоуровневых, в низкоуровневых языках отсутствуют скрытые фрагменты кода, добавляемые автоматически компилятором во время преобразования исходного текста в бинарный код.

В ходе развития программирования возникла необходимость разработки новых, более совершенных языков программирования, которые бы были схожи с естественными языками и позволяли бы не работать напрямую с машинными командами. Их стали называть языками высокого уровня. Языки высокого уровня ориентированы на описание алгоритмов, поэтому их называют алгоритмическими языками. Преимуществом таких языков является большая наглядность и независимость от конкретного компьютера. Разрабатывать программы на таких языках гораздо проще и ошибок допускается меньше. Значительно сокращается время разработки программы, что особенно важно при работе над большими программными проектами.

При попытке описать вещи из реального мира возникают проблемы с поддержанием целостности программы, появляется многократное повторение одного и того же кода. Чтобы минимизировать проблему повторения кода, были разработаны объектно-ориентированные языки программирования (далее – ООЯ).

Объектно-ориентированное программирование (далее – ООП) основывается на представлении программы в виде множества объектов. Каждый объект относится к какому-либо классу, который, в свою очередь, занимает свое место в наследуемой иерархии. Использование ООП минимизирует избыточные данные, это улучшает управляемость, понимание программы.

Удобство использования ООП заключается в том, что оно дает возможность создавать расширяемые системы. Это одно из самых значительных достоинств ООП, и именно оно отличает данный подход от традиционных методов программирования. Расширяемость означает, что существующую систему можно заставить работать с новыми компонентами, причем без внесения в нее каких-либо изменений. Компоненты могут быть добавлены на этапе исполнения программы.

Объектно-ориентированное программирование обладает рядом принципов:

Инкапсуляция – это механизм объединения данных в единый компонент, дающий возможность защитить и спрятать их. То есть ограничить доступ одних компонентов программы к другим. Инкапсуляция позволяет над каждой частью программы работать изолированно.

Наследование - создание нового объекта/класса на основании старого.

Полиморфизм - возможность переопределять методы, перегружать их в зависимости от данных.

В рамках данного курсового проекта планируется разработать компонент, который сможет использоваться в качестве иллюстрации механизмов объектно-ориентированных языков, поэтому он будет полезен для начинающих программистов тем, что на его примере можно будет рассмотреть все вышеописанные принципы объектно-ориентированных языков программирования, а также возможность использования в своих программных продуктах методов стрельбы и отражения от поверхности.

Компонент игра в современном мире сможет найти применение в обучении детей, развитии логики и моторики.

Для того, чтобы взаимодействовать с компонентом, потребуется всего лишь добавить исходные файлы, скомпилировать проект, переместить компонент на форму.

1 Исследовательский раздел

Для того, чтобы разработать компонент, на рассмотрение были взяты две среды программирования:

* Microsoft Visual Studio 2019;
* Borland Delphi 2007;

Рассмотрим первую среду программирования – Microsoft Visual Studio 2019. Microsoft Visual Studio – это интегрированная среда разработки (IDE) от Microsoft, используемая для разработки компьютерных программ для Microsoft Windows, а также веб-сайтов, веб-приложений и веб-сервисов. Visual Studio использует платформы разработки программного обеспечения Microsoft, такие как Windows API, Windows Forms, Windows Presentation Foundation, Windows Store и Microsoft Silverlight.

К основным преимуществам Visual Studio относится поддержка множества языков при разработке. Visual Studio позволяет писать код на своем языке или любых других предпочитаемых языках, используя все время один и тот же интерфейс. Более того, Visual Studio также еще позволяет создавать Web-страницы на разных языках, но помещать их все в одно и то же Web-приложение. Единственным ограничением является то, что в каждой Web-странице можно использовать только какой-то один язык.

Помимо этого, данная среда разработки поддерживает интуитивный стиль кодирования. По умолчанию Visual Studio форматирует код по мере его ввода, автоматически вставляя необходимые отступы и применяя цветовое кодирование для выделения элементов типа комментариев. Такие незначительные отличия делают код более удобным для чтения и менее подверженным ошибкам. Автоматические параметры форматирования можно даже настраивать, что очень удобно в случаях, когда разработчик предпочитает другой стиль размещения скобок.

Не стоит забывать и о более высокой скорости разработки. Многие из функциональных возможностей Visual Studio направлены на то, чтобы помогать разработчику делать свою работу как можно быстрее. Удобные функции, вроде функции IntelliSense (которая умеет перехватывать ошибки и предлагать правильные варианты), функции поиска и замены (которая позволяет отыскивать ключевые слова как в одном файле, так и во всем проекте) и функции автоматического добавления и удаления комментариев (которая может временно скрывать блоки кода), позволяют разработчику работать быстро и эффективно.

Кроме того, у Visual Studio есть и свои недостатки. Во-первых, Visual Studio массивная, то есть запуск Visual Studio на слабом устройстве может повлечь за собой зависания компьютера. Во-вторых, это – цена. Средние и малые компании не всегда могут позволить себе корпоративную подписку, что уже говорить о независимых разработчиках, они предпочтут бесплатные аналоги. Конечно, качество стоит своих денег, но старт от 500$ отпугивает новичков. В-третьих, данная среда разработки является относительно молодой, вследствие чего в ней могут содержаться ошибки, возникающие при написании кода, которые можно и не найти в интернете.

Вторая среда на рассмотрении – Borland Delphi 2007. Delphi 7 вышла в 2002 году и по праву считается одной из самых удачных версий Delphi за всю историю языка. Очень и очень многие программисты, если не работали, то хотя бы учились на этой версии. Достаточно большой контингент разработчиков до сих пор отдаёт ей предпочтение, как проверенной, надёжной и не требовательной к ресурсам. В сравнении с Visual Studio в Delphi 7 используется более простой язык программирования – Object Pascal. Одним из основных преимуществ Delphi перед его аналогами заключается в достаточно быстром процессе разработки программных продуктов, которые, в свою очередь, включают в себя довольно замысловатый интерфейс. Также можно отметить наличие обширного набора компонентов, обеспечивающих работу с базами данных. Упорядоченный набор данных работает с разными базами данных, как с локальными, так и с промышленными.

Однако несмотря на все достоинства Delphi 7, её использование на сегодняшний день является, как минимум, не актуальным. Во-первых, со времени выхода Delphi 7 прошло много лет, информационные технологии претерпели кардинальные изменения. Что-то утратило актуальность и даже ушло в историю. Что-то, наоборот, появилось и стало доминирующим направлением (или одним из таковых). 64-разрядная архитектура, сенсорные экраны, мобильные платформы, NoSQL и многое другое, всё это в Delphi 7 либо не доступно вообще, либо требует больших затрат времени и труда для реализации. Во-вторых, функционал среды разработки не отвечает современным требованиям. Для своего времени функционал среды разработки в Delphi 7, был одним из самых мощных, но на сегодня, только подсветки синтаксиса и базовых функций редактирования кода уже недостаточно для эффективной работы. Ведь при выполнении тех же функций рефакторинга вручную, программист затрачивает гораздо больше времени и сил, чем, когда этот процесс автоматизирован.

Исходя из всех выявленных выше преимуществ и недостатков обеих сред разработки, была выбрана Microsoft Visual Studio 2019.

Для разработки компонента был выбран родительский класс Control, так как данный класс содержит в себе все необходимые для реализации компонента методы взаимодействия с манипулятором “мышь”, а также у него имеются инструменты для рисования изображения, которые смогут быть унаследованы создаваемым классом.

Для управления компонентом будет использоваться манипулятор “мышь”, при взаимодействии с которым будут реализованы методы управления компонентом.

Главной целью проекта является разработка визуального компонента игры “Shoot Balls”, который можно будет встраивать в прикладные приложения. Основная идея компонента заключается в создании готовой для работы игры.

2 Конструкторский раздел

2.1 Математическая модель

В качестве основных процессов были выбраны методы поиска шариков одного цвета и поиска островов, состоящих из шариков одного цвета.

В начале метода поиска одинаковых по цвету шариков необходимо найти все шарики вокруг прилетевшего. Сложность заключается в том, что добавление шариков в список зависит от того, в строку с каким номером – четным и нечетным прилетел шарик. В зависимости от строки в список будут добавляться шарики из предыдущей и следующей строк со следующими координатами:

* (x-1; y-1), (x-1; y), (x+1; y-1), (x+1; y) – для прилетевшего шарика в четную строку;
* (x-1; y), (x-1; y+1), (x+1; y), (x+1; y+1) – для прилетевшего шарика в нечетную строку.

После завершения поиска всех шариков вокруг прилетевшего, перебором полученного списка необходимо найти все шарики вокруг шариков уже из списка, исключая повторы. Работа данного алгоритма продолжается, пока не будут найдены все шарики одного цвета с прилетевшим. Затем, если количество элементов итогового списка больше одного, то необходимо удалить все шарики из этого списка. После чего будет вызван метод поиска островов.

Алгоритм метода поиска островов состоит из двух частей. Сначала для каждого шарика на игровом поле будет искаться путь до первой строки. Если путь не существует, значит найденная совокупность шариков является островом, и в последствии он должен быть удалён. Но, если путь будет найден, то это значит, что шарик не принадлежит острову. Далее произойдет сброс параметров поискового алгоритма и переход к следующему проверяемому шарику.

В результате работы этих двух алгоритмов после прилета шарика произойдет удаление всех одноцветных шариков, а также удаление всех образовавшихся островов.

2.2 Разработка структуры компонента

Черная сфера – система, внутреннее устройство которой неизвестно. То есть имеется общее представление о том, что делает эта система, какую информацию на вход нужно подать, и то, что система должна вывести. При этом механизм работы данной системы неизвестен. Примером подобной системы служат любые технические устройства для обычного человека. Ему известно, для чего они предназначены, но как они выполняют свою функцию – неизвестно.

На рисунке 2.1 изображено общее представление черной сферы.

Выходные параметры

Входные параметры

Рисунок 2.1 – Общее представление модели «Черная сфера».

Входными параметрами разрабатываемого компонента являются:

* цвет фона – данный параметр отвечает за то, какого цвета будет фон игры;
* цвета шариков – данный параметр отвечает за цвета размещаемых в компоненте шариков;
* размер шарика;
* количество строк и столбцов – данный набор параметров отвечает за размеры игрового поля;
* управляющее воздействие пользователя – набор методов взаимодействия пользователя с компонентом.

При наличии всех входных параметров на выходе будет получено изображение – игровое поле, заполненное шариками.

На рисунке 2.2 изображено представление разрабатываемого компонента в виде модели “Черная сфера”.

Управляющее воздействие пользователя

Цвет фона

Цвета шариков

Размер шарика

Количество строк и столбцов

Изображение

Рисунок 2.2 – Представление разрабатываемого компонента   
в виде модели “Черная сфера”.

Для того, чтобы выявить внутренние механизмы и структуры данных, требующиеся для хранения внутренней информации, применим метод декомпозиции.

Декомпозицией называют структурное разделение одной сложной цели на составляющие и формирование из них иерархического дерева. Иначе говоря, это процесс разделения одной крупной задачи на более мелкие, которые в свою очередь разбираются на простейшие действия.

На рисунке 2.3 изображен результат проведения первого этапа декомпозиции.

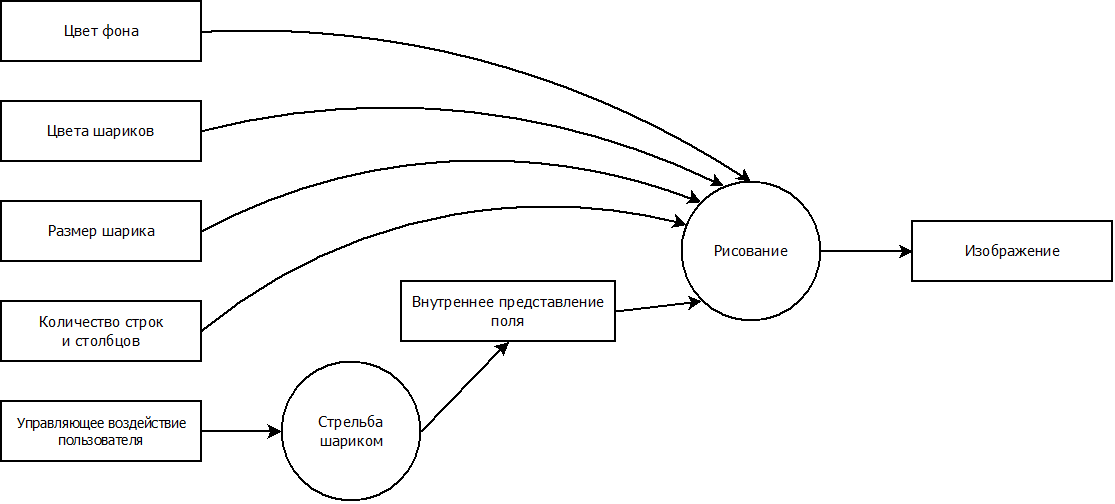


Рисунок 2.3 – Результат проведения первого этапа декомпозиции.

На представленной схеме прямоугольниками изображены внутренние переменные, кругами изображены методы, реализуемые в виде процедур и функций, стрелками поясняется логика и последовательность взаимодействия свойств и методов. Слева на схеме находятся входные параметры, справа отражены результаты работы класса в виде выходных параметров.

Для более детального представления представленной схемы, необходимо провести второй этап декомпозиции.

Окончательная структурная схема класса после проведения декомпозиции изображена на рисунке 2.4 и плакате КП.0902.07.000000.01 ПЛ.

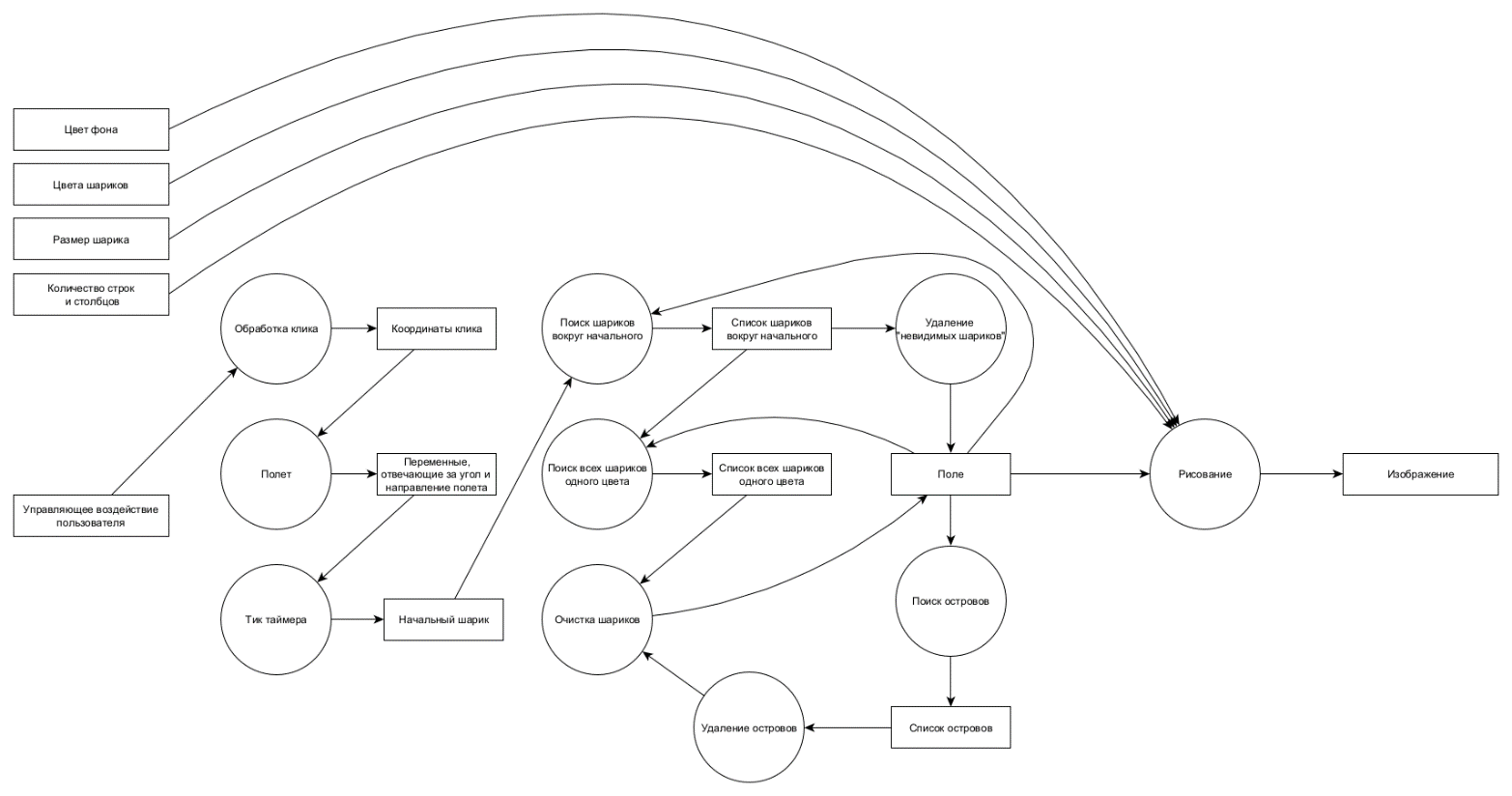


Рисунок 2.4 – Структурная схема класса ShootBalls.

В полученной окончательной структурной схеме были выявлены все нужные для реализации компонента методы, свойства и структуры для хранения данных.

2.3 Разработка графического интерфейса компонента

Реализуемый компонент будет соответствовать графической составляющей операционной системы Windows, которая была создана с идеей реализации программ в виде окон, имеющих прямоугольное представление.

Все окна, реализованные в операционных средах Windows, имеют единые инструменты взаимодействия пользователя с ними. Например, кнопки – свернуть развернуть и закрыть. Данные кнопки были впервые реализованы в Windows 95. С того времени они непрерывно сопровождали операционные среды меняясь только в дизайне. Интерфейс сред интуитивно понятен поскольку все элементы управления расположены по углам приложений.

Поскольку компонент разрабатывается под операционные среды Windows, на форме компонент примет прямоугольное представление.

При загрузке компонента на экране можно будет увидеть игровое поле, заполненное шариками разных цветов, а также начальный шарик, который в последствии нажатия левой кнопки мыши в определенную точку будет “выпущен” в указанном направлении. Количество цветов будет определять прикладной программист. Также компонент будет адаптируемым по размерам. Размеры будут также задаваться прикладным программистом и будут зависеть от размера шарика и их количества на игровом поле.

Игра заканчивается, когда пользователь сможет добиться устранения всех шариков на игровом поле.

2.4 Описание входных и выходных данных компонента

В подразделе 2.2 был выявлен ряд свойств. Также есть свойства, которые наследуются от родительского класса “Control” – BackColor, FieldSize.

Все требующиеся для управления работой класса свойства описаны в таблице 2.1

Таблица 2.1 – Входные и выходные данные компонента.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя свойства или события | Тип данных | Тип доступа | Метод доступа на чтение | Метод доступа на запись |
| BackColor (цвет заднего фона) | Color | чт./зп. | Унаследован от родительского класса | Унаследован от родительского класса |
| ColorsArr (цвета шариков) | List<Color> | чт./зп. | Стандартный механизм чтения | Модифицированная процедура set, которая выполняет перерисовку игрового поля при добавлении изменений цветов |
| BallWidth (размер шарика) | Integer | чт./зп. | Стандартный механизм чтения | Модифицированная процедура set с целью контроля за диапазоном вводимых значений и обеспечения перерисовки изображения на экране после смены значения свойства |
| Height (Высота компонента) | Integer | чт./зп. | Унаследован от родительского класса | Унаследован от родительского класса |
| Width (Ширина компонента) | Integer | чт./зп. | Унаследован от родительского класса | Унаследован от родительского класса |

2.5 Разработка алгоритмов компонента

В подразделе 2.2 была получена структурная схема класса. На этой схеме были выявлены все методы, необходимые для реализации бизнес логики компонента. Данные методы будут прорабатываться в этом разделе.

Метод StartGame служит для начальной настройки поля. Входных параметров не имеет. Блок-схема алгоритма метода приведена на рисунке 2.5.



Рисунок 2.5 – Блок-схема метода StartGame.

Метод EndGame служит для определения конца игры. Входных параметров не имеет. Блок-схема алгоритма метода приведена на рисунке 2.6.

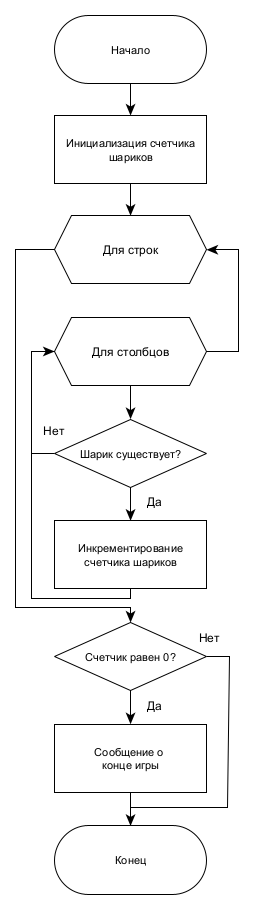


Рисунок 2.6 – Блок-схема метода EndGame.

Метод OnMouseDown служит для вызова метода полета шарика в случае, если он не запущен. Входным параметром является: MouseEventArgs e. Блок-схема алгоритма метода приведена на рисунке 2.7.

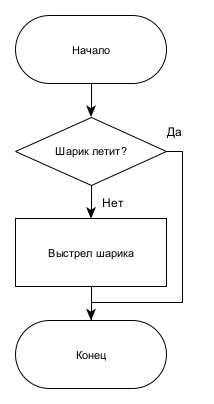


Рисунок 2.7 – Блок-схема метода OnMouseDown.

Метод Flight служит для определения координат нажатия левой кнопки мыши по компоненту, настройки переменных, отвечающих за направления полета шарика, а также запуск таймера. Входным параметром является: Point CurrentCircleCords, хранящий в себе координаты нажатия левой кнопки мыши по компоненту. Блок-схема алгоритма метода приведена на рисунке 2.8.



Рисунок 2.8 – Блок-схема метода Flight.

Метод DeleteNullBalls служит для удаления из списка несуществующих шариков. Входным параметром является: List<Ball> List, хранящий в себе список шариков вокруг прилетевшего. Выходным параметром является: List<Ball> ListWithoutNullBalls, хранящий в себе список шариков за исключением несуществующих. Блок-схема алгоритма метода приведена на рисунке 2.9.

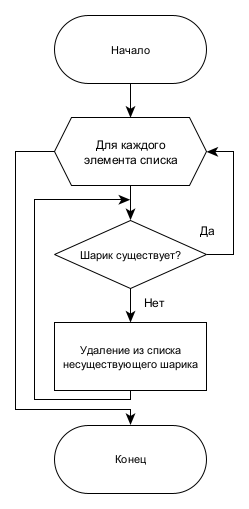


Рисунок 2.9 – Блок-схема метода DeleteNullBalls.

Метод FoundBalls служит для поиска шариков одного цвета с прилетевшим. Входными параметрами являются: int startRow, int startCol, содержащие в себе координаты прилетевшего шарика. Выходным параметром является List<Ball> BallsList, хранящий в себе список шариков одного цвета с прилетевшим. Блок-схема алгоритма метода приведена на рисунке 2.10.

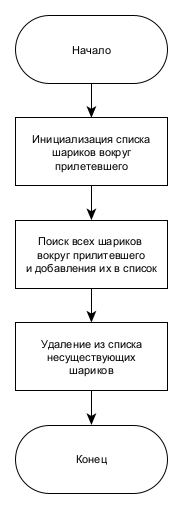


Рисунок 2.10 – Блок-схема метода FoundBalls.

Метод ClearList служит для удаления шариков, координаты которых передаются через входной параметр. Входным параметром является: List<Ball> List, хранящий в себе список всех шариков, которые нужно удалить. Блок-схема алгоритма метода приведена на рисунке 2.11.

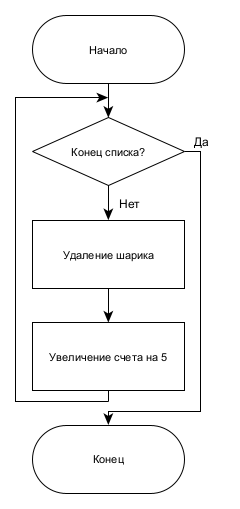


Рисунок 2.11. – Блок-схема метода ClearList.

Метод FoundIslands служит для поиска и добавления всех островов в список островов. Входным параметром является: Ball startBall, содержащий в себе координаты шарика, с которого будет осуществлен поиск островов. Выходным параметром является: List<Ball> FoundIslands, хранящий в себе список всех островов. Блок-схема алгоритма метода приведена на рисунке 2.12.

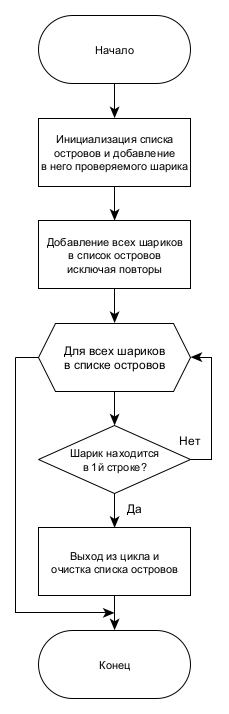


Рисунок 2.12. – Блок-схема метода FoundIslands.

Метод DeleteIslands служит для удаления всех островов. Входных параметров не имеет. Блок-схема алгоритма метода приведена на рисунке 2.13.

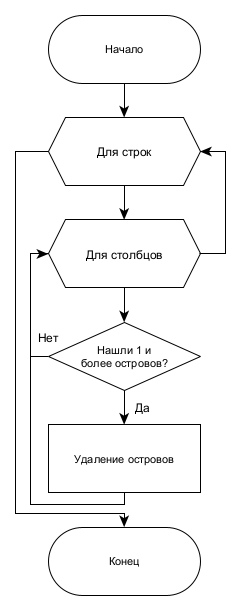


Рисунок 2.13. – Блок-схема метода DeleteIslands.

Метод OuterProc служит для удаления всех шариков одного цвета. Входным параметром является: Ball startBall, содержащий в себе координаты прилетевшего шарика. Блок-схема алгоритма метода приведена на рисунке 2.14.

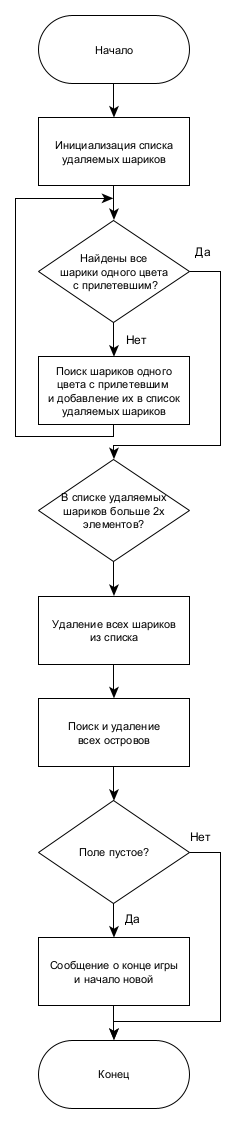


Рисунок 2.14. – Блок-схема метода OuterProc.

Метод OnPaint служит для отрисовки поля. Входным параметрам является: PaintEventArgs e. Блок-схема алгоритма метода приведена на рисунке 2.15.

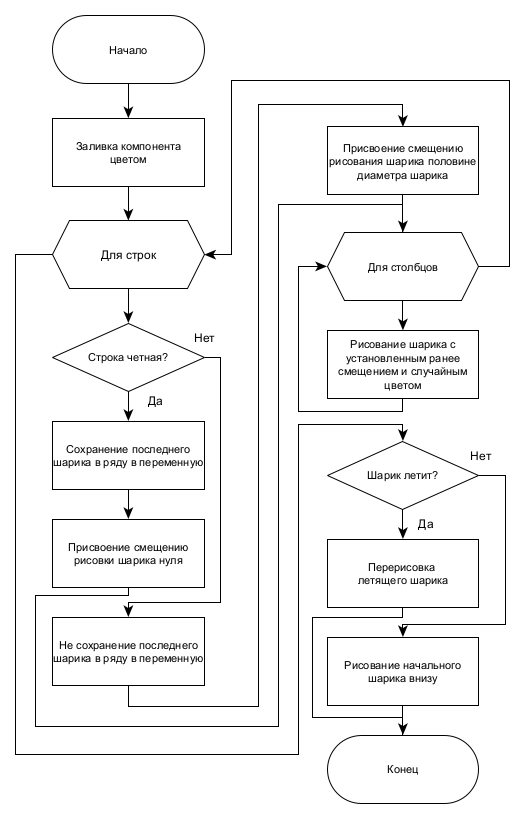


Рисунок 2.15. – Блок-схема метода OnPaint.

Метод TimerTick служит для расчета траектории полета шарика, а также для контроля за тем, чтобы шарик не вылетел за пределы игрового поля. Входными параметрами для него являются: object sender, EventArgs e. Блок-схема алгоритма метода приведена на рисунке 2.16.

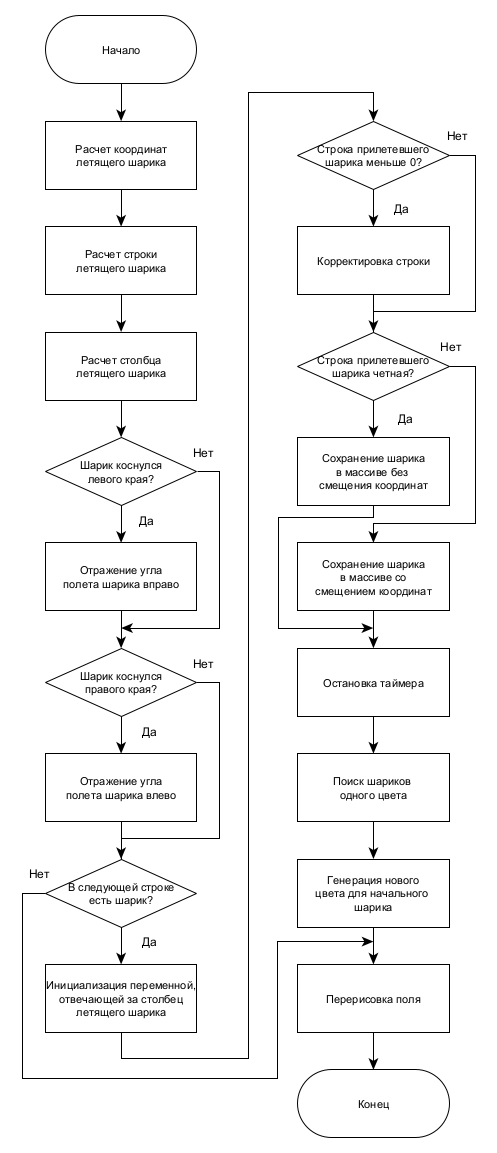


Рисунок 2.16. – Блок-схема метода TimerTick.

На основании составленных блок-схем был написан программный код. Код приведен в приложении А.

3 Технологический раздел

3.1 Отладка компонента

Отладка программы – это специальный этап в разработке программы, состоящий в выявлении и устранении программных ошибок, факт существования которых уже установлен.

Программные ошибки, как правило, делятся на три вида.

Первым видом является синтаксическая ошибка. Синтаксические ошибки относят к группе самых простых, так как синтаксис языка, как правило, строго формализован, и ошибки сопровождаются развернутым комментарием с указанием ее местоположения.

Второй вид – семантическая ошибка. Нарушение семантики той или иной конструкции, например передача функции параметров, не соответствующих ее аргументам.

Третий вид – логическая ошибка. Нарушение логики программы, приводящее к неверному результату. Это наиболее трудный для "отлова" тип ошибки, ибо подобного рода ошибки, как правило, кроются в алгоритмах и требуют тщательного анализа и всестороннего тестирования.

В больших и сложных программах логические ошибки и ошибки периода выполнения достаточно трудно отследить и обнаружить. В этих случаях вполне естественным является желание выполнить программу в интерактивном режиме, наблюдая за изменениями значений отдельных переменных или выражений. При этом желательно иметь возможность останавливаться в определенном месте программы и смотреть, что там происходит. Часто бывает полезно изменять значения некоторых переменных при выполнении программы. Это позволит повлиять на ее поведение и увидеть, в какую сторону оно изменилось. В каждой современной системе программирования существует специальное средство отладки программ, которое предоставляет все вышеперечисленные возможности — отладчик.

Существуют две взаимодополняющие технологии отладки. Первая – использование отладчиков. Вторая технология заключается в выводе текущего состояния программы с помощью расположенных в критических точках программы операторов вывода — на экран, принтер, громкоговоритель или в файл.

Методика отладки свойств и методов разработанного класса приведена в таблице 3.1.

Таблица 3.1 – методика отладки свойств и методов класса.

|  |  |  |
| --- | --- | --- |
| Имя свойства | Управляющее воздействие | Результат воздействия |
| OnMouseDown | Нажатие левой кнопки мыши по компоненту | Выстрел шариком |
| StartGame | Вызов процедуры | Первоначальное заполнение игрового поля шариками |
| EndGame | Вызов процедуры в случае, если игровое поле пустое | Демонстрация набранного счета и начало новой игры |
| Flight | Вызов процедуры при нажатии левой кнопки мыши по компоненту | Полет шарика в указанном направлении |
| DeleteNullBalls | Вызов процедуры после нахождения шариков вокруг прилетевшего | Удаление из списка несуществующих шариков |
| FoundBalls | Вызов процедуры после удаления “несуществующих” шариков | Поиск шариков одного цвета с прилетевшим |
| ClearList | Вызов процедуры в случае, если рядом лежат более 2х шариков одного цвета | Удаление шариков одного цвета |
| FoundIslands | Вызов процедуры после удаления шариков одного цвета | Добавление “острова” шариков в список |
| DeleteIslands | Вызов процедуры при обнаружении всех островов | Удаление всех “островов” |
| OuterProc | Вызов процедуры после прилета шарика | Удаление шариков одного цвета |

3.2 Инструкция по установке компонента

Чтобы добавить компонент в свой проект, прикладному программисту необходимо выполнить следующие действия. Для начала следует перейти в “Обозреватель решений” (Solution Explorer). “Обозреватель решений” изображен на рисунке 3.1.

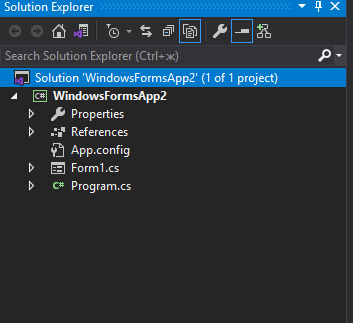


Рисунок 3.1 – “Обозреватель решений”.

Если “Обозреватель решений” закрыт, необходимо вызвать его из меню – Вид (View) командой “Обозреватель решений”. Команда “Обозреватель решений” из меню “Вид” представлена на рисунке 3.2.

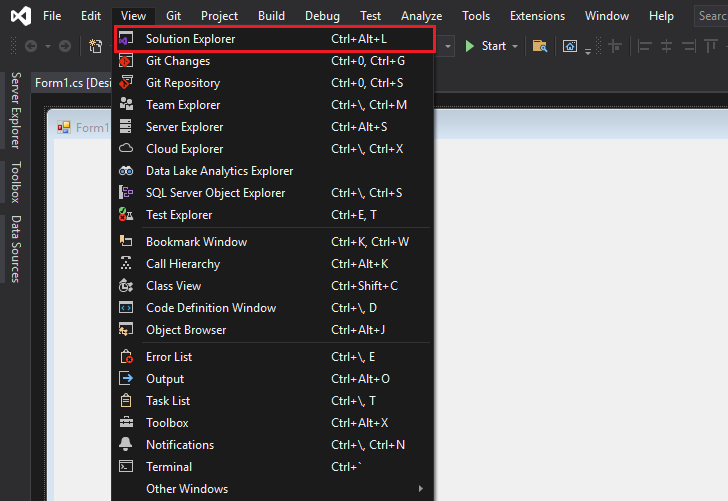


Рисунок 3.2 – Вызов “Обозревателя решений”.

Затем необходимо выполнить команду “Существующий проект” (Existing project) из групп команд “Добавить” (Add) контекстного меню, вызванного для текущего решения нажатием правой кнопки мыши по имени проекта, как это показано на рисунке 3.3.

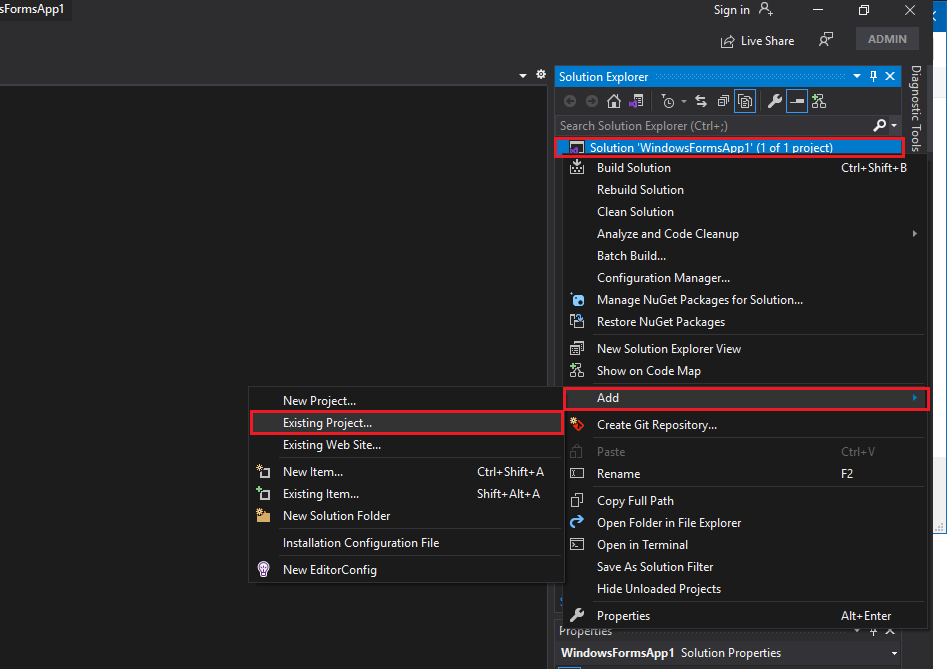


Рисунок 3.3 – Добавление файла через контекстное   
меню “Обозревателя решений”.

После этого среда разработки попросит выбрать файл с расширением “.csproj”. Пример диалогового окна с выбором файла представлен на рисунке 3.4.

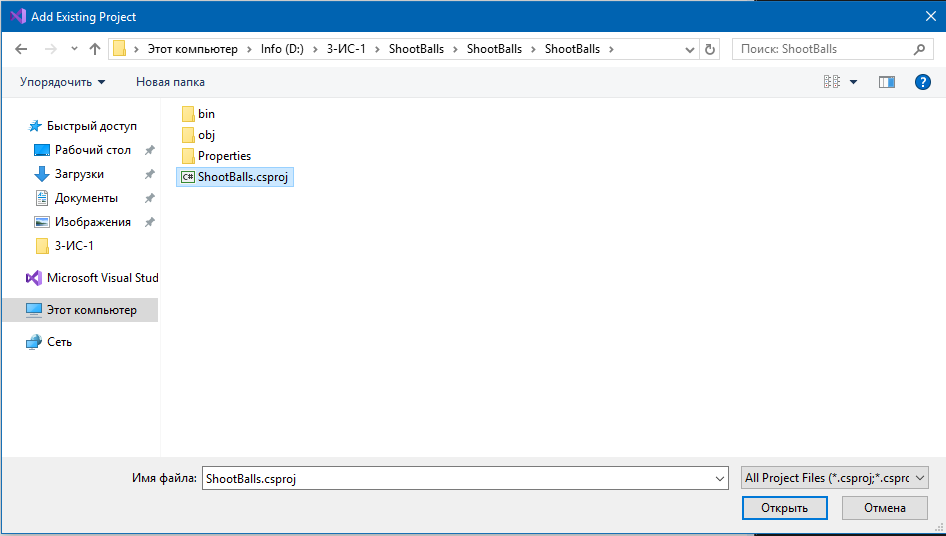


Рисунок 3.4 – Выбор файла в диалоговом окне.

В результате выбора файла в обозревателе решений добавится проект с классом, как это показано рисунке 3.5.

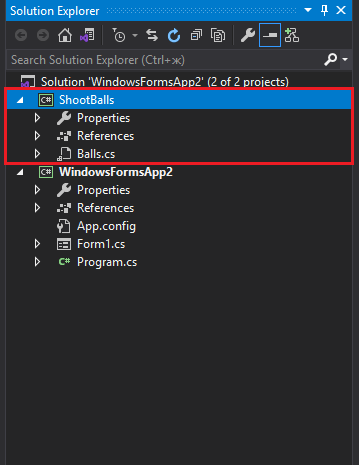


Рисунок 3.5 – Отображение добавленного проекта   
в обозревателе решений.

После успешного открытия проекта следует выполнить команду “Пересобрать решение” (Rebuild Solution) из контекстного меню, вызванного нажатием правой кнопки мыши по имени проекта. Команда “Пересобрать решение” представлена на рисунке 3.6.

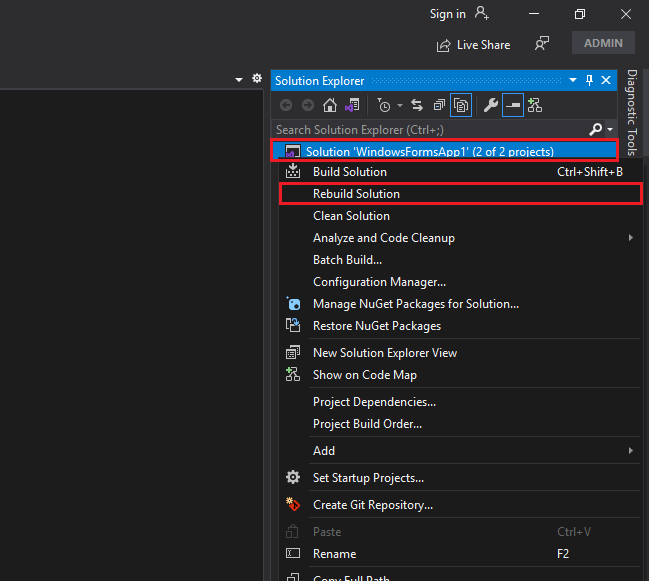


Рисунок 3.6 – Пункт “Пересобрать решение”.

После успешной сборки решения необходимо перейти в дизайнер формы путем двойного нажатия в обозревателе на “название формы.cs”. Пример перехода в дизайнер формы представлен на рисунке 3.7.

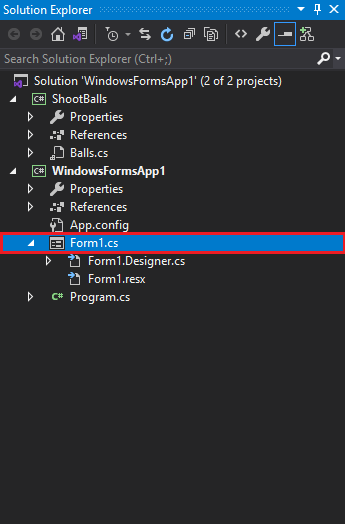


Рисунок 3.7 – Переход в дизайнер формы.

После перехода в дизайнер формы необходимо добавить компонент на форму. Для этого следует перейти в “Панель элементов” (ToolBox). “Панель элементов” изображена на рисунке 3.8.

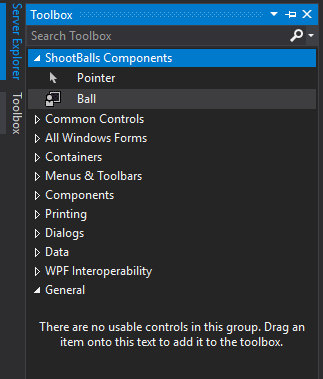


Рисунок 3.8 – Пункт “Панель элементов”.

Если “Панель элементов” закрыта, необходимо вызвать ее из меню “Вид” (View) - командой “Панель элементов”. Команда “Панель элементов” меню “Вид” представлена на рисунок 3.9.

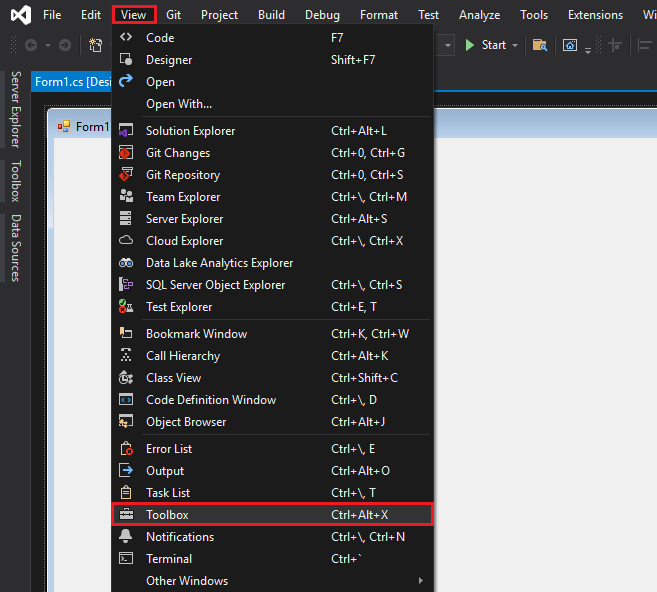


Рисунок 3.9 – Вызов “Панели элементов”.

Далее в “Панели элементов” необходимо найти установленный компонент и перетащить его на форму, как представлено на рисунке 3.10.

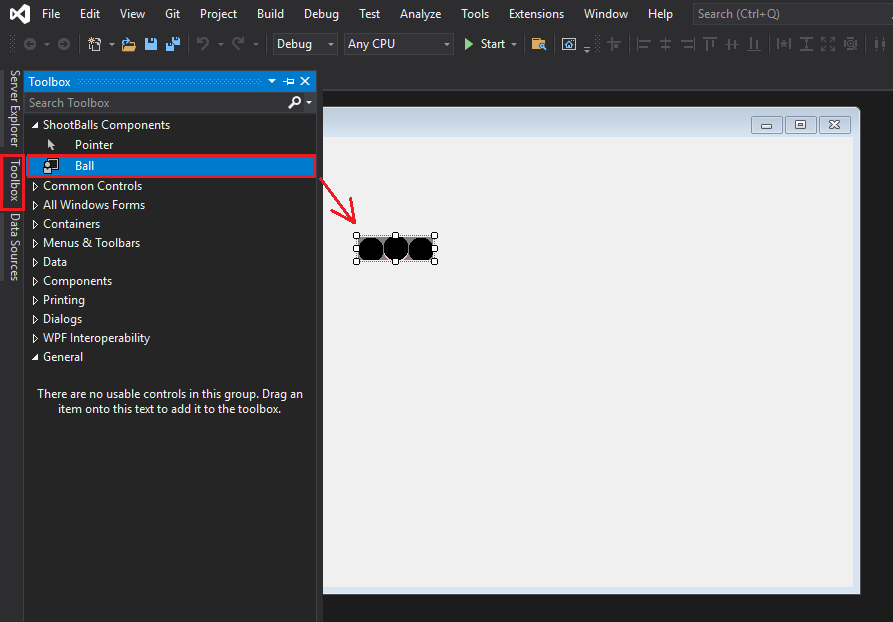


Рисунок 3.10 – Добавление установленного компонента на форму.

В результате выполнения всех вышеперечисленных действий, установленный компонент отобразится на форме и будет готов к работе. Пример отображения компонента на форме представлен на рисунке 3.11.

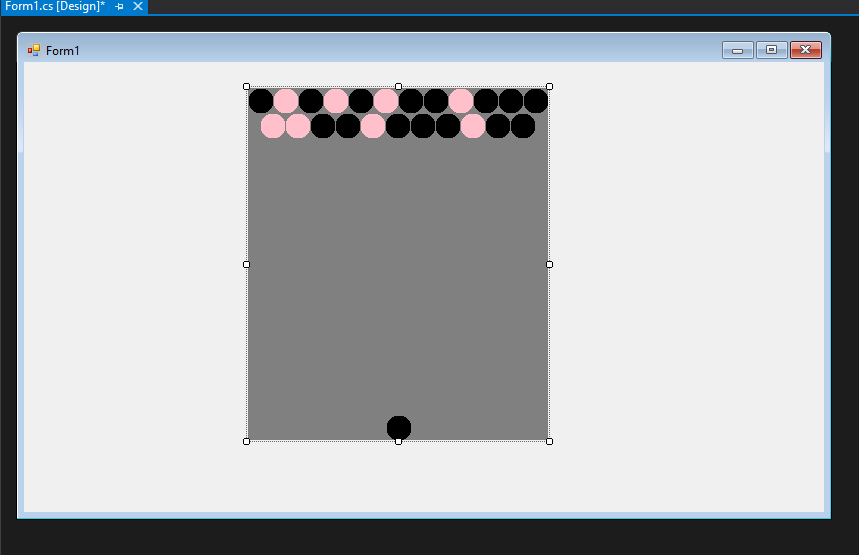


Рисунок 3.11 – Отображение установленного компонента на форме.

3.3 Инструкция по эксплуатации компонента

Для удобства использования разработанного визуального компонента вся инструкция по эксплуатации представлена в виде таблиц.

В таблице 3.2 представлены все свойства компонента.

Таблица 3.2 – свойства компонента.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя свойства | Тип данных | Тип доступа | Назначение |
| BallWidth | Int | Чт.\зп. | Позволяет установить размер шарика. |
| ColorsArr | List<Color> | Чт.\зп. | Позволяет задать цвета шариков. |
| Step | Int | Чт.\зп. | Определяет скорость полета шарика. |
| FieldSize | Int | Чт.\зп. | Определяет размеры игрового поля. |
| Score | Int | Чт. | Содержит в себе счет. |

В таблице 3.3 представлены все события компонента.

Таблица 3.3 – события компонента

|  |  |
| --- | --- |
| Имя события | Назначение |
| ChangeScore | Сообщает о том, что счет изменился. Вызов события после удаления шариков одного цвета и “островов”. |

В таблице 3.4 представлены все методы компонента.

Таблица 3.4 – методы компонента.

|  |  |  |
| --- | --- | --- |
| Имя | Входные и выходные параметры | Назначение |
| StartGame | - | Первоначальное заполнение игрового поля шариками. |
| EndGame | - | Демонстрация набранного счета и начало новой игры |

4 Раздел охраны труда

Безопасность жизнедеятельности — наука о комфортном и безопасном взаимодействии человека с техносферой, представляет собой область научных знаний, изучающая опасности, угрожающие человеку и разрабатывающие способы защиты от них в любых условиях обитания человека.

Для комфортной работы в помещении, где находится программист, должны быть определенные показатели температуры и влажности. Именно они создают микроклимат, который влияет на теплообмен с окружающей средой, самочувствие и работоспособность. В теплый период года в комнате, где работает программист, должно быть +20… 25°С, в холодный – +18… 21°С, а в переходный – +17… 21°С. Что касается влажности, то оптимальным показателем является от 40% до 60%. Чтобы организовать комфортные условия, уделяют внимание отопительной системе, вентиляции, кондиционированию воздуха. А также необходимо рационально спланировать режим дня с учетом времени года, чтобы происходило чередование труда с отдыхом.

Программист проводит на своем рабочем месте большую часть рабочего дня, поэтому организация этого пространства влияет и на его трудовую деятельность, и на его здоровье. Рабочая поза сидя вызывает минимальное утомление инженера-программиста. Рациональная планировка рабочего места предусматривает четкий порядок и постоянство размещения предметов, средств труда и документации. То, что требуется для выполнения работ чаще, должно быть расположено в зоне легкой досягаемости рабочего пространства. По этой причине важно правильно подобрать стол и другую мебель для работы программиста, позаботиться о микроклимате, вентиляции и хорошем освещении. Далее будут рассмотрены основные нюансы организации рабочего места инженера-программиста.

Подбор подходящей мебели помогает значительно увеличить производительность труда и снижает риск для здоровья программиста. При организации рабочего места важно, чтобы:

* мебель не ограничивала свободу движений в комнате и на рабочем месте, позволяла часто менять положение тела;
* все предметы были размещены оптимально (те, что нужны для работы чаще, должны быть ближе);
* рабочая поза не вызывала быстрого утомления;
* мебель соответствовала антропометрическим параметрам, регулировалась по высоте.

Основной мебелью рабочего места программиста являются кресло и стол. У кресла должна быть немного вогнутая поверхность и незначительный наклон спинки назад. Его высота должна изменяться, а вся конструкция не должна мешать свободе движений корпуса и рук. Желательно, чтобы у кресла имелись подлокотники.

Для комфортной работы стол должен удовлетворять следующим условиям:

* высота стола должна быть выбрана с учетом возможности сидеть свободно, в удобной позе, при необходимости опираясь на подлокотники;
* нижняя часть стола должна быть сконструирована так, чтобы программист мог удобно сидеть, не был вынужден поджимать ноги;
* поверхность стола должна обладать свойствами, исключающими появление бликов в поле зрения программиста;
* конструкция стола должна предусматривать наличие выдвижных ящиков (не менее 3 для хранения документации, листингов, канцелярских принадлежностей);
* высота рабочей поверхности рекомендуется в пределах 680-760мм;
* высота поверхности, на которую устанавливается клавиатура, должна быть около 650мм.

На рисунке 3.11 показан пример размещения основных и периферийных составляющих ПК на рабочем столе программиста., где 1 – сканер, 2 – монитор, 3 – принтер, 4 – поверхность рабочего стола, 5 – клавиатура, 6 – мышь.

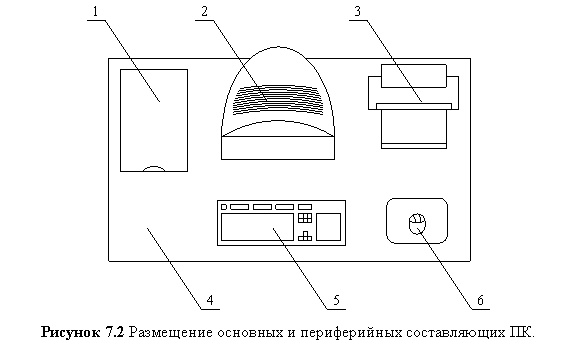


Рисунок 3.11 – Расположение основных и периферийных составляющих ПК.

При работе с вычислительной техникой могут возникнуть различные чрезвычайные ситуации. Наиболее вероятная чрезвычайная ситуация которая может возникнуть при работе с ПЭВМ – пожар, так как в современных ЭВМ очень высокая плотность размещения элементов электронных схем, в непосредственной близости друг от друга располагаются соединительные провода и кабели, при протекании по ним электрического тока выделяется значительное количество теплоты, при этом возможно оплавление изоляции и возникновение возгорания. Возникновение других видов ЧС – маловероятно.

Основным фактором, от которого зависит пожарная безопасность помещения, является автоматическое средство обнаружения пожаров. Оно позволяет оповестить дежурный персонал о пожаре и месте его возникновения. Данная система предназначена для обнаружения начальной стадии пожара и при необходимости включения автоматических систем пожаротушения и дымоудаления. Так как в помещении, в котором разрабатывается ПО, находится вычислительная техника и радиоаппаратура, то следует установить дымовой пожароизвещатель.

Исходя из норм пожарной безопасности, для машинного зала площадью до 100 м2 требуются следующие первичные средства пожаротушения:

* один углекислотный огнетушитель типа ОУ-5 или ОУ-8, с помощью которого можно тушить загорания различных материалов и установок напряжением до 1000 В;
* один порошковый огнетушитель.

При единовременном нахождении на этаже более 10 человек должны быть разработаны и вывешены на видных местах планы (схемы) эвакуации людей в случае пожара. План эвакуации должен включать в себя графическую и текстовую часть. Графическая часть представляет собой чертеж поэтажных планов здания, которые не должны загромождаться второстепенными деталями. Все работники должны пройти инструктаж по плану эвакуации в случае пожара, и должны уметь пользоваться схемой эвакуации и знать значения опознавательных знаков. Также необходимой мерой является инструктаж персонала по электробезопасности. Сюда же следует отнести правильную эксплуатацию техники, указанную в технических паспортах.

Для предотвращения поражения электрическим током предусмотрено наличие провода защитного заземления в электрической розетке, либо наличие заземляющего контура для внешнего заземления.

В случае поражения работника электрическим током, необходимо быстро освободить его от действия электрического тока. Приближаться к человеку, попавшему в зону поражения током, нужно минимальными шажками, чтобы не попасть под воздействие шагового напряжения. Шаговое напряжение – это напряжение, обусловленное электрическим током, протекающим в земле или токопроводящем полу, и равное разности потенциалов между двумя точками поверхности земли (пола), находящимися на расстоянии одного шага человека. Прикасаться к человеку, находящемуся под напряжением, опасно для жизни. Для освобождения пострадавшего от воздействия электрического тока следует воспользоваться сухой одеждой, доской или каким-либо другим предметом, не проводящим электрический ток, избегая при этом прикосновения к металлическим предметам и открытым частям тела. Далее необходимо оказать пострадавшему первую доврачебную помощь. Вызов скорой медицинской помощи обязателен независимо от того в сознании пострадавший или нет.

Заключение

В результате работы над проектом был получен визуальный компонент игра “Shoot Balls”, а также был получен опыт разработки компонентов в среде Visual Studio 2019 и документации.

Разработанный компонент позволит прикладным программистам использовать реализованные методы, не тратя время на их написание.

Как говорилось выше, визуальный компонент-игра в современном мире сможет найти применение в обучении детей, развитии логики и моторики.

В дальнейшем данный проект можно модифицировать, дополнив его различными алгоритмами стрельбы, удаления шариков и островов, а также отражения от поверхности.

Список использованных источников

1. <https://pandia.ru/text/77/412/193.php>
2. <https://www.cleverence.ru/articles/auto-busines/vidy-programmnogo-obespecheniya-kakie-byvayut-tipy-klassifikatsiya-primery/>
3. <https://it-black.ru/nizkourovnevye-i-vysokourovnevye-jazyki-programmirovanija/>
4. <https://studbooks.net/2258619/informatika/opisanie_sredy_razrabotki_microsoft_visual_studio>
5. <https://webformyself.com/sreda-dlya-veb-razrabotki-visual-studio/>
6. <https://streletzcoder.ru/5-prichin-otkazatsya-ot-delphi-7/>
7. <https://multi-lady.ru/plyusy-i-minusy-yazyka-delphi/>
8. <https://sales-generator.ru/blog/dekompozitsiya-tseley/#1>
9. <https://wikichi.ru/wiki/Debugging>
10. <https://skarlupka.ru/articles.php?id=33>
11. <http://www.taurion.ru/outlook/4/32>
12. <https://infourok.ru/user/3482220/blog/bezopasnost-zhiznedeyatelnosti-128391.html>
13. <https://vplate.ru/programmist/rabochee-mesto/>

Приложение А

Код класса Balls

using ShootBalls;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace Balls

{

public partial class Balls : Control

{

//размеры игрового поля (в шариках)

public const int RowCount = 19;

public const int ColCount = 12;

//матрица шариков

private int[,] BallsMatrix;

//список цветов шариков

public List<Color> ColorsArr { get; set; }

Random rnd = new Random();

//размер шарика

public int \_ballWidth = 25;

//Координаты летящего шарика

private int ShotX, ShotY;

//private int ShotY; //= Height - \_ballWidth;

private double k, d;

System.Windows.Forms.Timer MyTimer;

//флаг, отвечающий, был произведен выстрел или нет

private bool BallShot;

Brush CurrentBrush;

//переменная, обозначающая смещение шарика, которое зависит от ряда, в который прилетел шарик

int Offset;

Point CurrentCircleCords;

int NextBallColor;

//шаг полета шарика

public int Step = 5;

public List<Ball> BallsList;

public delegate void ScoreHandler();

public event ScoreHandler Notify;

//переменная, отвечающая за счет

public int Score { get; private set; }

//акксессор и мьютатор для поля ballWidth

public int ballWidth

{

get

{

return \_ballWidth;

}

set

{

if (value != \_ballWidth)

if (value >= 5 && value <= 25)

\_ballWidth = value;

}

}

//пассивный конструктор

public Balls()

{

//начальное заполнение списка цветов

ColorsArr = new List<Color>();

ColorsArr.Add(Color.Transparent);

ColorsArr.Add(Color.Black);

ColorsArr.Add(Color.Pink);

ColorsArr.Add(Color.Red);

ColorsArr.Add(Color.Blue);

ColorsArr.Add(Color.Yellow);

//старт игры

StartGame();

DoubleBuffered = true;

}

//метод расчета траектории полета шарика

private void TimerTick(object sender, EventArgs e)

{

int OldShotX = ShotX;

ShotX = k != 0 ? (int)((ShotY - d) / k) : CurrentCircleCords.X;

ShotX = ShotX < 0 ? -ShotX : ShotX;

ShotY -= Step;

BallShot = true;

int Row = ShotY / \_ballWidth;

int Col1 = ShotX / \_ballWidth;

int Col2 = (ShotX + 3 \* \_ballWidth / 4) / \_ballWidth;

if ((ShotX <= Step / 2) && (ShotX < OldShotX))

{

d -= k \* ShotX;

k = -k;

}

if ((ShotX >= Width - \_ballWidth - Step / 2) && (ShotX > OldShotX))

{

d += k \* ShotX;

k = -k;

}

if ((Row > 0) && (((Row % 2 != 0) && (BallsMatrix[Row - 1, Col2] != 0)) ||

((Row % 2 == 0) && (BallsMatrix[Row - 1, Col1] != 0))))

{

int Column;

if (Row < 0) Row = 0;

if (Row % 2 != 0)

{

BallsMatrix[Row, Col1] = NextBallColor;

Column = Col1;

}

else

{

BallsMatrix[Row, Col2] = NextBallColor;

Column = Col2;

}

(sender as System.Windows.Forms.Timer).Stop();

BallShot = false;

OuterProc(new Ball(Row, Column));

NextBallColor = rnd.Next(1, ColorsArr.Count);

}

Invalidate();

}

//метод начальной настройки поля

public void StartGame()

{

BallsMatrix = new int[RowCount, ColCount];

int LastCol;

//первоначальное заполнение стакана шариками

for (int Row = 0; Row < 7; Row++)

{

//проверка, нужно ли заполнять шарик в последней колонке или нет

if (Row % 2 != 0) LastCol = BallsMatrix.GetLength(1) - 1; else LastCol = BallsMatrix.GetLength(1);

for (int Col = 0; Col < LastCol; Col++)

BallsMatrix[Row, Col] = rnd.Next(1, ColorsArr.Count);

}

//инициализация таймера

MyTimer = new System.Windows.Forms.Timer();

MyTimer.Interval = 5;

MyTimer.Enabled = true;

MyTimer.Tick += TimerTick;

MyTimer.Stop();

//Рандомный выбор цвета для шарика, которым будет осуществляться выстрел

CurrentBrush = new SolidBrush(ColorsArr[rnd.Next(1, ColorsArr.Count)]);

Size = new Size(ColCount \* \_ballWidth, 500);

MaximumSize = new Size(ColCount \* \_ballWidth, 500);

NextBallColor = 1;

Score = 0;

}

//метод определения конца игры

public bool EndGame()

{

int cnt = 0;

for (int Row = 0; Row < 7; Row++)

for (int Col = 0; Col < BallsMatrix.GetLength(1); Col++)

if (BallsMatrix[Row, Col] > 0) { cnt++; }

if (cnt == 0) return true; else return false;

}

//метод рисования компонента

protected override void OnPaint(PaintEventArgs e)

{

base.OnPaint(e);

Brush b = new SolidBrush(Color.Gray);

e.Graphics.FillRectangle(b, 0, 0, Width, Height);

if (ColorsArr != null)

{

for (int Row = 0; Row < BallsMatrix.GetLength(0); Row++)

{

int LastCol;

if (Row % 2 == 0)

{

LastCol = BallsMatrix.GetLength(1);

Offset = 0;

}

else

{

LastCol = BallsMatrix.GetLength(1) - 1;

Offset = \_ballWidth / 2;

}

for (int Col = 0; Col < LastCol; Col++)

{

CurrentBrush = new SolidBrush(ColorsArr[BallsMatrix[Row, Col]]);

if (BallsMatrix[Row, Col] != 0) e.Graphics.FillEllipse(CurrentBrush, Col \* \_ballWidth + Offset, Row \* \_ballWidth, \_ballWidth, \_ballWidth);

}

}

}

CurrentBrush = new SolidBrush(ColorsArr[NextBallColor]);

//если шарик летит, то перерисовка шарика с новыми координатами иначе рисование шарика внизу

if (BallShot)

{

e.Graphics.FillEllipse(CurrentBrush, ShotX, ShotY, \_ballWidth, \_ballWidth);

}

else

{

e.Graphics.FillEllipse(CurrentBrush, Width / 2 - \_ballWidth / 2, Height - \_ballWidth, \_ballWidth, \_ballWidth);

}

}

//событие нажатия на левую кнопку мыши

protected override void OnMouseDown(MouseEventArgs e)

{

//если шарик не летит, то выстрел шарика в указанную сторону

if (!BallShot) Flight(e.Location);

}

//событие нажатия на клавишу клавиатуры

protected override void OnKeyDown(KeyEventArgs e)

{

if (!BallShot)

if (e.KeyCode == Keys.Space)

{

NextBallColor = rnd.Next(1, ColorsArr.Count);

Invalidate();

}

}

//процедура полета шарика

protected void Flight(Point CurrentCircleCords)

{

CurrentCircleCords.X -= \_ballWidth / 2;

CurrentCircleCords.Y -= \_ballWidth / 2;

ShotX = Width / 2 - \_ballWidth / 2;

ShotY = Height - \_ballWidth;

BallShot = true;

if (CurrentCircleCords.X != ShotX)

{

k = 1.0 \* (CurrentCircleCords.Y - ShotY) / (CurrentCircleCords.X - ShotX);

d = ShotY - k \* ShotX;

}

else

{

k = 0;

}

MyTimer.Start();

}

//метод очистки несуществующих шариков

private List<Ball> DeleteNullBalls(List<Ball> List)

{

for (int item = List.Count - 1; item >= 0; item--)

{

if (BallsMatrix[List[item].Row, List[item].Col] == 0)

{

List.Remove(List[item]);

}

}

return List;

}

//метод поиска шариков вокруг прилетевшего

private List<Ball> FoundBalls(int startRow, int startCol)

{

BallsList = new List<Ball>();

if (startRow >= 0 && startRow < BallsMatrix.GetLength(0) && startCol >= 0 && startCol < BallsMatrix.GetLength(1))

{

if (startCol > 0)

{

BallsList.Add(new Ball(startRow, startCol - 1));

}

if (startCol < BallsMatrix.GetLength(1) - 1)

{

BallsList.Add(new Ball(startRow, startCol + 1));

}

if (startRow > 0)

{

if (startRow % 2 == 0)

{

if (startCol > 0) BallsList.Add(new Ball(startRow - 1, startCol - 1));

if (startCol < BallsMatrix.GetLength(1) - 1) BallsList.Add(new Ball(startRow - 1, startCol));

}

else

{

BallsList.Add(new Ball(startRow - 1, startCol));

if (startCol < BallsMatrix.GetLength(1) - 1) BallsList.Add(new Ball(startRow - 1, startCol + 1));

}

}

if (startRow < BallsMatrix.GetLength(0) - 1)

{

if (startRow % 2 == 0)

{

if (startCol > 0) BallsList.Add(new Ball(startRow + 1, startCol - 1));

if (startCol < BallsMatrix.GetLength(1) - 1) BallsList.Add(new Ball(startRow + 1, startCol));

}

else

{

BallsList.Add(new Ball(startRow + 1, startCol));

if (startCol < BallsMatrix.GetLength(1) - 1) BallsList.Add(new Ball(startRow + 1, startCol + 1));

}

}

}

DeleteNullBalls(BallsList);

return BallsList;

}

//метод удаления шариков из списка

private void ClearList(List<Ball> List)

{

foreach (Ball item in List)

{

BallsMatrix[item.Row, item.Col] = 0;

Score += 5;

}

List.Clear();

}

//метод поиска островов

private List<Ball> FoundIslands(Ball startBall)

{

List<Ball> FoundIsland = new List<Ball> { startBall };

int i = 0;

do

{

foreach (Ball item in FoundBalls(FoundIsland[i].Row, FoundIsland[i].Col))

{

if (!FoundIsland.Exists(P => P.Row == item.Row && P.Col == item.Col))

{

FoundIsland.Add(item);

}

}

} while (++i < FoundIsland.Count);

if (FoundIsland.Find(Ball => Ball.Row == 0) != null) return new List<Ball>();

return FoundIsland;

}

//метод удаления островов

private void DeleteIslands()

{

for (int Row = 0; Row < BallsMatrix.GetLength(0); Row++)

{

for (int Col = 0; Col < BallsMatrix.GetLength(1); Col++)

{

List<Ball> Island = FoundIslands(new Ball(Row, Col));

if (Island.Count > 0)

{

ClearList(Island);

}

}

}

}

//метод удаления шариков одного цвета

private void OuterProc(Ball startBall)

{

List<Ball> DeleteBalls = new List<Ball>();

int i = 0;

DeleteBalls.Add(startBall);

do

{

foreach (Ball item in FoundBalls(DeleteBalls[i].Row, DeleteBalls[i].Col))

{

if ((BallsMatrix[DeleteBalls[0].Row, DeleteBalls[0].Col] == BallsMatrix[item.Row, item.Col]) &&

!DeleteBalls.Exists(P => P.Row == item.Row && P.Col == item.Col))

{

DeleteBalls.Add(item);

}

}

} while (++i < DeleteBalls.Count);

if (DeleteBalls.Count > 2) ClearList(DeleteBalls);

DeleteBalls.Clear();

DeleteIslands();

Invalidate();

if (EndGame())

{

MessageBox.Show("Конец игры!\n" + "Ваш счет: " + Score);

StartGame();

}

}

}

}

Код класса Ball

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ShootBalls

{

public class Ball

{

public int Row;

public int Col;

public Ball(int Row, int Col)

{

this.Row = Row;

this.Col = Col;

}

}

}