**ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ЯРОСЛАВСКОЙ ОБЛАСТИ**

государственное профессиональное образовательное учреждение   
Ярославской области

Рыбинский полиграфический колледж

**КУРСОВОЙ ПРОЕКТ**

|  |  |
| --- | --- |
| Разработка класса визуального компонента-игры «Arcanoid» | |
|  | |
| по дисциплине | Объектно-ориентированное программирование |
|  | |

Пояснительная записка

|  |
| --- |
| КП.0902.12.000000.00 ПЗ |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы | 4-ИС-2 |  |  |  | Пахомов И А |
|  | *(Код учебной группы)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |
| Руководитель | преподаватель |  |  |  | Смирнов В Б |
|  | *(Должность, звание)* |  | *(Подпись, дата)* |  | *( И.О.Фамилия)* |
| Нормоконтроль | преподаватель |  |  |  | Смирнов В Б |
|  | *(Должность, звание)* |  | *(Подпись, дата)* |  | *(И.О.Фамилия)* |

г. Рыбинск

2022

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc113480419)

[1 Исследовательский раздел 5](#_Toc113480420)

[2 Конструкторский раздел 8](#_Toc113480421)

[2.1 Математическая модель 8](#_Toc113480422)

[2.2 Разработка структуры компонента 9](#_Toc113480423)

[2.3 Разработка графического интерфейса компонента 12](#_Toc113480424)

[2.4 Описание входных и выходных данных компонента 14](#_Toc113480425)

[2.5 Разработка алгоритмов компонента 16](#_Toc113480426)

[3 Технологический раздел 26](#_Toc113480427)

[3.1 Отладка компонента 26](#_Toc113480428)

[3.2 Инструкция по установке компонента 29](#_Toc113480429)

[3.3 Инструкция по эксплуатации компонента 34](#_Toc113480430)

[4 Раздел охраны труда 35](#_Toc113480431)

[Заключение 37](#_Toc113480432)

[Список источников 38](#_Toc113480433)

[Приложение А 39](#_Toc113480434)

Введение

На сегодняшний день вычислительная техника тесно связана практически со всеми людьми на планете. Вычислительная техника, упрощает жизнь в десятки раз. С её помощью человек может развлечься, найти буквально все, что хочет в глобальной сети Интернет. В данный момент, человек не может представить жизнь без всех удобств, что дает вычислительная техника.

Вычислительная техника, не всегда была такой удобной и простой для понимания. Все начиналось с машинного кода, на основе которого работают языки программирования, способствующие работе современных программ и приложений. Упомянутые выше языки программирования служат для упрощения программного кода. Первый язык программирования «Ассемблер» – низкоуровневый, создан по принципу «Одна инструкция – одна строка». Низкоуровневые языки, ориентированы на конкретный тип процессора и учитывают его особенности. Их появление, действительно облегчило написание кода, так как вместо единиц и нулей, можно было писать программу командами, состоящую из символов, приближенных к разговорному языку[1].

Наиболее важной проблемой, с которыми столкнулись разработчики языков второго поколения, стала задача убедить клиентов в том, что созданный компилятором код выполняется достаточно хорошо, чтобы оправдать отказ от программирования на ассемблере. Скептицизм по поводу возможности создания эффективных программ с помощью автоматических компиляторов был довольно распространён, поэтому разработчикам таких систем должны были продемонстрировать, что они действительно могут генерировать почти такой же эффективный код, как и при ручном кодировании, причём практически для любой исходной задачи. Но сложность разработки в нём больших программных комплексов, привела к появлению языков третьего поколения — языков высокого уровня. Такие языки были машинно-независимы, для каждого такого языка был компилятор. В связи с ростом сложности программ и развитием программных средств, возникла необходимость увеличить производительность труда программистов, что привело к разработке структурного программирования. Следующим достижением были [процедуры](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), то есть, если есть задача, которая выполняется несколько раз, то её можно объявить как функцию или как процедуру и в выполнении программы просто вызывать её. Программный код становится меньше. Следующим достижением было объединение разнородных данных, которые используются в программе в связке, в структуры. С развитием технологий родилось объектно-ориентированное программирование. При использовании структур данных в программе вырабатываются и соответствующие им функции для работы с ними. Это привело к мысли их объединить и использовать совместно, так появились классы. Теперь программирование можно было разбить на классы и тестировать не всю программу, состоящую из 10 000 строк кода, а разбить программу на 100 классов, и тестировать каждый класс. Объектно-ориентированное программирование сочетает лучшие принципы структурного программирования с новыми концепциями [инкапсуляции](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [полиморфизма подтипов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC_%D0%BF%D0%BE%D0%B4%D1%82%D0%B8%D0%BF%D0%BE%D0%B2) и [наследования](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))[2].

В ходе работы над курсовым проектом требуется разработать компонент-игру «Arcanoid» для одной из современных сред разработки программного обеспечения. Планируется, что компонент-игра будет реализовывать базовую версию игры «Arcanoid», будет выполнять все необходимое для полноценного игрового процесса. Актуальность проекта, прежде всего, заключается в возможности использовать его результаты в прикладных программах. Программистам не потребуется писать программный код с нуля, они смогут посмотреть где и как использовался метод или функция, достаточно понять программный код и взять себе необходимое.

Как таковая игра будет полезна для детей дошкольного и младшего школьного возраста, она позволит им потратить некоторое время, также поможет развить логику, ловкость рук и реакцию. Для начинающих программистов данный проект будет полезен тем, что позволит взглянуть на возможности, которые им предстоят в будущем, это заинтересует их обучатся программированию.

1 Исследовательский раздел

Для реализации проекта существуют две наиболее подходящие среды программирования - Borland Delphi 7 и Visual Studio 2019. Рассмотрим Borland Delphi 7. Среда не новая, легкая в понимании, практичная, есть возможность использовать только один язык программирования и к сожалению, среда устарела.

Visual Studio 2019 – это новая среда программирования с удобным интерфейсом и возможностью писать код на многих языках программирования. Онавключает в себя [редактор исходного кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0) с поддержкой технологии [IntelliSense](https://ru.wikipedia.org/wiki/IntelliSense) и возможностью простейшего [рефакторинга кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3). Встроенный [отладчик](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) может работать как отладчик уровня исходного кода, так и в качестве отладчика машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). В отличии от других сред Visual Studio более удобен и практичен, с этой средой намного приятнее работать. У этой среды есть минус – не все ошибки, имеют четкое решение[3].

Сравним среды программирования, начиная с Borland Delphi7. Плюсы среды: работает на старых компьютерах, занимает не много места на диске, работает на Microsoft Windows 2000 и старше. Система предсказуема, то есть имеет решения на все ошибки.

Минусы среды Borland Delphi7: при компилировании имеет достаточно большой размер .exe фалов на диске, писать код можно только на одном языке (Delphi). В целом среда неплохая, для своих задач будет хорошо справляться.

Плюсы Visual Studio 2019: совершенно новая система, удобный интерфейс и дизайн, много полезных дополнений, можно писать на многих языках, к примеру на C#, C++, F#. Среда постоянно обновляется.

Минусы Visual Studio 2019: присутствуют ошибки, на которые так и не найдены решения, не бесплатна, занимает большой объем на диске.

Для реализации компонента-игры «Arcanoid» выбрана среда программирования Visual Studio 2019. Среда имеет максимум преимуществ и минимум недостатков.

В качестве родительского класса в проекте будет выбран класс Control. Класс отвечает за визуальное представление. OnPaint - один из методов класса Control, он не реализует функциональность рисования, но вызывает делегаты событий, зарегистрированные в событии [Paint](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.control.paint). Также есть метод Invalidate, который вызывает процесс перерисовки элементов управления. Проект будет соответствовать требованиям графического интерфейса операционных систем Microsoft Windows.

В глобальной сети «Интернет» есть аналоги игры «Arcanoid». Рассмотрим «Arcanoid базовый» с сайта <https://play.arkanoid.online/> [4]. Игрок контролирует небольшую платформу-ракетку, которую можно передвигать горизонтально от одной стенки до другой, подставляя её под шарик, предотвращая его падение вниз. Удар шарика по кирпичу приводит к разрушению кирпича. После того как все кирпичи на данном уровне уничтожены, происходит переход на следующий уровень, с новым набором кирпичей.

Есть и аналоги с некоторыми разнообразиями: определённые кирпичи нужно ударять несколько раз, иногда появляются летающие враги, от которых отталкивается шарик, удар по некоторым кирпичам приводит к выпадению из них капсул-призов — приз активируется, если поймать такую капсулу ракеткой. У таких аналогов есть один недостаток – они все требуют доступ в Интернет.

Проект будет насыщен всей базовой логикой: управление платформой с помощью клавиатуры, генерация кирпичиков будет произвольно располагать кирпичики на экране, движение шарика по правильной траектории и счёт, который будет обновляться, если произошло столкновение с кирпичиком. Также будет реализована логика адаптивности компонента при изменении его размера. Отрисовка игрового поля, будет происходить с помощью метода OnPaint.

Основным процессом для реализации является движение шарика по правильной траектории. Также движение шарика является основной проблематикой игрового процесса. Заставить его двигаться по траектории, которая получается математическими расчетами довольно сложно.

2 Конструкторский раздел

2.1 Математическая модель

За движение шарика будет отвечать метод, в котором будут происходить проверки выхода шарика за пределы поля, отскок шарика от платформы, отскок от кирпичика, отскок от стенки.

Скорость полета шарика будет зависеть от интервала обновления таймера. Чем больше интервал, тем медленнее летит шарик, сама скорость будет изменяемой, благодаря интервалу таймера. Угол отражения шарика будет зависеть от угла его падения при столкновении с платформой, кирпичиком или стенкой. Если объект стоит на месте, то угол падения равен углу отражения, если же объект будет двигаться, то угол падения увеличивается или же уменьшается. После каждого обновления, будет вызываться метод проверки столкновении шарика. Игра окончена в том случае, если пользователь не поймал мяч платформой.

Изначально платформа с шариком будут рисоваться в центре игрового поля. Блоки рисуются в верхней части игрового поля, с каждым пятым сломанным блоком, будет добавляться новая линия блоков, все блоки будут цвета, который укажет пользователь. Движение платформы будет происходить по координате X, сама платформа будет иметь цвет, который укажет пользователь. За движение шарика будут отвечать координаты X и Y, цвет шарика задаётся пользователем.

2.2 Разработка структуры компонента

Для описания процесса проектирования компонента будет использоваться путь проведения серии декомпозиций, начиная с «черной сферы». Построение «черной сферы» поможет в дальнейших этапах разработки компонента. Черная сфера – это система внутреннее устройство которой неизвестно, в ней доступны лишь входные и выходные параметры. Модель «Черная сфера» изображена на рисунке 2.1

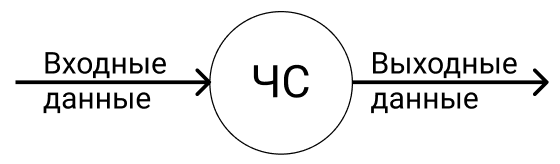


Рисунок 2.1 – Модель «Черная сфера»

В компоненте-игре будут присутствовать такие входные параметры: размеры, цвета, управляющее воздействие пользователя. На выходе будем получать: изображение на экране и счёт. Размеры будут редактироваться с помощью манипулятора типа «мышь», цвета задаются пользователям, управляющее воздействие пользователя будет осуществляться с помощью клавиатуры. Счёт и изображение будут выводится на экран. Представление класса в виде модели «Черная сфера» изображено на рисунке 2.2.

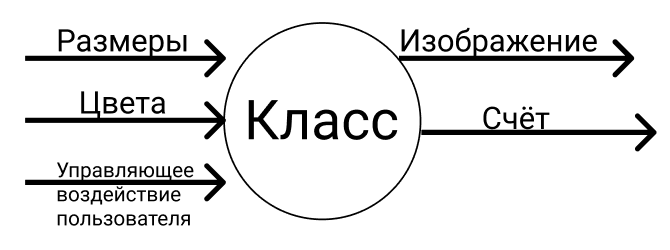


Рисунок 2.2 – Представление класса в виде модели «Черная сфера»

Декомпозиция – операция мышления, состоящая в разделении целого на части. Декомпозиция поможет правильно понять суть того или иного свойства или метода.

Получим первый этап декомпозиции с выявлением основных внутренних структур данных, необходимых для хранения информации компонента. От изменения размеров компонента будут зависеть размеры игровых элементов. Изменения цветов будут вызывать перерисовку компонента и вывод его на экран с указанным цветом. Обработка нажатия клавиш будет исполняться событием, которое вызывается при нажатии определенной клавиши, тем самым меняя направление платформы по оси X. Событие отвечающие за изменение счёта, будет выводить его на экран. Первый этап декомпозиции представлен на рисунке 2.3



Рисунок 2.3 - Первый этап декомпозиции

После проведения первого этапа декомпозиции были получены основные структуры данных, необходимых для хранения информации, и процессы компонента, которые позволят построить базовую логику.

Для получения более точных структур данных и процессов компонента проведем второй этап декомпозиции. Управляющее воздействие пользователя будет влиять на следующие методы: запуск таймера, остановка таймера, изменение направление по нажатию клавиш. Метод запуска таймера изменит состояние игры и запустит таймер, при этом все элементы установятся в начальные позиции, а также произойдет обновление экрана. Метод остановки таймера также изменит состояние игры, остановит таймер и отрисует новое положение блоков. При обновлении таймера происходит проверка статуса игры и проверка столкновения с препятствием. В методе проверки коллизии происходит обновление счета, а также при выходе шарика за границы, вызывается остановка таймера. При изменении цвета элементов, компонент будет перерисовываться в соответствии с заданными цветами. Размеры компонента и размеры его элементов (карта, блоки, шарик, платформа) будут зависеть от изменения размеров окна формы пользователем. Результат второго этапа декомпозиции представлен на рисунке 2.4. и на плакате КП.0902.11.000000.01 ПЛ.

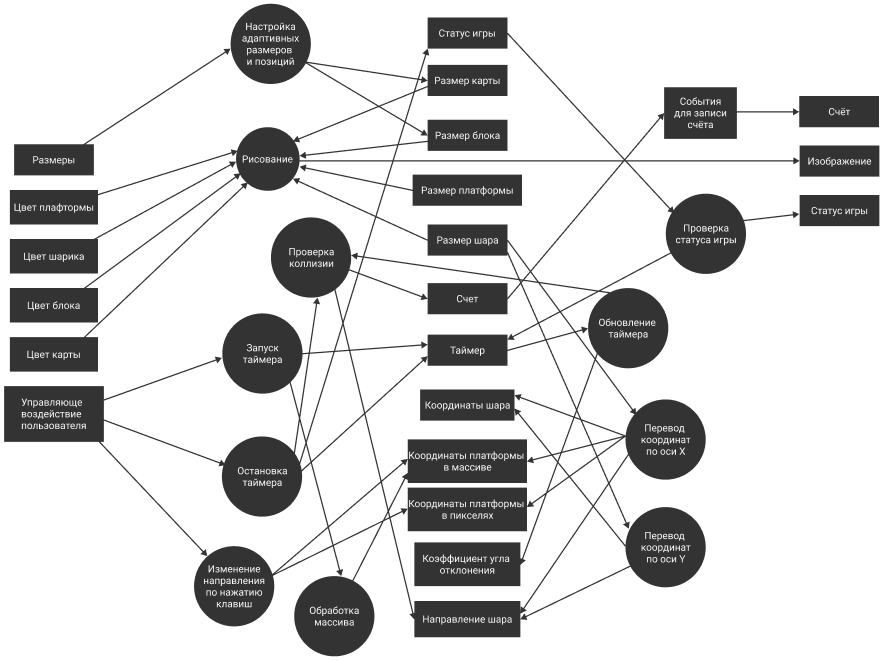


Рисунок 2.4 – Результат второго этапа декомпозиции

2.3 Разработка графического интерфейса компонента

Интерфейсы современных пользовательских программ имеют технологию WIMP – Windows, Icons, Mouse, Pop-up. Получается так, что основными элементами графических интерфейсов являются окна, пиктограммы, компоненты ввода-вывода и мышь. Окно – это ограниченная рамкой область экрана, которая может менять размеры и местоположение в пределах экрана. Окна можно разделить на пять категорий: основные, дочерние, окна диалога, информационные, окна меню. Пиктограмма – это небольшое окно с графическим изображением. Различают следующие пиктограммы: программные пиктограммы, пиктограммы дочерних окон, пиктограммы панели инструментов, пиктограммы объектов. Прямое манипулирование объектами- это возможность замены команды воздействия на некоторый объект физическим действием в интерфейсе, осуществляемым с помощью мыши. При этом любая область экрана рассматривается как адресат, который может быть активизирован при подведении курсора и нажатии клавиши мыши. По реакции на воздействие различают следующие типы адресатов: указание и выбор, буксировка и «резиновая нить» (перенос объекта и его границ), экранные кнопки. Для реализации диалогов, управляемых пользователем, применяют меню различных видов. Для реализации диалогов, управляемых системой, обычно используют диалоговые окна.

Развитие графических интерфейсов Windows имеет большую историю. Windows 1.0 - особенностью этой операционной системы было наличие окон, которые можно было расположить друг около друга и возможность управления манипулятором типа “мышь”. В следующей версии операционной системы Windows (Windows 2.0) окна могли перекрывать друг друга, для отображения интерфейса использовались 16 цветов, также появилось функции “свернуть” и “развернуть”. В Windows 3.0 пользователь мог любоваться 256-ю цветами, что однозначно делало работу более приятной, хотя основные элементы интерфейса остались без существенных изменений [5]. Windows 95-98-98SE-2000-ME-XP-Vista-7 закрепляют успех и позиции Microsoft на рынке операционных систем. В этом ряду стоит выделить, наверное, Windows Vista – в плане развития графического интерфейса она не выделяется из общего ряда, но прохладно принята общественностью. Windows 8 – пионер операционных систем с оконным интерфейсом, «тайлами» и возможностью управления с помощью мыши, клавиатуры и сенсорного экрана (пальцами) [6].

Визуальный компонент должен иметь концепцию графических образов и набор приемов управления в соответствии с теми, что используются в операционной системе Windows. Это облегчит действия конечного пользователя и сэкономит время. Также будет обеспечена возможность использования манипулятора типа “мышь” для управления параметрами и настройки компонента.

Визуальный компонент будет выглядеть в виде поля, где всегда высота больше, чем ширина. Также на поле будут присутствовать кирпичики, занимающее от высоты поля. Платформа, будет располагаться на оси X - по центру, а по оси Y - от высоты игрового поля. Движение по оси X будет осуществляться в соответствии с нажатой пользователем клавишей. В начале игры на платформе будет располагаться шарик, который в ходе игры будет двигаться, сбивая блоки. Компонент будет адаптивным, то есть будет иметь возможность подстраиваться под размеры окна и также будет иметь публичные свойства для изменения цвета элементов, таких как шарик, игровое поле, платформа и кирпичики.

2.4 Описание входных и выходных данных компонента

В подразделе 2.2 была получена структурная схема компонента. Для удобства описания всех свойств и событий класса сведены в табличный вид. Свойства класса представлены в таблице 2.1.

Таблица 2.1 – Свойства класса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Тип данных | Тип доступа | Доступ на чтение | Доступ на запись |
| Score (счет) | Int | чтение | Вернет значение переменной score |  |
| MapColor (цвет фона) | Color | чтение/запись | Вернет значение переменной \_mapColor | Специально разработанный механизм обновления значения переменной \_mapColor, служащий для перерисовки картинки на экране после изменения значения |
| BallColor (цвет шара) | Color | чтение/запись | Вернет значение переменной ballColor | Специально разработанный механизм обновления значения переменной \_ballColor, служащий для перерисовки картинки на экране после изменения значения |

Окончание таблицы 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя | Тип данных | Тип доступа | Доступ на чтение | Доступ на запись |
| PlatformColor (цвет платформы) | Color | чтение/запись | Вернет значение переменной \_platformColor | Специально разработанный механизм обновления значения переменной platformColor, служащий для перерисовки картинки на экране после изменения значения |
| BlockColor (цвет блока) | Color | чтение/запись | Вернет значение переменной \_blockColor | Специально разработанный механизм обновления значения переменной \_blockColor, служащий для перерисовки картинки на экране после изменения значения |
| MapRow (количество строк) | Int | чтение/запись | Вернет значение переменной \_mapRow | Специально разработанный механизм обновления значения переменной \_mapRow, служащий для перерисовки картинки на экране после изменения значения |
| MapCol (количество столбцов) | Int | чтение/запись | Вернет значение переменной \_mapCol | Специально разработанный механизм обновления значения переменной \_mapCol, служащий для перерисовки картинки на экране после изменения значения |

Описание событий, присутствующих в компоненте, приведено в таблице 2.2.

Таблица 2.2 - События класса

|  |  |  |  |
| --- | --- | --- | --- |
| Имя события | Тип данных | Тип доступа | Описание события |
| RecordScore | EventHandler | добавление / удаление | Событие происходящие при изменении счёта. Отвечает за изменение счёта на экране |

2.5 Разработка алгоритмов компонента

В подразделе 2.2 была получена структурная схема компонента. Все выявленные на структурной схеме класса методы были разработаны и в данном разделе будут детально рассмотрены алгоритмы, заложенные в их основу.

XToCol – метод преобразующий координаты X шарика в номер колонки массива. Входные параметры x и dir типа Int, выходной параметр - номер колонки. Блок-схема метода приведена на рисунке 2.5

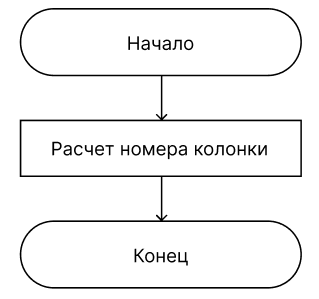


Рисунок 2.5 – Алгоритм метода XToCol

YToRow – метод преобразующий координаты Y шарика в номер столбца массива. Входные параметры y и dir типа Int, выходной параметр - номер столбца. Блок-схема метода приведена на рисунке 2.6

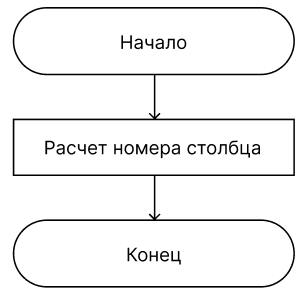


Рисунок 2.6 - Алгоритм метода YToRow

InitFunction – метод, инициализирующий массив игрового поля. Входные и выходные параметры отсутствуют. Блок-схема метода приведена на рисунке 2.7

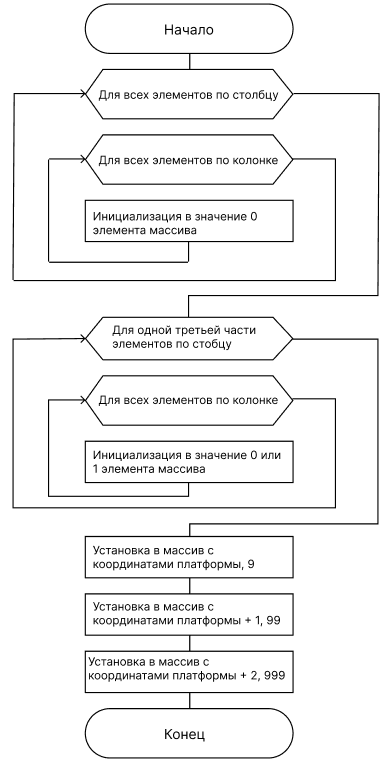


Рисунок 2.7 – Алгоритм метода InitFunction

InputCheck – метод, задающий платформе вектор-направление по оси Х в соответствии с нажатой клавишей. Входные параметры sender типа object, e типа KeyEventArgs, выходные параметры отсутствуют. Блок-схема метода приведена на рисунке 2.8

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.8 – Алгоритм метода InputCheck

StartGame – метод вызывается при запуске игры. Он запускает таймер и переводит фокус на компонент. Метод обновляет элементы в массиве и ставит их в начальное положение, также обнуляет счет. В конце обновляет экран с помощью метода Invalidate. Входные и выходные параметры отсутствуют. Блок-схема алгоритма метода StartGame приведена на рисунке 2.9

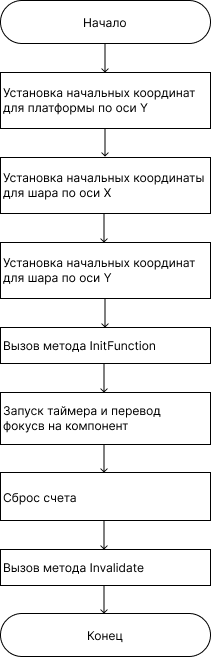


Рисунок 2.9 – Алгоритм метода StartGame

StopGame – метод вызывается для остановки игры. Он останавливает таймер и меняет статус игры, в конце обновляет экран. Входные и выходные параметры отсутствуют. Блок-схема алгоритма метода StopGame приведена на рисунке 2.10

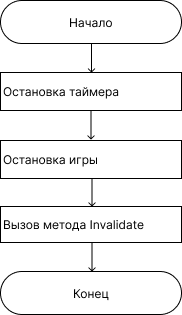


Рисунок 2.10 – Алгоритм метода StopGame

CheckGameStatus – метод проверяющий, остались ли блоки на карте, и не упал ли шар за границы платформы по оси Y. Входные и выходные параметры отсутствуют. Блок-схема алгоритма метода CheckGameStatus приведена на рисунке 2.11

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.11 – Алгоритм метода CheckGameStatus

Update – метод, отвечающий за выполнение основной логики за каждый тик таймера. Входные параметры source типа Object и e типа EventArgs, выходные параметры отсутствуют. Блок-схема алгоритма метода Update приведена на рисунке 2.12



Рисунок 2.12 – Алгоритм метода Update

IsCollide – метод, проверяющий коллизию, с которой сталкивается шар. Меняет траекторию шара при столкновении с границей карты, блоком или платформой. При попадании в блок инкрементирует счет, а также удаляет его блок. Входные и выходные параметры отсутствуют. Блок-схема алгоритма метода IsCollide приведена на рисунке 2.13

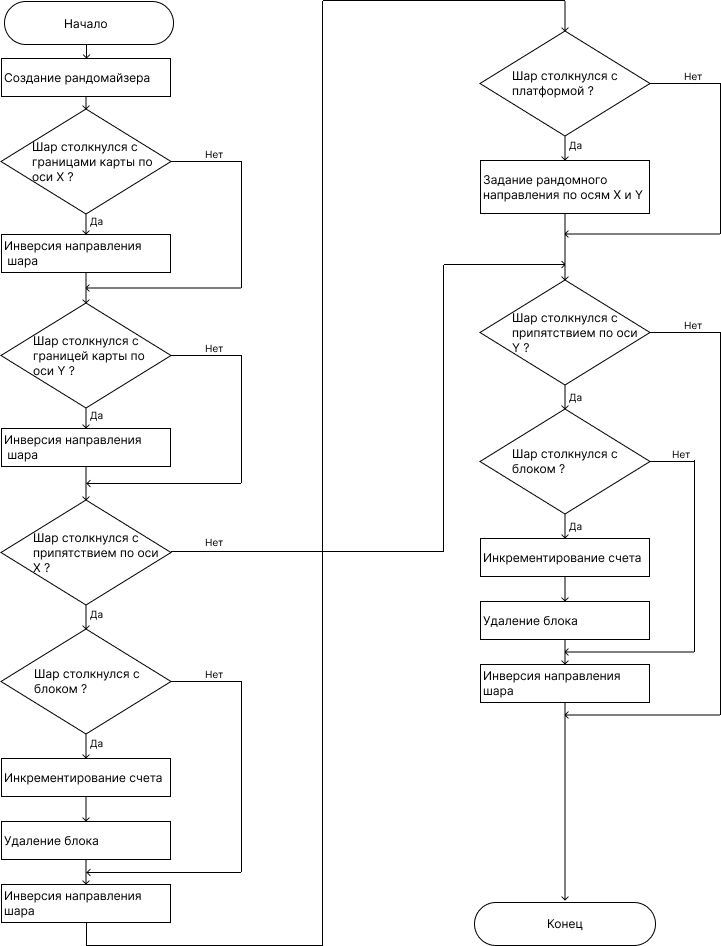


Рисунок 2.13 – Алгоритм метода IsCollide

SetBoundsCore – метод, отвечающий за регулирование границ компонента. Метод не позволяет задать ширину меньше 501 пикселя или высоту меньше 601 пикселя. Входные параметры x, y, width, height типа Int и specified типа BoundsSpecified, выходные параметры отсутствуют. Блок-схема алгоритма метода SetBoundsCore приведена на рисунке 2.14

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.14 – Алгоритм метода SetBoundsCore

OnPaint – метод, отвечающий за прорисовку карты, платформы, шара и блоков в компоненте. Входные параметры e типа PaintEventArgs, выходные параметры отсутствуют. Блок-схема алгоритма метода OnPaint приведена на рисунке 2.15

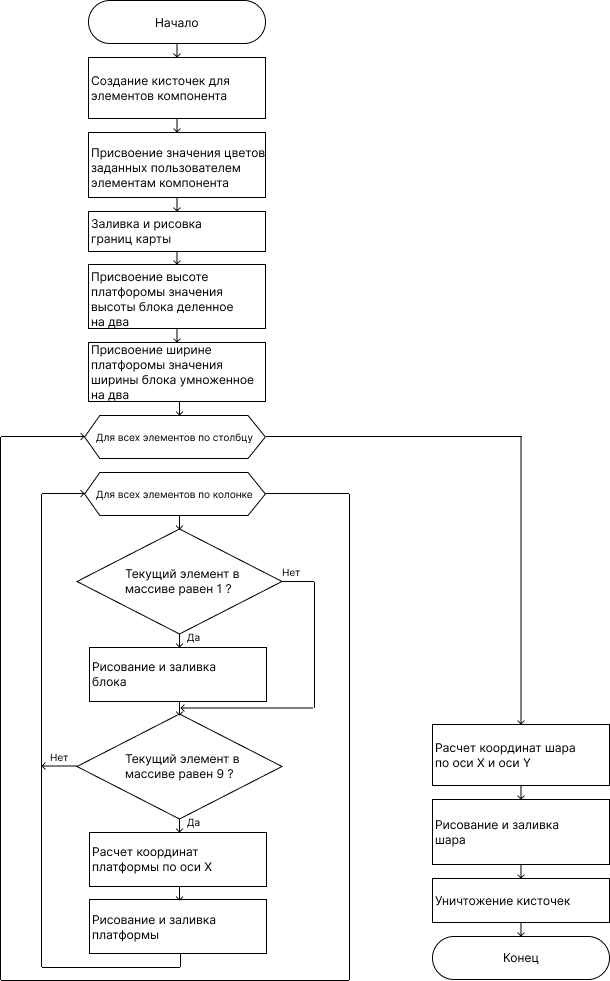


Рисунок 2.15 – Алгоритм метода OnPaint

Согласно разработанным алгоритмам всех методов класса, был написан программный код, представленный в Приложении А к пояснительной записке.

3 Технологический раздел

3.1 Отладка компонента

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки, информация из работы [7]. Программу без ошибок составить почти невозможно, а для достаточно сложной программы найти и решить ошибки будет трудно. Разумно уже при разработке программы на этапах алгоритмизации и программирования готовиться к обнаружению ошибок на стадии отладки принимать профилактические меры по их предупреждению, информация из работы[8].

Тестирование будет происходить через тест-кейсы. Тест-кейс – это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. При передаче тестировщику тест-кейсов, он должен пройтись по всем его пунктам и выполнить описанные действия, которые должны привести к определенным результатам. информация из работы [9]. Тест-кейс для свойств и событий представлен в таблице 3.1.

Таблица 3.1 – Тест-кейс для свойств и событий

|  |  |  |
| --- | --- | --- |
| Имя свойства | Управляющее воздействие | Результат воздействия |
| Score | Сбить блок в процессе игры | Инкрементируется счёт |
| MapColor | Изменить значение на выбранное | Цвет меняется на выбранный |
| BallColor | Изменить значение на выбранное | Цвет меняется на выбранный |
| MapRow | Изменить значение на выбранное | Количество строк меняется на выбранное |
| MapCol | Изменить значение на выбранное | Количество колонок меняется на выбранное |

Окончание таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Имя свойства | Управляющее воздействие | Результат воздействия |
| PlatformColor | Изменить значение на выбранное | Цвет меняется на выбранный |
| BlockColor | Изменить значение на выбранное | Цвет меняется на выбранный |
| RecordScore | Сбить блок в процессе игры | Вывод нового значения счета на экран |

Тест кейс для методов представлен в таблице 3.2.

Таблица 3.2 – Тест-кейс для методов

|  |  |  |
| --- | --- | --- |
| Имя метода | Управляющее воздействие | Результат воздействия |
| SetBoundsCore | Изменить размер компонента | Ширина и высота игровых элементов меняется в зависимости от изменения размеров компонента |
| StartGame | Программно вызвать метод | Размещение игровых элементов на свои места, при этом обнуление счета и запуск таймера, перевод фокуса на компонент |
| StopGame | Программно вызвать метод | Прекращение игры, выключение таймера, при этом изменение статуса игры |
| InitFuncion | Программно вызвать метод | Происходит случайная генерация блоков, платформа встает в начальное положение |
| XToCol | Программно вызвать метод | Преобразование пиксельной координаты шара по оси Х в элемент в массиве |
| YToRow | Программно вызвать метод | Преобразование пиксельной координаты шара по оси Y в элемент в массиве |
| CheckGameStatus | Програмнно вызвать метод | Завершение игры, если шар вылетит за границы платформы или произойдет сбитие всех блоков |

Окончание таблицы 3.2

|  |  |  |
| --- | --- | --- |
| Имя метода | Управляющее воздействие | Результат воздействия |
| Update | Запустить игру | Движение шара, вызов методов проверки с коллизией и проверки статуса игры |
| CheckGameStatus | Програмнно вызвать метод | Завершение игры, если шар вылетит за границы платформы или произойдет сбитие всех блоков |
| Update | Запустить игру | Движение шара, вызов методов проверки с коллизией и проверки статуса игры |
| IsCollide | Отбить шар в процессе игры | Изменение траектории шара при столкновении с коллизией, удаление блока |
| InputCheck | Нажать клавиши (a,d) на клавиатуре | Движение платформы |
| OnPaint | Изменить свойства цвета или свойства размеров компонента. Остановить и запустить игру | Изменение цвета у игровых элементов, прорисовка карты |

3.2 Инструкция по установке компонента

Для установки компонента в проект необходимо выполнить следующий набор действий. В обозревателе решений щелчком правой кнопки мыши по ветке “Ссылки” целевого проекта вызвать контекстное меню. В нем выбрать команду “Добавить ссылку”, как показано на рисунке 3.1

Изображение выглядит как текст, снимок экрана, монитор, черный

Автоматически созданное описание

Рисунок 3.1 – Вызов контекстного меню

В появившемся на экране окне «Менеджер ссылок» необходимо выбрать раздел «Обзор» и в нем нажать на кнопку «Обзор» в нижней правой части окна. Все описанные действия представлены на рисунке 3.2

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

Рисунок 3.2 – Менеджер ссылок

В появившемся окне «Менеджер ссылок» необходимо найти и выбрать файловый пакет с компонентом, как показано на рисунке 3.3

Изображение выглядит как текст, монитор, снимок экрана, черный

Автоматически созданное описание

Рисунок 3.3 – Выбор файлового пакета

После успешного завершения выбора файлового пакета необходимо в панели элементов создать новую вкладку и щелчком правой кнопки мыши по ней вызвать контекстное меню. В контекстном меню следует выбрать команду «Выбрать элементы», как показано на рисунке 3.4

Изображение выглядит как текст, снимок экрана, монитор, черный

Автоматически созданное описание

Рисунок 3.4 – Выбор элемента

В результате на экране появиться окно «Выбор элементов панели элементов», в нем необходимо нажать кнопку «Обзор» в правом нижнем углу окна, как показано на рисунке 3.5

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.5 - Выбор элементов панели элементов

В появившемся окне проводника необходимо выбрать файловый пакет с компонентом как показано на рисунке 3.6

Изображение выглядит как текст, монитор, снимок экрана, внутренний

Автоматически созданное описание

Рисунок 3.6 – Выбор файлового пакета

После успешного выполнения всех действий в панели элементов появился компонент, чтобы разместить его на форме, необходимо в панели элементов найти и выбрать компонент, после переместить его на форму, как показано на рисунке 3.7

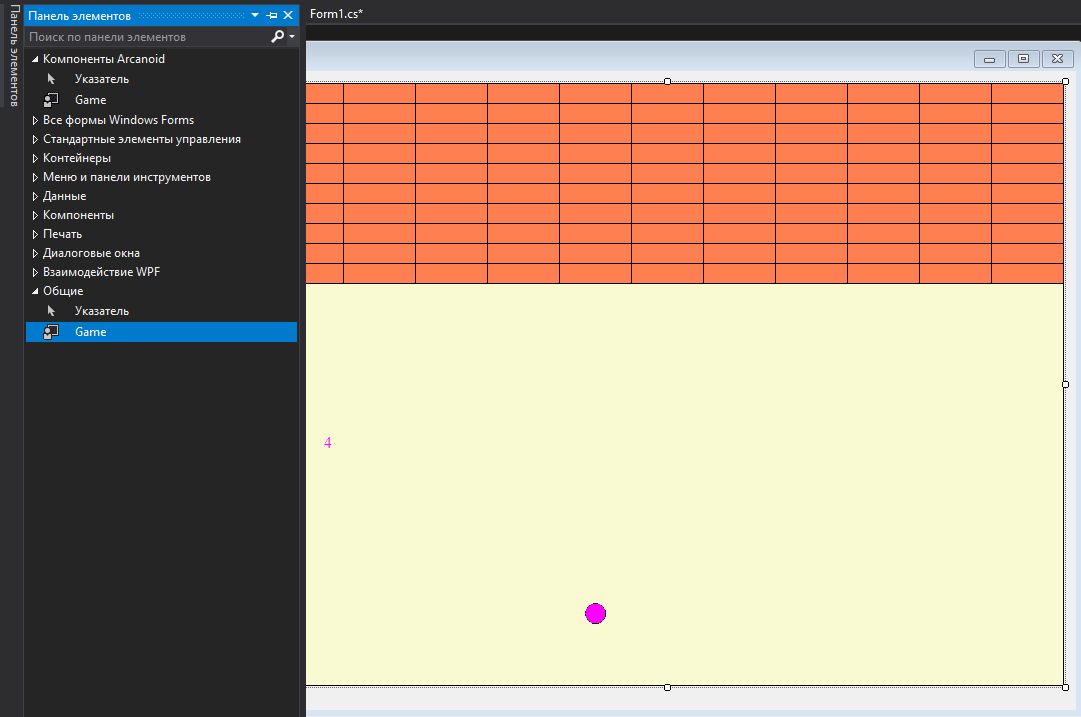


Рисунок 3.7 – Выбор и перемещение компонента на форму

3.3 Инструкция по эксплуатации компонента

Для удобства вся инструкция по эксплуатации визуального компонента сведена в табличный вид. Все свойства компонента перечислены в таблице 3.1.

Таблица 3.1 – Свойства компонента

|  |  |  |  |
| --- | --- | --- | --- |
| Имя свойства | Тип данных | Тип доступа | Назначение |
| Score (Счёт игры) | Int | чтение | Свойство необходимо для отображения текущего счёта |
| MapColor (Цвет карты) | Color | чтение/запись | Свойство необходимо для задания цвета карты |
| BallColor (Цвет шарика) | Color | чтение/запись | Свойство необходимо для задания цвета шарика |
| PlatformColor (Цвет платформы) | Color | чтение/запись | Свойство необходимо для задания цвета платформы |
| BlockColor (Цвет блоков) | Color | чтение/запись | Свойство необходимо для задания цвета блоков |
| MapRow (Количество строк) | Int | чтение/запись | Свойство необходимо для задания количества строк в массиве |
| MapCol (Количество колонок) | Int | чтение/запись | Свойство необходимо для задания количества колонок в массиве |

События компонента представлены в таблице 3.2.

Таблица 3.2 – События компонента

|  |  |
| --- | --- |
| Имя события | Назначение |
| RecordScore | Событие срабатывает при изменении счёта |

Публичные методы компонента представлены в таблице 3.3.

Таблица 3.3 – Публичные методы компонента

|  |  |  |
| --- | --- | --- |
| Имя события | Входные и выходные параметры | Назначение |
| EndProcess | Нет входных и выходных параметров | Метод служит для принудительного завершения игры |
| StartProcess | Нет входных и выходных параметров | Метод служит для старта игры |

4 Раздел охраны труда

Охрана труда – это целая система законодательных и нормативно-правовых актов, технических, гигиенических, лечебно-профилактических мероприятий и средств, которые обеспечивают безопасность, сохранение здоровья и работоспособности человека в процессе труда. В наши дни труд стал более интенсивным и требует огромных затрат умственной, эмоциональной и физической нагрузок.

На рабочем месте программист осуществляет трудовую деятельность и проводит большую часть рабочего времени. Правильная организация рабочего места программиста повышает производительность труда от 8 до 20%. Следуя рекомендациям ГОСТ 12.2.032-78, необходимо организовать рабочее место таким образом, чтобы взаимное расположение всех его элементов соответствовало физическим и психологическим требованиям. Главные элементы рабочего места программиста – это письменный стол и кресло. Рабочее место организуется в соответствии с ГОСТ 12.2.032-78, информация из работы [10].

Площадь рабочего места с компьютером с жидкокристаллическим или плазменным экраном должна быть не менее 4,5 кв. м, а расстояние между столами с мониторами (от тыла одного монитора до экрана другого) не менее 2 м. Монитор должен располагаться на расстоянии 50-70 см от глаз программиста. Параметры рабочего стола сотрудника: возможность регулировки высоты рабочего стола, или точная высота — 72,5 см, ширина — 80, 100, 120 или 140 см, глубина рабочего стола 80 или 100 см, высота и ширина пространства под столешницей (для ног) – не менее 50 см, глубина на уровне колен не менее 45 см, а на уровне вытянутых ног не менее 65 см.

Правильное освещение рабочего места – это очень важный момент в трудовой деятельности человека, влияющий на эффективность труда, при этом такой момент предупреждает травматизм и профессиональные заболевания. При недостаточном освещении приходится напрягать зрение, при этом ослабляется внимание и это приводит к наступлению преждевременной утомленности. Слишком яркое освещение тоже плохо, так как оно вызывает ослепление, раздражение и резь в глазах. При искусственном освещении, источниками света служат два вида ламп: лампы накаливания и люминесцентные.

Известно, что шум ухудшает условия труда и оказывает вредное воздействие на организм человека. Согласно ГОСТ 12.1.003-88 «Шум для помещений расчетчиков и программистов, уровни шума не должны превышать соответственно: 71, 61, 54, 49, 45, 42, 40, 38 дБ», информация из работы [11].

При работе компьютерной техники выделяется много тепла, что может привести к пожароопасной ситуации. Источниками зажигания так же могут служить приборы, применяемые для технического обслуживания, устройства электропитания, кондиционеры воздуха. Серьёзную опасность представляют различные электроизоляционные материалы, используемые для защиты от механических воздействий отдельных радиодеталей. В связи с этим, участки, на которых используется компьютерная техника, по пожарной опасности относятся к категории пожароопасных “В”. При пожаре люди должны покинуть помещение в течение минимального времени. В помещениях с компьютерной техникой, недопустимо применение воды и пены ввиду опасности повреждения или полного выхода из строя дорогостоящего электронного оборудования. Для тушения пожаров необходимо применять углекислотные и порошковые огнетушители, которые обладают высокой скоростью тушения, большим временем действия, возможностью тушения электроустановок, высокой эффективностью борьбы с огнем. Воду разрешено применять только во вспомогательных помещениях, информация из работы [12].

Заключение

По итогам работы был разработан компонент для игры в Арканоид. Компонент адаптивный и представляет из себя карту с задаваемым количеством колонок и столбцов, где блоки подстраиваются под константы массива, также присутствуют платформа и шар. Для карты, платформы, шара и блоков пользователь может задать цвета на свое усмотрение. В компоненте присутствует система счета. Счет инициализируется при каждом сбитом блоке. Если игрок достиг главной цели – сбил все блоки, то игра закончится

Главным достоинством сделанного компонента можно генерация блоков. В дальнейшем алгоритм можно доработать, добавив сложно-сбиваемые блоки, не сбиваемые блоки.

У компонента есть перспективы развития в плане визуализации, добавления анимации и звуковых эффектов, а также реализация системы уровней с разными препятствиями для шара.

Разработанный игровой класс можно использовать в качестве развлекательных целях, чтобы скоротать время или просто развлечь себя. Компонент можно использовать в учебных целях. Компонент может продемонстрировать разные методы рисования, объектно-ориентированное программирование, работу с потоками и таймером. Алгоритмы в компоненте можно использовать при разработке других компонентов.

Список источников

1. <https://studentopedia.ru/informatika/pervij-etap-razvitiya-yazikov-programmirovaniya-1940---1960-gg---mashinnie-kodi-i-assembler--.html>
2. <https://ru.wikipedia.org/wiki/История_языков_программирования>
3. <https://info-comp.ru/programmirovanie/739-install-visual-studio-2019-community.html>
4. <https://play.arkanoid.online/?lang=ru>
5. <https://studme.org/223367/informatika/osnovnye_komponenty_interfeysov>
6. <https://total.kz/ru/news/biznes/istoriya_graficheskih_interfeyso>
7. <https://studopedia.ru/22_29871_neobhodimost-otladki-programmnogo-produkta.html>
8. <https://infopedia.su/4x1ec5.html>
9. <https://sergeygavaga.gitbooks.io/kurs-lektsii-testirovanie-programnogo-obespecheni/content/lektsiya-4-ch3.html>
10. <https://www.retail.ru/rbc/pressreleases/tsentr-povysheniya-kvalifikatsii-lider-organizatsiya-rabochego-mesta-ofisnogo-rabotnika/>
11. <https://xn--d1aux.xn--p1ai/opisanie-rabochego-mesta-programmista-na-predpriyatii/>
12. <https://studopedia.ru/8_107307_osveshchenie-pomeshcheniy-vichislitelnih-tsentrov.html>

Приложение А

Программный код класса Arcanoid

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Arcanoid

{

public class Game : Control

{

static int \_mapRow = 30; //Константы для нашего массива

static int \_mapCol = 30;

protected int[,] \_map;//игровое поле = массив

protected Color \_mapColor;//цвета игровых элементов

protected Color \_ballColor;

protected Color \_platformColor;

protected Color \_blockColor;

protected int \_mapSizeW = 500;//ширина и высота карты

protected int \_mapSizeH = 601;

protected int \_mapSizeBlockW;//ширина и высота блоков

protected int \_mapSizeBlockH;

protected int \_platformX;//координаты платформы в массиве

protected int \_platformY;

protected int \_platformW;//размеры и координаты платформы

protected int \_platformH;

protected int \_rxplatform;

protected int \_ryplatform;

protected int \_cfX;//отклоенение шара по оси Х, генерируется в методе IsCollide()

protected int \_cfY;//отклоенение шара по оси Y, генерируется в методе IsCollide()

protected int \_ballX; //размеры и координаты шара

protected int \_ballY;

protected int \_dirX;

protected int \_dirY;

protected int \_radiusBall;

protected int \_gmStatus = 0;

protected event EventHandler \_recordScore;

public System.Timers.Timer \_timer; //таймер для обработки

protected int score = 0;

public event EventHandler RecordScore//событие

{

add { \_recordScore += value; }

remove { \_recordScore -= value; }

}

public int Score//счет, свойство

{

get { return score; }

private set

{

if (score != value)

{

score = value;

OnRecordScore();

}

}

}

public int GmStatus//статус игры, свойство

{

get { return \_gmStatus; }

private set

{

if (\_gmStatus != value)

{

\_gmStatus = value;

}

}

}

protected void OnRecordScore()//счет

{

\_recordScore?.Invoke(this, new EventArgs());

}

protected override void SetBoundsCore(int x, int y, int width, int height, BoundsSpecified specified)//адаптация, при изменени размера компонента

{

if (width < 501 || height < 601)

{

width = 501;

height = 601;

}

\_mapSizeBlockW = width / \_mapCol;

\_mapSizeBlockH = height / \_mapRow;

width = \_mapSizeBlockW \* \_mapCol;

\_radiusBall = \_mapSizeBlockH / 3;

\_ryplatform = \_mapSizeBlockH \* \_platformY;

Invalidate();

base.SetBoundsCore(x, y, width, height, specified);

}

public Game() : base()//конструктор

{

\_map = new int[\_mapRow, \_mapCol];

MapColor = Color.Black;

BallColor = Color.Red;

BlockColor = Color.DarkGreen;

PlatformColor = Color.Blue;

\_platformX = (\_mapCol - 2) / 2;

\_platformY = \_mapRow - 3;

\_mapSizeBlockW = \_mapSizeW / \_mapCol;

\_mapSizeBlockH = \_mapSizeH / \_mapRow;

\_ballY = ((\_mapRow - 4) \* \_mapSizeBlockH);

\_ballX = (\_mapCol / 2) \* \_mapSizeBlockW;

\_timer = new System.Timers.Timer(1);

\_timer.AutoReset = true;

\_timer.Elapsed += Update;

\_timer.Enabled = false;

this.KeyDown += new KeyEventHandler(InputCheck);

InitFunction();//метод для обработки массива

\_dirX = 1;

\_dirY = 1;

\_cfX = 1;

\_cfY = 1;

Score = 0;

}

public void StartGame()//начать игру

{

\_platformX = (\_mapCol - 2) / 2;

\_platformY = \_mapRow - 3;

\_ballY = ((\_mapRow - 4) \* \_mapSizeBlockH);

\_ballX = ((\_mapCol + 1) / 2) \* \_mapSizeBlockW;

InitFunction();

\_timer.Enabled = true;

this.Focus();

Score = 0;

Invalidate();

}

public void StopGame()//закончить игру

{

\_timer.Enabled = false;

\_gmStatus = 0;

Invalidate();

}

protected void InitFunction()//метод для обработки массива, сначала обнуляет все элементы, потом генерирует блоки

{

Random r = new Random();

for (int i = 0; i < \_mapRow; i++)

{

for (int j = 0; j < \_mapCol; j++)

{

\_map[i, j] = 0;

}

}

for (int i = 0; i < \_mapRow / 3; i++)

{

for (int j = 0; j < \_mapCol; j++)

{

\_map[i, j] = r.Next(0, 2);

}

}

\_map[\_platformY, \_platformX] = 9;

\_map[\_platformY, \_platformX + 1] = 99;

\_map[\_platformY, \_platformX + 2] = 999;

}

protected int XToCol(int x, int dir)//преобразование координаты в пикселях по оси X

{

return (x + dir \* \_radiusBall) / \_mapSizeBlockW;

}

protected int YToRow(int y, int dir)//преобразование координаты в пикселях по оси Y

{

return (y + dir \* \_radiusBall) / \_mapSizeBlockH;

}

protected void CheckGameStatus()//првоерка статуса игры, если блоки сбиты, то начинается новая игра

{

\_gmStatus = 0;

if (YToRow(\_ballY, \_dirY) + \_dirY > \_mapRow - 1)

{

StopGame();

}

for (int i = 0; i < \_mapRow / 3; i++)

{

for (int j = 0; j < \_mapCol; j++)

{

if (\_map[i, j] == 1)

{

\_gmStatus++;

}

}

}

if (\_gmStatus < 1)

{

StartGame(); //победа! все блоки сбиты, начинается следующий уровень

}

}

private void Update(object Source, EventArgs e) // обновление, используется в таймере

{

CheckGameStatus();

\_ballX += (int)(\_cfX \* \_dirX);// Менять коэф. движения шарика

\_ballY += (int)(\_cfY \* \_dirY);// Менять коэф. движения шарика

Thread thread = new Thread(() => //создание потока, для инкремента счета, без него все рушиться

{

Invoke((Action)(() =>

{

IsCollide();

}));

});

thread.Start();

Invalidate();

}

protected void IsCollide() // проверка коллизии, прибавление счета, удаление блока

{

Random r = new Random();

if (\_ballX + \_radiusBall > Width - 10 || \_ballX - \_radiusBall < 10)//работает исправно, бывает не успевает менять траекторию, из-за большого количества проверок

{ //из-за этого мяч может заедать в стене

\_dirX \*= -1;

}

if (\_ballY < \_radiusBall + 10)//работает исправно

{

\_dirY \*= -1;

}

if (\_map[YToRow(\_ballY, \_dirY), XToCol(\_ballX, 0)] != 0)//работает исправно

{

if (\_map[YToRow(\_ballY, \_dirY), XToCol(\_ballX, 0)] < 9)

{

Score++;

\_map[YToRow(\_ballY, \_dirY), XToCol(\_ballX, 0)] = 0;

}

\_dirY \*= -1;

if (\_ballY > (\_mapRow / 2) \* \_mapSizeBlockH)

{

\_cfY = r.Next(1, 4);

\_cfX = r.Next(1, 4);

}

}

if (\_map[YToRow(\_ballY, -1), XToCol(\_ballX, \_dirX)] != 0)//работает исправно

{

if (\_map[YToRow(\_ballY, -1), XToCol(\_ballX, \_dirX)] < 9)

{

Score++;

\_map[YToRow(\_ballY, -1), XToCol(\_ballX, \_dirX)] = 0;

}

\_dirX \*= -1;

}

}

private void InputCheck(object Source, KeyEventArgs e) // движение платформы, проверка нажатия на клавишы A - Left, D - Right

{

\_map[\_platformY, \_platformX] = 0;

\_map[\_platformY, \_platformX + 1] = 0;

\_map[\_platformY, \_platformX + 2] = 0;

switch (e.KeyCode.ToString())

{

case "D":

if (\_platformX + 1 < \_mapCol - 2)

\_platformX += 1;

break;

case "A":

if (\_platformX > 0)

\_platformX -= 1;

break;

}

\_map[\_platformY, \_platformX] = 9;

\_map[\_platformY, \_platformX + 1] = 99;

\_map[\_platformY, \_platformX + 2] = 999;

}

public Color MapColor // цвет фона

{

get

{

return \_mapColor;

}

set

{

if (\_mapColor != value)

{

\_mapColor = value;

Invalidate();

}

}

}

public Color BallColor //цвет шара

{

get

{

return \_ballColor;

}

set

{

if (\_ballColor != value)

{

\_ballColor = value;

Invalidate();

}

}

}

public Color PlatformColor //цвет платформы

{

get

{

return \_platformColor;

}

set

{

if (\_platformColor != value)

{

\_platformColor = value;

Invalidate();

}

}

}

public int MapRow //количество строк

{

get

{

if (\_mapRow < 10)

{

\_mapRow = 10;

}

else if (\_mapCol > \_mapRow)

{

\_mapCol = \_mapRow;

}

\_mapSizeBlockW = Width / \_mapCol;

\_mapSizeBlockH = Height / \_mapRow;

\_platformX = (\_mapCol - 2) / 2;

\_platformY = \_mapRow - 3;

\_ryplatform = \_mapSizeBlockH \* \_platformY;

\_ballY = ((\_mapRow - 4) \* \_mapSizeBlockH);

\_ballX = ((\_mapCol + 1) / 2) \* \_mapSizeBlockW;

\_radiusBall = \_mapSizeBlockH / 3;

InitFunction();

return \_mapRow;

}

set

{

if (\_mapRow != value)

{

\_mapRow = value;

\_mapSizeBlockW = Width / \_mapCol;

\_mapSizeBlockH = Height / \_mapRow;

\_platformX = (\_mapCol - 2) / 2;

\_platformY = \_mapRow - 3;

\_ryplatform = \_mapSizeBlockH \* \_platformY;

\_ballY = ((\_mapRow - 4) \* \_mapSizeBlockH);

\_ballX = ((\_mapCol + 1) / 2) \* \_mapSizeBlockW;

\_radiusBall = \_mapSizeBlockH / 3;

InitFunction();

Invalidate();

}

}

}

public int MapCol //количество колонок

{

get

{

if (\_mapCol < 5)

{

\_mapCol = 5;

}

else if (\_mapCol > \_mapRow)

{

\_mapCol = \_mapRow;

}

\_mapSizeBlockW = Width / \_mapCol;

\_mapSizeBlockH = Height / \_mapRow;

\_platformX = (\_mapCol - 2) / 2;

\_platformY = \_mapRow - 3;

\_ryplatform = \_mapSizeBlockH \* \_platformY;

\_ballY = ((\_mapRow - 4) \* \_mapSizeBlockH);

\_ballX = ((\_mapCol + 1) / 2) \* \_mapSizeBlockW;

\_radiusBall = \_mapSizeBlockH / 3;

InitFunction();

return \_mapCol;

}

set

{

if (\_mapCol != value)

{

\_mapCol = value;

\_mapSizeBlockW = Width / \_mapCol;

\_mapSizeBlockH = Height / \_mapRow;

\_platformX = (\_mapCol - 2) / 2;

\_platformY = \_mapRow - 3;

\_ryplatform = \_mapSizeBlockH \* \_platformY;

\_ballY = ((\_mapRow - 4) \* \_mapSizeBlockH);

\_ballX = ((\_mapCol + 1) / 2) \* \_mapSizeBlockW;

\_radiusBall = \_mapSizeBlockH / 3;

InitFunction();

Invalidate();

}

}

}

public Color BlockColor // цвет блоков

{

get

{

return \_blockColor;

}

set

{

if (\_blockColor != value)

{

\_blockColor = value;

Invalidate();

}

}

}

protected override void OnPaint(PaintEventArgs e)//рисование

{

Brush bMap = new SolidBrush(BackColor);

Brush bBlock = new SolidBrush(BackColor);

Brush bPlatform = new SolidBrush(BackColor);

Brush bBall = new SolidBrush(BackColor);

bMap = new SolidBrush(MapColor);

bBlock = new SolidBrush(BlockColor);

bPlatform = new SolidBrush(PlatformColor);

bBall = new SolidBrush(BallColor);

Pen p = new Pen(Color.Black, 1);

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

e.Graphics.DrawRectangle(p, 0, 0, Width - 1, Height - 1);

\_platformH = \_mapSizeBlockH / 2;

\_platformW = \_mapSizeBlockW \* 3;

for (int i = 0; i < \_mapRow; i++)//цикл по массиву

{

for (int j = 0; j < \_mapCol; j++)

{

if (\_map[i, j] == 1)//отрисовка блоков

{

e.Graphics.DrawRectangle(p, j \* \_mapSizeBlockW, i \* \_mapSizeBlockH, \_mapSizeBlockW, \_mapSizeBlockH);

e.Graphics.FillRectangle(bBlock, j \* \_mapSizeBlockW, i \* \_mapSizeBlockH, \_mapSizeBlockW, \_mapSizeBlockH);

e.Graphics.DrawRectangle(p, j \* \_mapSizeBlockW - 1, i \* \_mapSizeBlockH - 1, \_mapSizeBlockW, \_mapSizeBlockH);

}

if (\_map[i, j] == 9)//отрисовка платформы

{

\_rxplatform = j \* \_mapSizeBlockW;

e.Graphics.DrawRectangle(p, \_rxplatform - 1, \_ryplatform - 1, \_platformW, \_platformH);

e.Graphics.FillRectangle(bPlatform, \_rxplatform, \_ryplatform, \_platformW, \_platformH);

e.Graphics.DrawRectangle(p, \_rxplatform, \_ryplatform, \_platformW, \_platformH);

}

}

}

Rectangle \_rectangleForBall = new Rectangle(\_ballX - \_radiusBall, \_ballY - \_radiusBall, 2 \* \_radiusBall, 2 \* \_radiusBall);

e.Graphics.DrawEllipse(p, \_rectangleForBall);

e.Graphics.FillEllipse(bBall, \_rectangleForBall);

bBall.Dispose();

bMap.Dispose();

bBlock.Dispose();

bPlatform.Dispose();

}

protected override CreateParams CreateParams

{

get

{

CreateParams cp = base.CreateParams;

cp.ExStyle |= 0x02000000;

return cp;

}

}

}

}