**ИНСТИТУТ ТРАНСПОРТА И СВЯЗИ**



Факультет компьютерных наук и телекоммуникаций

Кафедра программного обеспечения компьютерных систем

**Курсовая работа**

по дисциплине

“ Технология конструирования программного обеспечения”

Тема: «Разработка компьютерной игры Pong»

Выполнили: Алексей Ануфриев, Денис Пурвиньш

Группа: 4502BD

Проверила: К.В. Костькина

Mg.sc.comp

Рига

2018

# Содержание

[Содержание 2](#_Toc534499508)

[Введение 3](#_Toc534499509)

[1. Задание на курсовую работу 4](#_Toc534499510)

[2. Этап Начало 5](#_Toc534499511)

[2.1. Спецификация требований заказчика 5](#_Toc534499512)

[2.2. Идентификация актёров 5](#_Toc534499513)

[2.3. Идентификация и краткое описание элементов Use Case 6](#_Toc534499514)

[2.4. Системные требования 7](#_Toc534499515)

[2.5. Прототип интерфейса приложения: 7](#_Toc534499516)

[3. Этап Развитие 8](#_Toc534499517)

[3.1. Конечная Use Case диаграмма 9](#_Toc534499518)

[3.2. Сценарии Use Case элементов 10](#_Toc534499519)

[3.3. Создание классов 15](#_Toc534499520)

[4. Этап Конструирование 17](#_Toc534499521)

[5. Тестирование 28](#_Toc534499522)

[6. Пример работы программы 30](#_Toc534499523)

[Заключение 31](#_Toc534499524)

[Список используемой литературы 32](#_Toc534499525)

[Приложение 33](#_Toc534499526)

[Файл ViewModel.cs 33](#_Toc534499527)

[Файл MainWindow.xaml 35](#_Toc534499528)

[Файл Game.cs 37](#_Toc534499529)

[Файл GameEngine.cs 40](#_Toc534499530)

[Файл NeuralNetwork.cs 42](#_Toc534499531)

# Введение

Данная курсовая работа написана в рамках дисциплины «Технологии конструирования программного обеспечения» и посвящена разработке десктопного приложения - компьютерной игры Pong для операционной системы Microsoft Windows. В работе подробно описаны все этапы разработки.

Выбрана методология разработки Rational Unified Process, использующая итеративную модель разработки. Программа написана на языке программировании С# с использованием среды разработки Visual Studio 2017. В проекте используется технология WPF и паттерн MVVM.

В качестве системы контроля версий был использован Git.

Репозиторий (сервис GitHub): <https://github.com/Ary0n-Z/SmartPong>

## Задание на курсовую работу

1. Разработать визуальную модель системы (бизнес-модель, модель требований, модель анализа, проектную модель, компонентную модель, модель размещения).
2. Выбрать язык и среду программирования и создать исполняемый код системы.
3. Графический интерфейс пользователя оформить с помощью графической библиотеки среды программирования.
4. Оценить качество визуальной модели и исполняемого кода системы. Для оценки модели использовать следующий набор метрик: Чидамбера-Кемерера; Лоренца – Кидда;
5. Технология процесса разработки: унифицированный инкрементный процесс. Количество итераций - не менее трёх.
6. Предоставляемые материалы:
   * Пояснительная записка (бумажная и электронная версия);
   * Файл визуальной модели;
   * Файл исполняемой системы.

## Этап Начало

### Спецификация требований заказчика

Разработать программу для операционной системы Windows с использованием технологии WPF: компьютерная игра Pong. Пользователи системы: Люди, изучающие Нейронные Сети, игроки.

Pong является простым спортивным симулятором настольного тенниса на 1го игрока. В роли противника выступает искусственная нейронная сеть, которая может обучаться в ходе игры. Небольшой квадратик, заменяющий пинг-понговый мячик, двигается по экрану по линейной траектории. Если он ударяется о периметр игрового поля, то его траектория изменяется в зависимости от угла столкновения. Если шарик отбивается ракеткой игрока, то его движение дополнительно зависит от скорости и направления движения ракетки. Управление ракетками в Pong осуществляется с помощью клавиш стрелочек верх и низ. Периметр игрового поля обозначен краями экрана, а мячик не может покинуть поле через верхний или нижний край. В верхней части поля отображаются очки игроков, у каждого на своей половине экрана.

Игровой процесс состоит в том, что игроки передвигают свои ракетки вертикально для защиты своих ворот. В начале каждого раунда мячик подаётся одному из игроков, и раунд продолжается до тех пор, пока один из игроков не заработает очко. Это происходит тогда, когда его оппонент не может отбить мячик. Со временем игры скорость движения мячика постепенно увеличивается, и так игра усложняется.

Особенностью игры является то, что ракетки не могут дойти до самого верха экрана и отбить мячик, если он туда попадает — в этом случае оппоненту засчитывается очко.

### Идентификация актёров

Актёры (Пользователи системы)

* Gamer: Человек, играющий в видеоигры
* Neural Network: Математическая модель, выступающая в роли игрока
* Game Engine: Объект, обеспечивающий отрисовку игровых компонентов и перемещение шарика

Области видимости

* Game: Бизнес функции
* GUI: Функции пользовательского интерфейса

### Идентификация и краткое описание элементов Use Case

Функциональные требования на этапе 'Начало' включают в себя:

Запуск (Start Game) - Начало новой игры

Пауза (Pause Game) - Временная остановка игры

Продолжение (Continue Game) - Восстановление временно остановленной игры

Завершение (End Game) - Конец запущенной игры

Запуск обучения (Train Neural Network) - Старт обучения сети ~ изменение весов связей

Остановка обучения (Stop Train Neural Network) - Остановка обучения сети ~ фиксация весов связей

Перемещение ракетки (Move Paddle) - Изменение местоположения на игровом поле (Изменение параметров X, Y)

Расчёт физики (Calculate Physics) - Распознание столкновения с объектом (ракеткой / игровым полем / воротами) и вычисление нового вектора перемещения

Отрисовка кадра (Render Image) - Создание и обновление игрового поля / ракеток / шарика / индикатора счёта / кнопок интерфейса

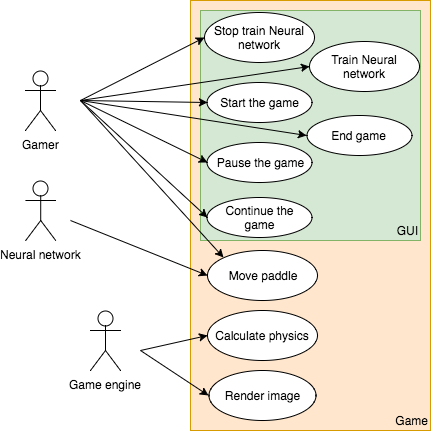


Рис. 2.1. Начальная диаграмма Use Case

### Системные требования

|  |  |  |
| --- | --- | --- |
| Системные требования | Минимальные | Рекомендуемые |
| Процессор (CPU) | 1.4 GHz | 2.0 GHz |
| Оперативная память (RAM) | 1 Gb | 2 Gb |
| Свободное место на диске | 100 Mb | 100 Mb |
| Устройства взаимодействия с пользователем | Клавиатура | Клавиатура и мышь |
| Операционная система | Microsoft Windows 7/10 - 32/64 bit | Microsoft Windows 7/10 - 32/64 bit |
| Программное обеспечение | .NET Framework 4.0 | .NET Framework 4.0 |

Таб. 2.1. Начальные системные требования

### Прототип интерфейса приложения:

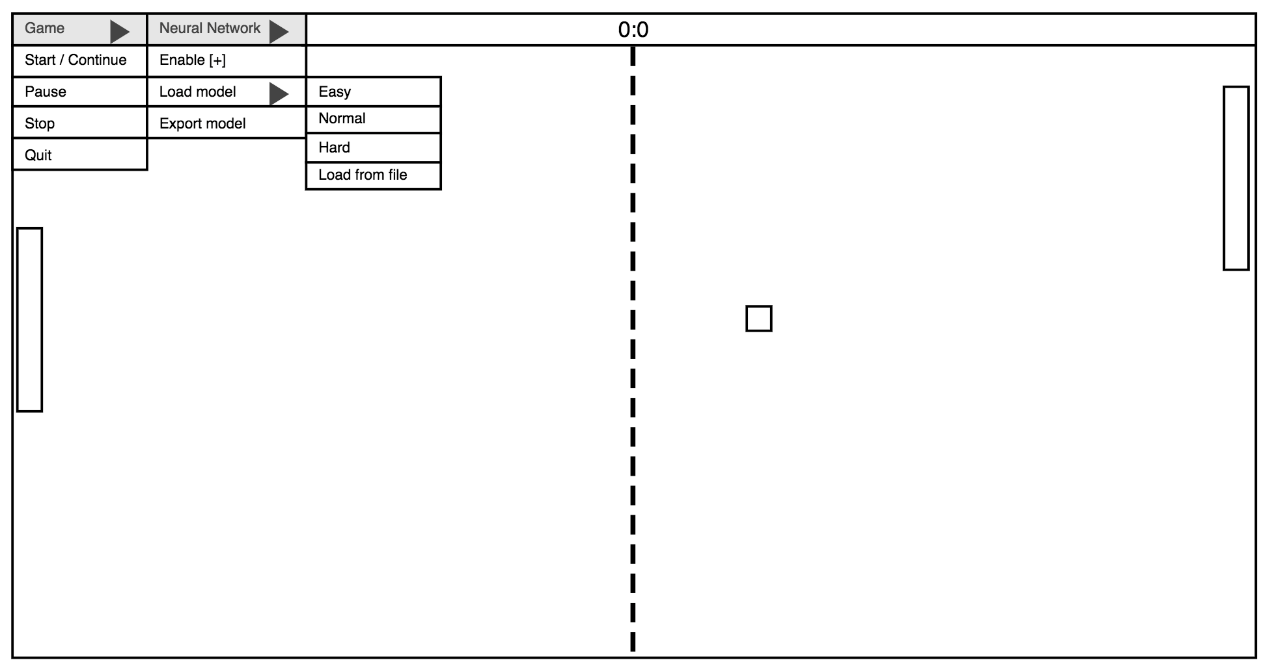


Рис. 2.2. Прототип интерфейса приложения

## Этап Развитие

На данном этапе дополняется и корректируется начальная Use Case диаграмма, создаются сценарии Use Case элементов, диаграммы последовательности и классов. Происходит начальная оценка качества.

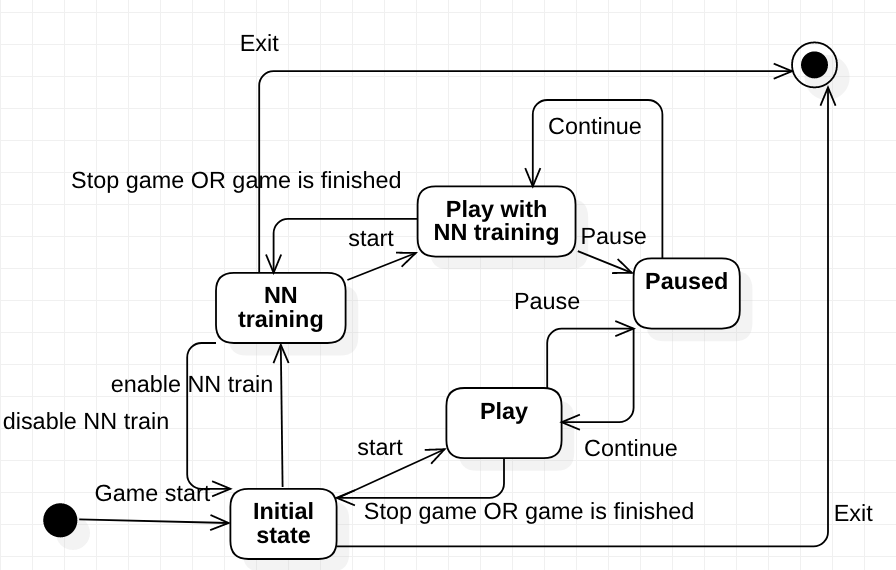


Рис. 3.0.1. Начальная версия диаграммы состояний

### Конечная Use Case диаграмма

* Chalk up (Ведение счёта): Игра должна останавливать и перезапускать игру при окончании раунда.

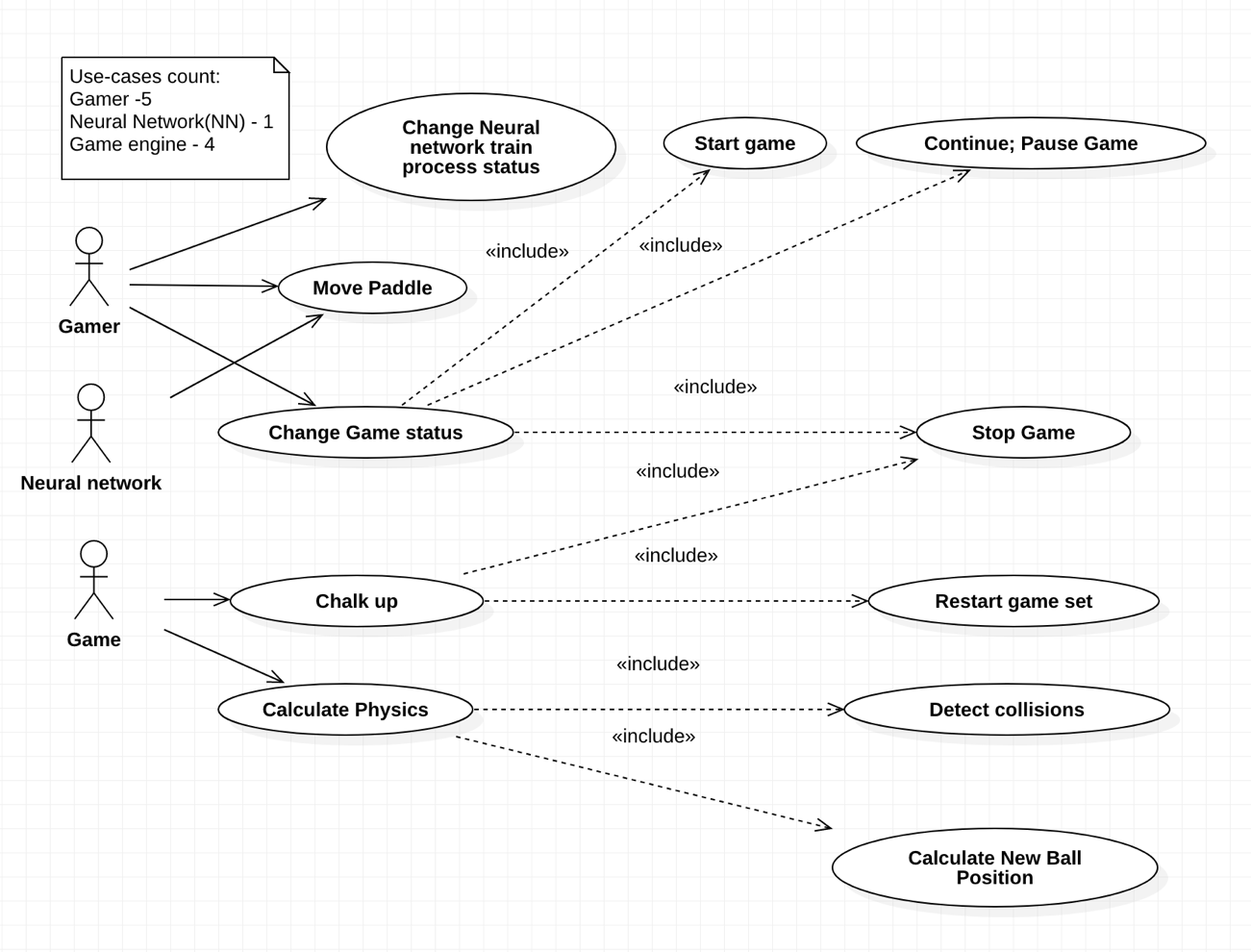


Рис. 3.1. Модифицированная диаграмма Use Case

### Сценарии Use Case элементов

* Перемещение ракетки (Move Paddle)

Актёр Gamer (Игрок) нажимает кнопку на клавиатуре (Up/Down). После перехвата события через MVVM Framework новые координаты рассчитываются в Game Engine и устанавливаются в игре (Game).

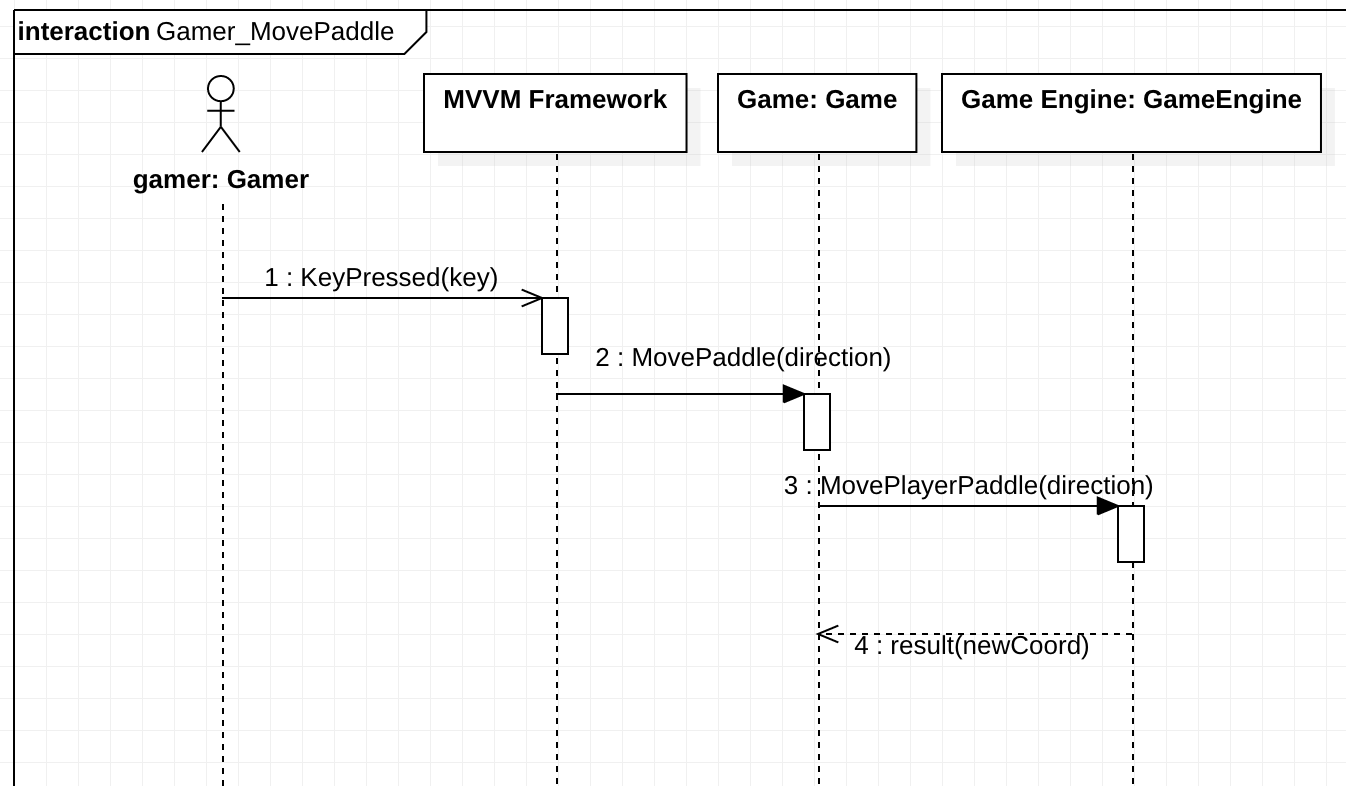


Рис. 3.2. Диаграмма последовательности для MovePaddle (Gamer)

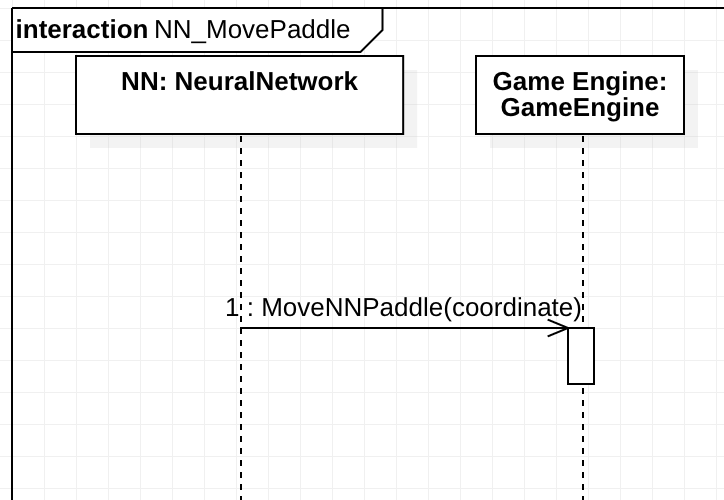


Рис. 3.3. Диаграмма последовательности для MovePaddle (NeuralNetwork)

* Изменение состояния обучения Neural Network (Change Neural network train process status)

Актёр Gamer (Игрок) нажимает кнопку, размещённую на пользовательском интерфейсе (Enable/Disable NN). После перехвата события через MVVM Framework изменяется статус нейронной сети в игре (Game).

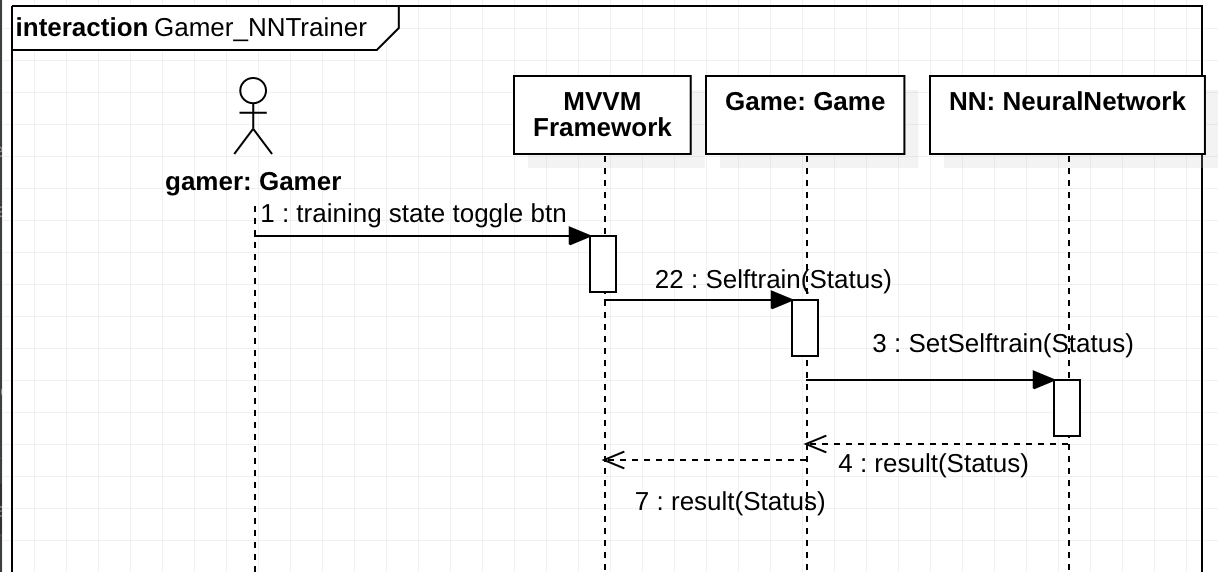


Рис. 3.4. Диаграмма последовательности для Change Neural network train process status

* Продолжение/Пауза (Continue; Pause game)

Актёр Gamer (Игрок) нажимает кнопку, размещённую на пользовательском интерфейсе (Pause/Continue Game). После перехвата события через MVVM Framework изменяется статус игры (Game).

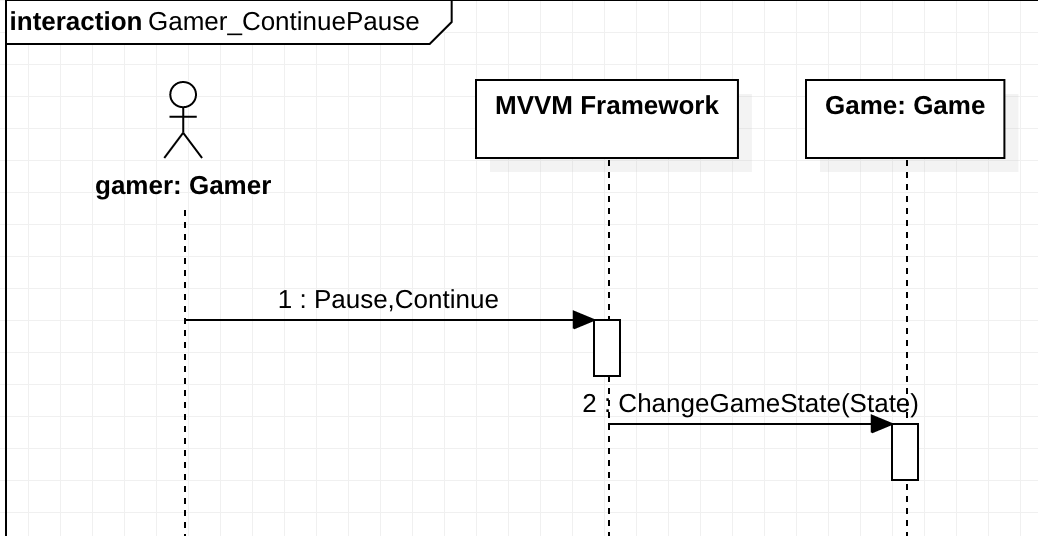


Рис. 3.5. Диаграмма последовательности для Continue; Pause Game

* Расчёт физики (Chalk up)

По событию таймера рассчитывается новое положение шарика, и проверяется его присутствие в зонах ракеток и ворот. Рассчитывается новый угол перемещения.

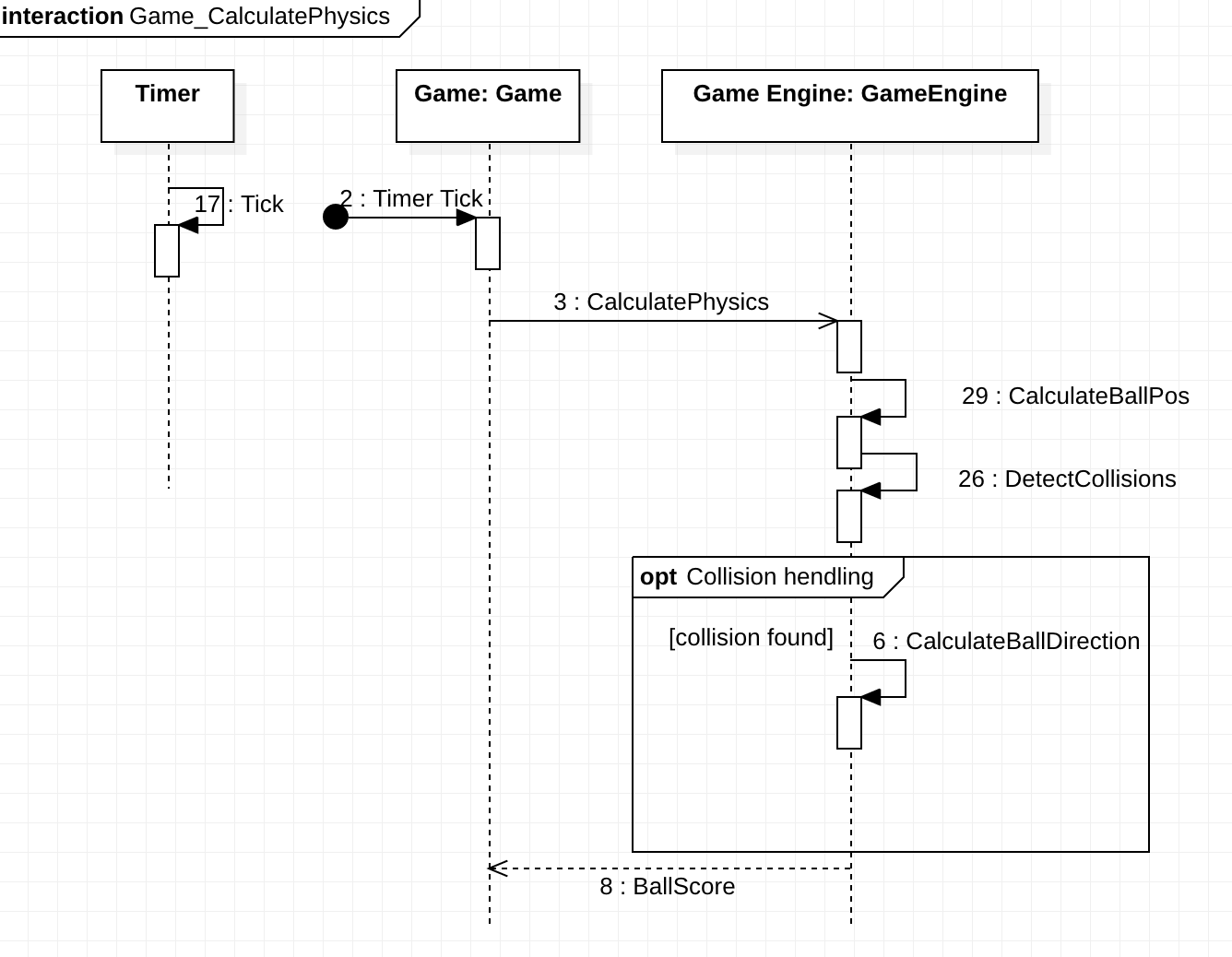


Рис. 3.6. Диаграмма последовательности для Calculate Physics

* Ведение счёта (Chalk up)

По событию BallScored засчитывается очко. Если их количество равно установленному числу (Victory condition) – игра останавливается. Иначе запускается новый матч.

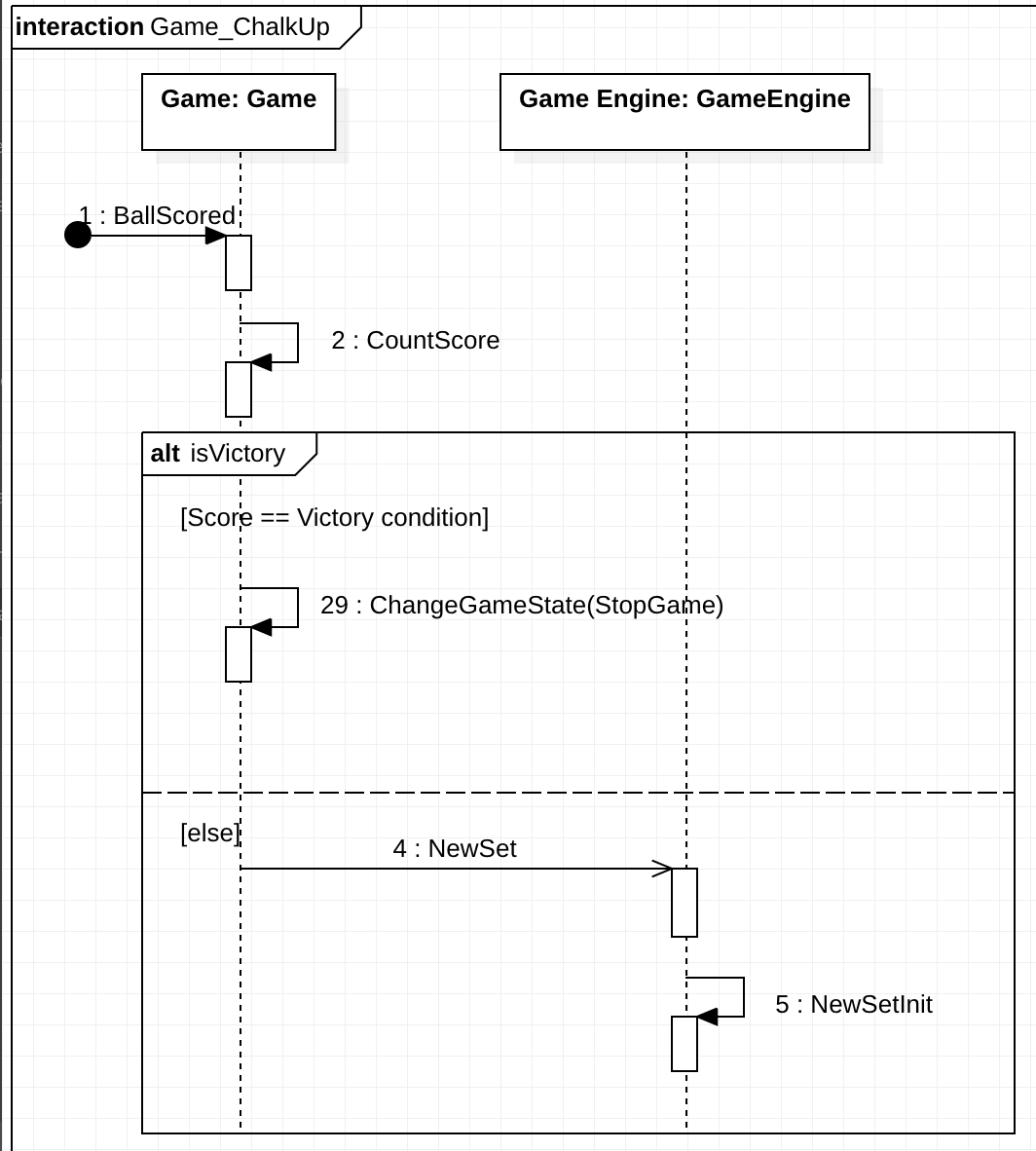


Рис. 3.7. Диаграмма последовательности для Chalk up

* Запуск игры (Start game)

Актёр Gamer (Игрок) нажимает кнопку, размещённую на пользовательском интерфейсе (Start Game). После перехвата события через MVVM Framework изменяется статус игры (Game).

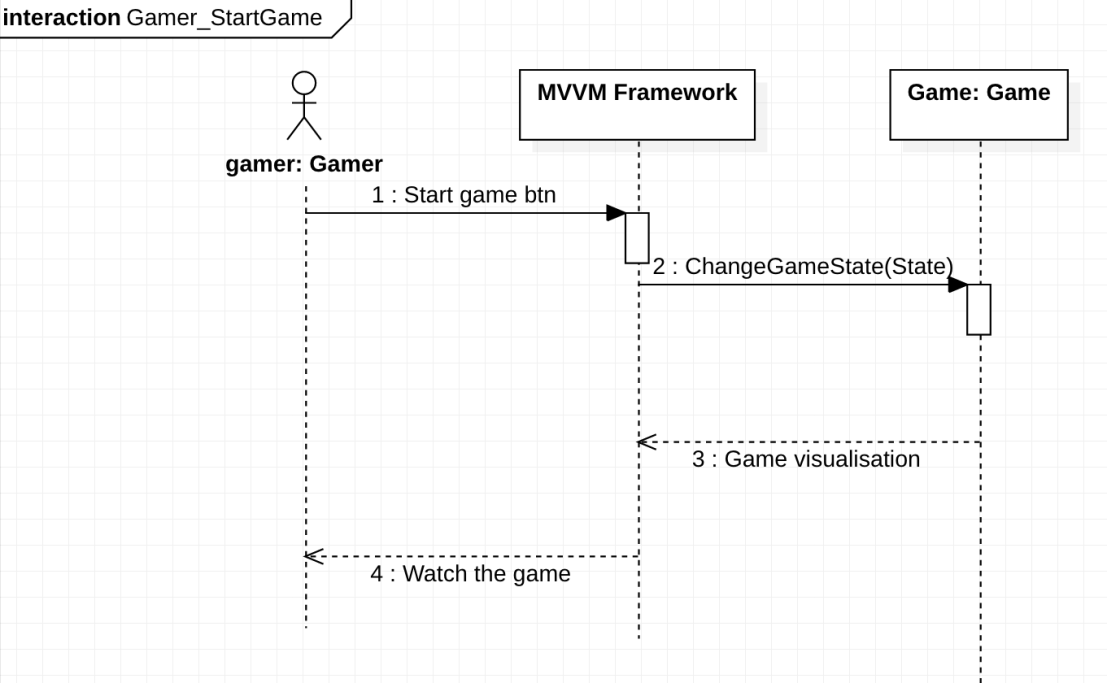


Рис. 3.8. Диаграмма последовательности для Start Game

* Остановка (Stop game)

Актёр Gamer (Игрок) нажимает кнопку, размещённую на пользовательском интерфейсе (Stop Game). После перехвата события через MVVM Framework изменяется статус игры (Game).

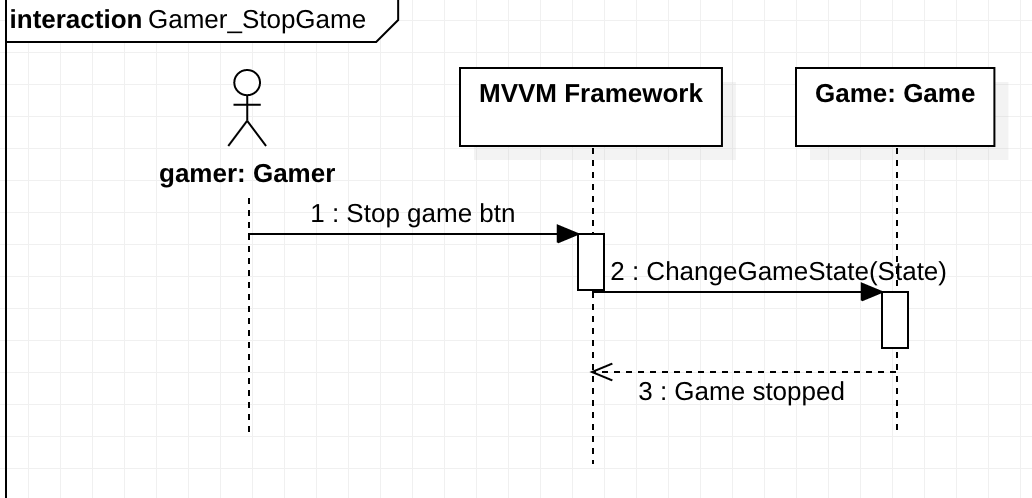


Рис. 3.9. Диаграмма последовательности для Stop Game

### Создание классов

Описание классов

* Game: Основной класс приложения – отвечает за игровой процесс;
* GameAttributes: инкапсулирует игровые объекты;
* GameEngine: содержит методы для вычисления физики шарика и обнаружения столкновений;
* NeuralNetwork: нейронная сеть, выступающая в роли противника;
* Ball: квадрат, содержащий свойства “Позиция”, “Угол полёта” и “Размер”;
* ViewModel: используется для привязывания элементов модели к WPF элементам.

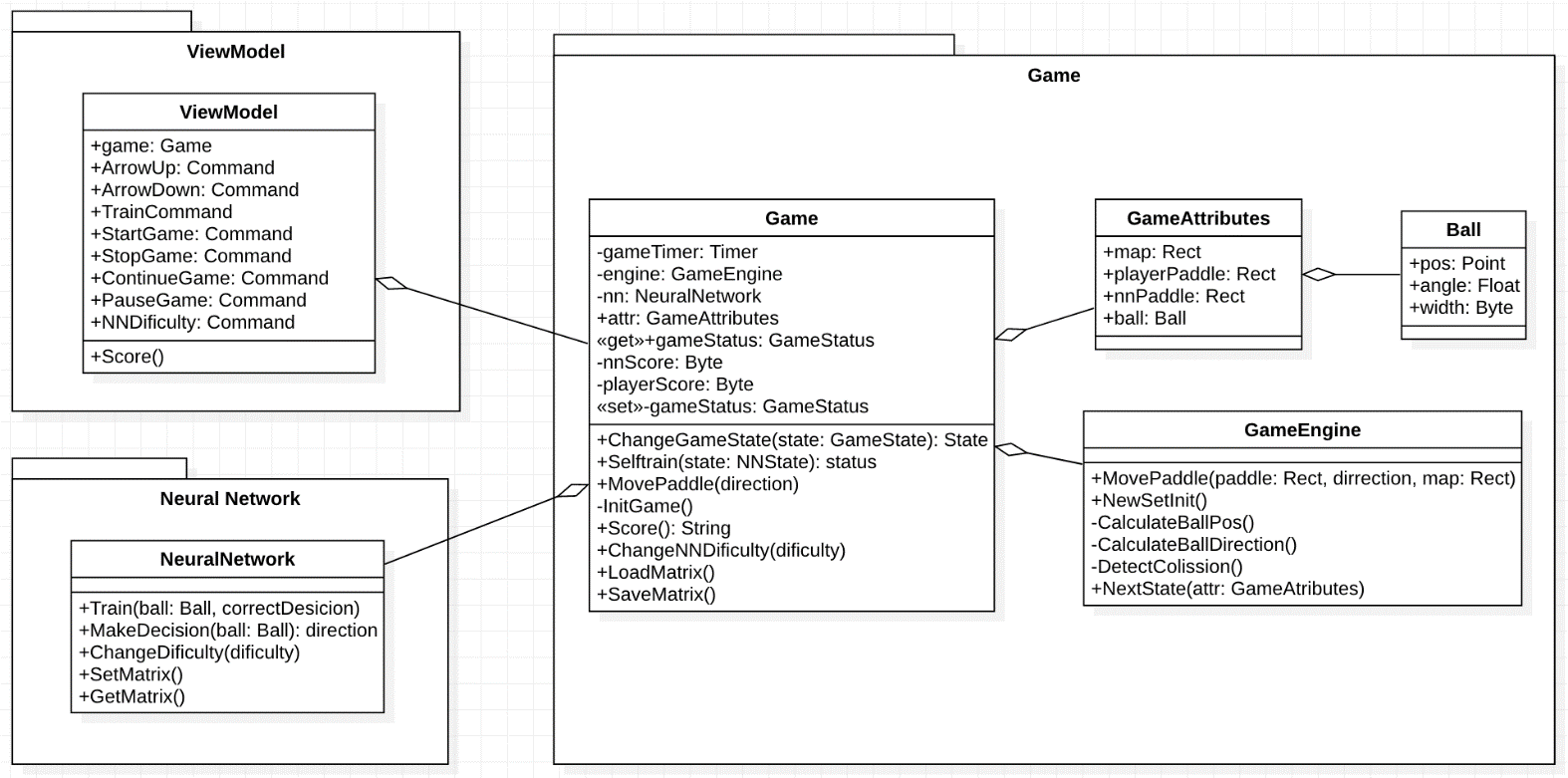


Рис. 3.10. Начальная диаграмма классов

Описание начальных метрик

WMC: взвешенные методы на класс

NOC: количество детей

DIT: высота дерева наследования

NC: общее количество классов в системе

NOM: суммарное количество методов в системе

Результаты начальной оценки качества

|  |  |  |
| --- | --- | --- |
|  | WMC | NOC |
| ViewModel | 1 | 0 |
| Game | 6 | 0 |
| NeuralNetwork | 3 | 0 |
| GameEngine | 6 | 0 |
| GameAttributes | 0 | 0 |
| Ball | 0 | 0 |
| Average | *2,67* | *0* |

Таб. 3.1. Метрики, вычисляемые для класса

|  |  |  |
| --- | --- | --- |
| DIT | NC | NOM |
| 0 | 6 | 16 |

Таб. 3.2. Метрики, вычисляемые для системы

Планирование итераций конструирования

Итерация 1 – Инициализация проекта. Подготовка архитектуры MVVM.

Итерация 2 – Реализация перемещения ракеток и шарика.

Итерация 3 – Создание игрового меню и эл. интерфейса.

Итерация 4 – Исправление ошибок и рефакторинг.

## Этап Конструирование

**Описание метрик**

NOF: Количество полей

NOM: Количество методов

NOP: Количество свойств

NOPF: Количество публичных полей

NOPM: Количество публичных методов

LOC: Количество строк кода

WMC: Взвешенные методы на класс

NC: Количество классов-наследников

DIT: Глубина дерева наследования

LCOM: Недостаток связности в методах LСOM

FAN-IN: Количество внешних классов, ссылающихся на текущий

FAN-OUT: Количество ссылаемых классов (зависимостей)

Итерация 1

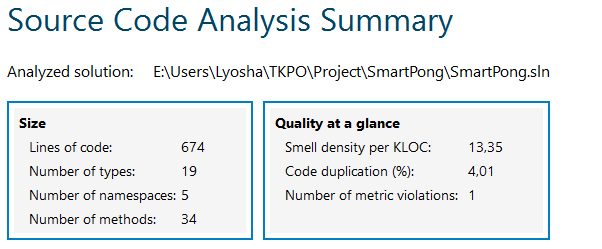


Рис. 4.11. Итерация 1 (Метрики 1)

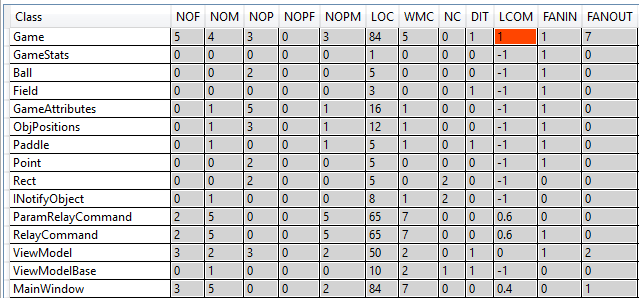


Рис. 4.12. Итерация 1 (Метрики 2)

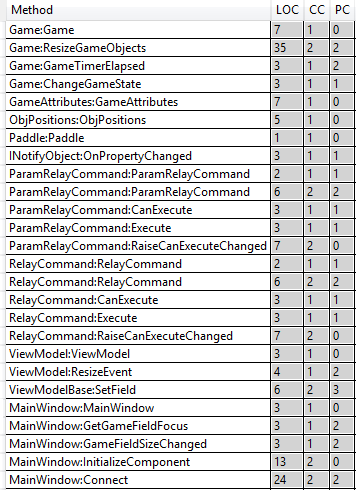


Рис. 4.13. Итерация 1 (Метрики 3)

Итерация 2

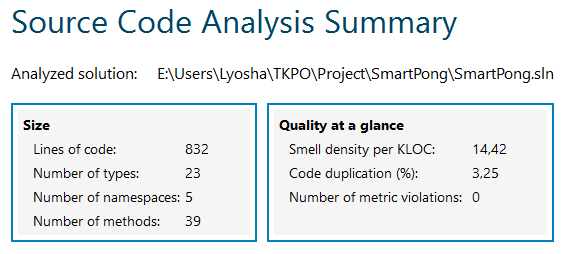


Рис. 4.14. Итерация 2 (Метрики 1)

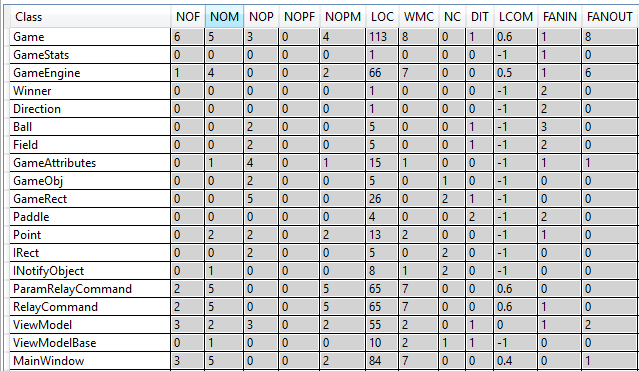


Рис. 4.15. Итерация 2 (Метрики 2)

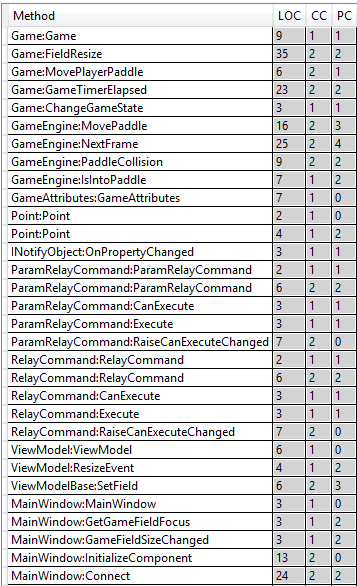


Рис. 4.16. Итерация 2 (Метрики 3)

Итерация 3

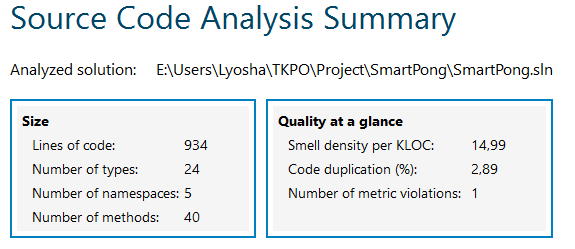


Рис. 4.17. Итерация 3 (Метрики 1)

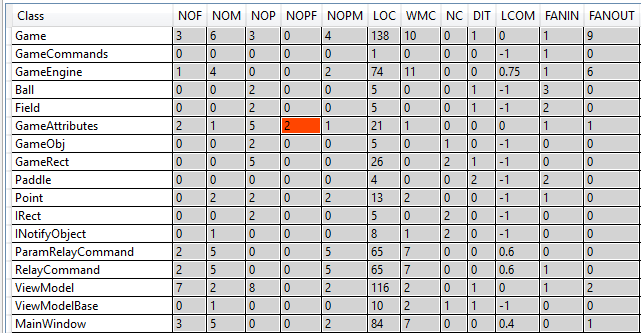


Рис. 4.18. Итерация 3 (Метрики 2)

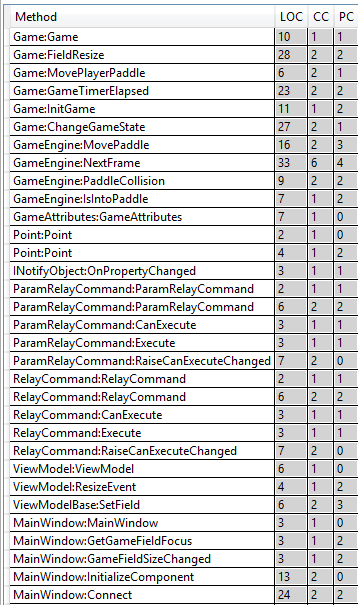


Рис. 4.19. Итерация 3 (Метрики 3)

**Итерация 4**

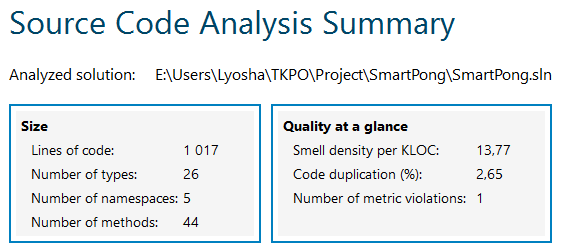


Рис. 4.20. Итерация 4 (Метрики 1)

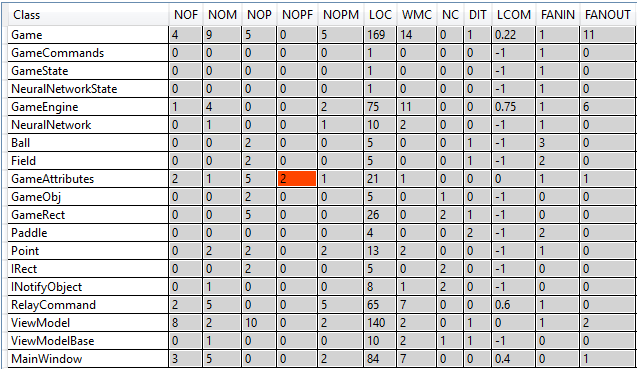


Рис. 4.21. Итерация 4 (Метрики 2)

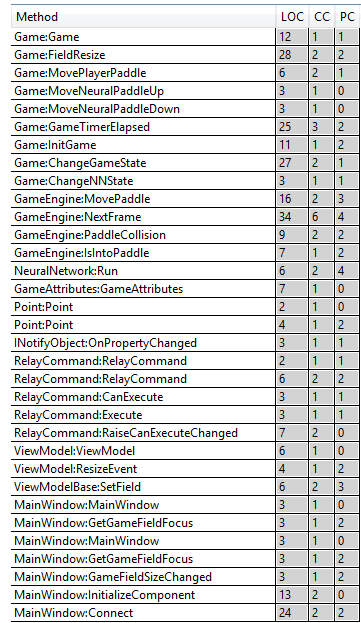


Рис. 4.22 Итерация 4 (Метрики 3)

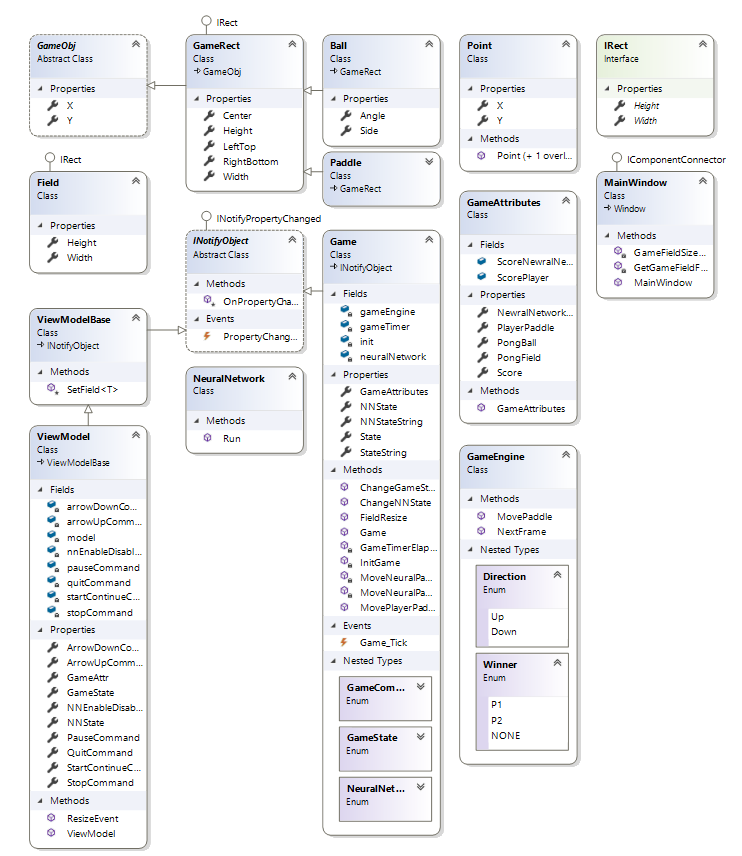


Рис. 4.23 Финальная диаграмма классов

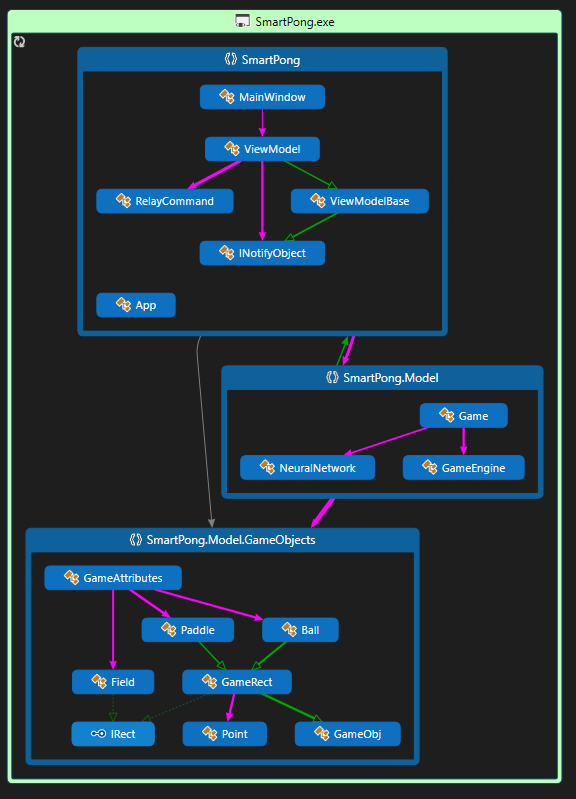


Рис. 4.24 Финальная диаграмма зависимостей

## Тестирование

Во время написания кода были разработаны юнит-тесты для функций, определяющих взаимодействие игровых объектов и вычисляющих их новые координаты. При обнаружении ошибки функции исправлялись. Тестирование было проведено по принципу «черного ящика» (рис.5.1). Дальнейшее покрытие тестами осуществлялось согласно функциональным требованьям заказчика, через игровой интерфейс приложения. Были составлены тесты и протестированы сценарии использования (табл. 4.3.1 и 4.3.2) и устранены все аномалии.

|  |  |
| --- | --- |
| Действия татуировщика | Ожидаемый результат |
| Запуск игры {Game->Start/Continue} | Игра началась, мяч начал своё движение в направлении игрока |
| Запуск-приостановка-продолжение{ Start/Continue ->Pause-> Start/Continue } | Остановка и возобновление движения всех игровых объектов, а так же отсутствие реакции на управление пользовательской ракеткой в приостановленной игре |
| Запуск-приостановка-продолжение после 1 победы/поражения { Start/Continue ->Pause-> Start/Continue } | Остановка и возобновление движения всех игровых объектов, отсутствие реакции на управление пользовательской ракеткой в приостановленной игре, ко всему этому сохранение состояния игрового счёта |
| Запуск-проигрыш-остановка игры{ Start/Continue ->Stop } | При завершении игры счет обнуляется, все игровые объекты возвращаются |

Табл. 4.3.1 сценарии тестирования состояния игры

|  |  |
| --- | --- |
| Действия татуировщика | Ожидаемый результат |
| Запуск игры {Game->Start/Continue} | Мяч отскакивает от обеих ракеток в зависимости от угла падения, а при попадании на верхнюю/нежную границы, угол падения = углу отражения. |
| Запуск игры {Game->Start/Continue} | При попадании меча за ракетку, счёт увеличивается, мяч возвращается в центр поля и начинает своё движение в сторону проигравшего игрока. |

Табл. 4.3.2 сценарии тестирования физики игры

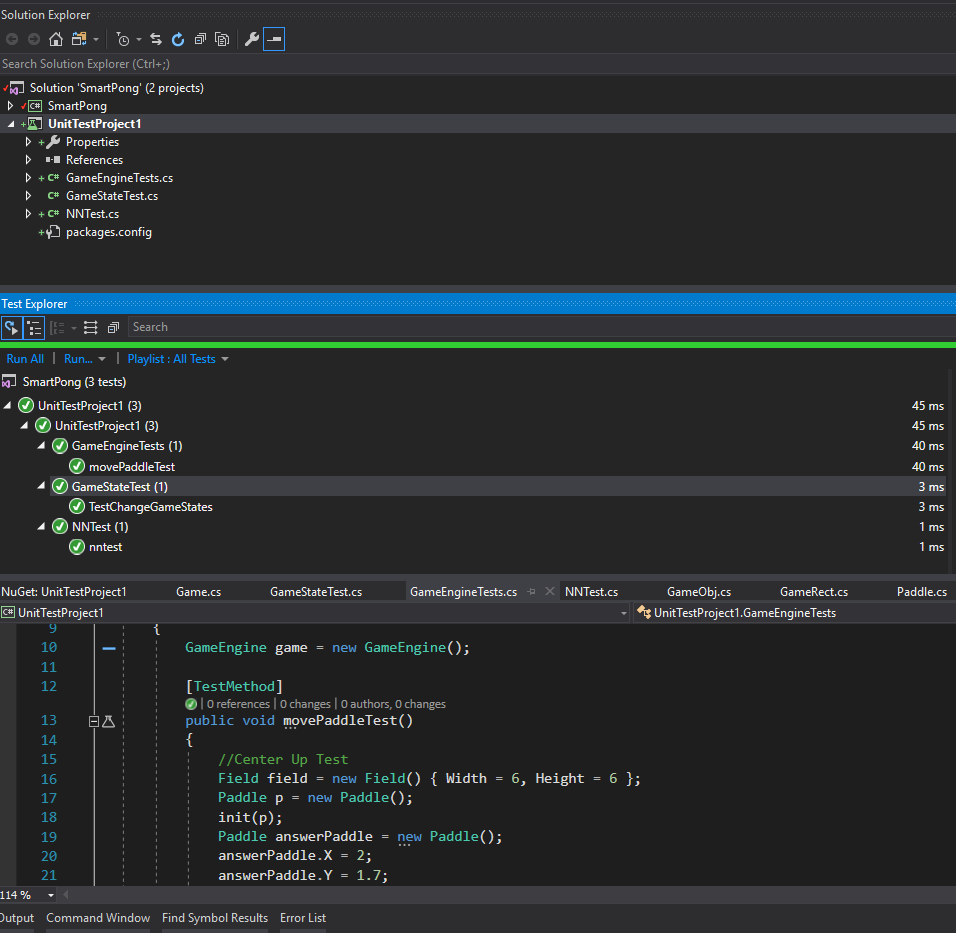


Рис. 5.1 Финальная диаграмма зависимостей

## Пример работы программы

При запуске программы открывается главное окно приложение, где пользователь может начать новую игру, видеть текущий статус игры и оппонента, счёт (рис 6.1.).

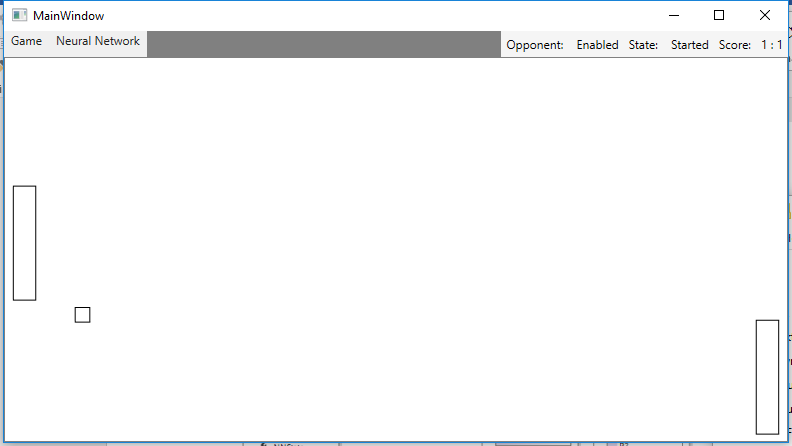


Рис. 6.1. Пример работы программы

# Заключение

Данная курсовая работа посвящена разработке игры в Pong. Авторы поставили перед собой цель воплотить соперника в лице нейронной сети. В группе разработчиков основным языком проекта был выбран английский, соответственно все артефакты, полученные в ходе работы, составлены на основном языке проекта. В рамках данной работы были пройдены все этапы разработки программного обеспечения. На первом этапе были определены требованья и функции к программному обеспечению с начальной cсценарий использования, так же был сделан макет пользовательского интерфейса. На втором этапе проектирования были дополнены существующие и разработаны диаграммы последовательностей, состояний и классов. Так же были разработаны и представлены 2 алгоритмов, решающих основные подзадачи: отражения меча от игровых объектов и взаимодействие с ними. В качестве языка программирования был выбран C# и использовалась Visual Studio 2017, как среда разработки. Для реализации игры использовалась графическая подсистема Windows Presentation Foundation с архитектурным паттерном MVVM.

Итого получилось 1017 строк кода, который был разделен на 18 файлов с исходным кодом. Всего было разработано: 26 типов данных, 5 пространств имен и 44 метода.

В результате работы, авторы разработали игру типа Pong, согласно требованьям. Игра представляет собой игровое поле с 2 прямоугольниками, представленных за ракетки и квадрата меньшего размером, в роле мяча. Игрок управляет одной ракеткой при помощи клавиш на клавиатуре. Вторая ракетка управляется нейронной сетью. Игра позволяет отслеживать свое состояние, в которое входит: игровой счет, текущее состояние игры и состояние нейронной сети. Интерфейс игры предоставляет возможность управлять жизненным циклом игры, а также состоянием нейронной сети.

Программа была протестирована. Всего было разработано 3 набора тестов для основных классов, отвечающих за логику игры. В каждом наборе реализовано несколько тестов, проверяющих классы на ожидаемый от них функционал.

В ходе выполнения данной курсовой работы авторы осознали, что тщательное проектирование архитектуры программного продукта сокращает время затраты на этап кодирования, т.к. разработчикам, в принципе, не нужно заботится о логике и других аспектах архитектуры, кодер просто переводит уже четко сформулированную в диаграммах концепцию программы. Полное понимание начальных требований и их документация позволяет определить больше информации на этапе проектирования, что приведёт к уменьшению затрат на работу программистов.

# Список используемой литературы

1. С. А. Орлов. Технологии Разработки Программного Обеспечения. Разработка сложных программных систем.
2. Мартин Р. Чистый код. Создание, анализ и рефакторинг. Библиотека программиста. Питер, 2017
3. MVVM Паттерн. <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>
4. Ball physics <https://www.real-world-physics-problems.com/bouncing-ball-physics.html>
5. Нейронные сети для начинающих. Часть 1. <https://habr.com/post/312450/>

# Приложение

## Файл ViewModel.cs

using SmartPong.Model;

using SmartPong.Model.GameObjects;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace SmartPong

{

public class ViewModel : ViewModelBase

{

private Game model;

private RelayCommand arrowDownCommand;

private RelayCommand arrowUpCommand;

private RelayCommand startContinueCommand;

private RelayCommand nnEnableDisableCommand;

private RelayCommand pauseCommand;

private RelayCommand stopCommand;

private RelayCommand quitCommand;

public string GameState

{

get {

return model.StateString;

}

}

public string NNState

{

get

{

return model.NNStateString;

}

}

public GameAttributes GameAttr

{

get

{

return model.GameAttributes;

}

}

public ViewModel()

{

model = new Game(

()=>

OnPropertyChanged("GameAttr")

);

}

public void ResizeEvent(double x, double y)

{

model.FieldResize(x,y);

OnPropertyChanged("GameAttr");

}

public RelayCommand ArrowUpCommand

{ get

{

if (arrowUpCommand == null) {

arrowUpCommand = new RelayCommand(

() => {

model.MovePlayerPaddle(0);

OnPropertyChanged("GameAttr");

}

);

}

return arrowUpCommand;

} }

public RelayCommand ArrowDownCommand

{

get

{

if (arrowDownCommand == null)

{

arrowDownCommand = new RelayCommand(

() => {

model.MovePlayerPaddle(1);

OnPropertyChanged("GameAttr");

}

);

}

return arrowDownCommand;

}

}

public RelayCommand StartContinueCommand

{

get

{

return startContinueCommand != null ? startContinueCommand : startContinueCommand = new RelayCommand(

() =>

{

if(model.State == Game.GameState.Paused)

model.ChangeGameState(Game.GameCommands.Continue);

else

model.ChangeGameState(Game.GameCommands.Start);

OnPropertyChanged("GameState");

}

);

}

}

public RelayCommand NNEnableDisableCommand

{

get

{

return nnEnableDisableCommand != null ? nnEnableDisableCommand : nnEnableDisableCommand = new RelayCommand(

() =>

{

if (model.NNState == Game.NeuralNetworkState.Disabled)

model.ChangeNNState(Game.NeuralNetworkState.Enabled);

else

model.ChangeNNState(Game.NeuralNetworkState.Disabled);

OnPropertyChanged("NNState");

}

);

}

}

public RelayCommand PauseCommand { get {

return pauseCommand != null ? pauseCommand : pauseCommand = new RelayCommand(

() =>

{

model.ChangeGameState(Game.GameCommands.Pause);

OnPropertyChanged("GameState");

}

);

}

}

public RelayCommand StopCommand

{

get

{

return stopCommand != null ? stopCommand : stopCommand = new RelayCommand(

() =>

{

model.ChangeGameState(Game.GameCommands.Stop);

OnPropertyChanged("GameState");

}

);

}

}

public RelayCommand QuitCommand

{

get

{

return quitCommand != null ? quitCommand : quitCommand = new RelayCommand(

() =>

{

Application.Current.MainWindow.Close();

}

);

}

}

}

}

## Файл MainWindow.xaml

<Window x:Class="SmartPong.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:SmartPong"

xmlns:gattr="clr-namespace:SmartPong.Model.GameObjects"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Window.DataContext>

<local:ViewModel></local:ViewModel>

</Window.DataContext>

<Grid Background="Gray">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="auto"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

</Grid.RowDefinitions>

<DockPanel Grid.Row="0"

Grid.Column="0"

>

<Menu IsMainMenu="True" DockPanel.Dock="Left">

<MenuItem Header="Game">

<MenuItem Header="Start/Continue" Command="{Binding StartContinueCommand}"></MenuItem>

<MenuItem Header="Pause" Command="{Binding PauseCommand}"></MenuItem>

<MenuItem Header="Stop" Command="{Binding StopCommand}"></MenuItem>

<MenuItem Header="Quit" Command="{Binding QuitCommand}"></MenuItem>

</MenuItem>

<MenuItem Header="Neural Network">

<MenuItem Header="Enable/Disable" Command="{Binding NNEnableDisableCommand}"></MenuItem>

</MenuItem>

</Menu>

<Label DockPanel.Dock="Right"

Content="{Binding GameAttr.Score}"

HorizontalAlignment="Right"

Background="WhiteSmoke"></Label>

<Label DockPanel.Dock="Right"

Content="Score:"

HorizontalAlignment="Right"

Background="WhiteSmoke"></Label>

<Label DockPanel.Dock="Right"

Content="{Binding GameState}"

HorizontalAlignment="Right"

Background="WhiteSmoke"></Label>

<Label DockPanel.Dock="Right"

Content="State: "

HorizontalAlignment="Right"

Background="WhiteSmoke"></Label>

<Label DockPanel.Dock="Right"

Content="{Binding NNState}"

HorizontalAlignment="Right"

Background="WhiteSmoke"></Label>

<Label DockPanel.Dock="Right"

Content="Opponent: "

HorizontalAlignment="Right"

Background="WhiteSmoke"></Label>

</DockPanel>

<StackPanel x:Name="GameField"

MouseDown="GetGameFieldFocus"

SizeChanged="GameFieldSizeChanged"

FocusManager.IsFocusScope="True"

Grid.Row="1"

Background="White"

Margin="1">

<StackPanel.InputBindings>

<KeyBinding

Key="Up"

Command="{Binding ArrowUpCommand}"></KeyBinding>

<KeyBinding

Key="Down"

Command="{Binding ArrowDownCommand}"></KeyBinding>

</StackPanel.InputBindings>

<Canvas>

<Border

BorderBrush="Black"

BorderThickness="1"

Height="{Binding GameAttr.PlayerPaddle.Height}"

Width="{Binding GameAttr.PlayerPaddle.Width}"

Canvas.Left="{Binding GameAttr.PlayerPaddle.X}"

Canvas.Top="{Binding GameAttr.PlayerPaddle.Y}"/>

<Border

BorderBrush="Black"

BorderThickness="1"

Height="{Binding GameAttr.NewralNetworkPaddle.Height}"

Width="{Binding GameAttr.NewralNetworkPaddle.Width}"

Canvas.Left="{Binding GameAttr.NewralNetworkPaddle.X}"

Canvas.Top="{Binding GameAttr.NewralNetworkPaddle.Y}"/>

<Border

x:Name="GameBall"

BorderBrush="Black"

BorderThickness="1"

Height="{Binding GameAttr.PongBall.Side}"

Width="{Binding ElementName=GameBall, Path=Height}"

Canvas.Left="{Binding GameAttr.PongBall.X}"

Canvas.Top="{Binding GameAttr.PongBall.Y}">

</Border>

</Canvas>

</StackPanel>

</Grid>

</Window>

## Файл Game.cs

using SmartPong.Model.GameObjects;

using System;

using System.Timers;

namespace SmartPong.Model

{

public class Game : INotifyObject

{

private bool init = false;

private Timer gameTimer = new Timer();

public enum GameCommands { Start, Pause, Stop, Continue };

public enum GameState { Started, Paused, Stopped };

public GameState State { get; private set; }

public string StateString

{

get

{

string state = "";

switch (State)

{

case GameState.Started:

state = "Started";

break;

case GameState.Paused:

state = "Paused";

break;

case GameState.Stopped:

state = "Stopped";

break;

}

return state;

}

}

public enum NeuralNetworkState { Enabled, Disabled };

public NeuralNetworkState NNState { get; private set; }

public string NNStateString

{

get

{

return NNState.ToString();

}

}

public GameAttributes GameAttributes { get; set; }

public event Action Game\_Tick;

private GameEngine gameEngine;

private NeuralNetwork neuralNetwork;

//Newral Network

public Game(Action action)

{

Game\_Tick += action;

gameEngine = new GameEngine();

neuralNetwork = new NeuralNetwork();

GameAttributes = new GameAttributes();

//Timer setup

gameTimer.Interval = 5;

gameTimer.Elapsed += GameTimerElapsed;

gameTimer.Enabled = false;

State = GameState.Stopped;

NNState = NeuralNetworkState.Enabled;

}

public void FieldResize(double x, double y)

{

var paddleW = x \* 0.03;

var paddleH = y \* 0.3;

if (!init)

{

InitGame(x, y);

init = true;

}

var koefX = (x / GameAttributes.PongField.Width);

var koefY = (y / GameAttributes.PongField.Height);

//Ball configuration

GameAttributes.PongBall.Side = x \* 0.02;

GameAttributes.PongBall.X \*= koefX;

GameAttributes.PongBall.Y \*= koefY;

//Player paddle configuration

GameAttributes.PlayerPaddle.Width = paddleW;

GameAttributes.PlayerPaddle.Height = paddleH;

GameAttributes.PlayerPaddle.X \*= koefX;

GameAttributes.PlayerPaddle.Y \*= koefY;

//NN paddle configuration

GameAttributes.NewralNetworkPaddle.Width = paddleW;

GameAttributes.NewralNetworkPaddle.Height = paddleH;

GameAttributes.NewralNetworkPaddle.X \*= koefX;

GameAttributes.NewralNetworkPaddle.Y \*= koefY;

//Field

GameAttributes.PongField.Height = y;

GameAttributes.PongField.Width = x;

}

public void MovePlayerPaddle(int i)

{

if (i == 0)

gameEngine.MovePaddle(GameAttributes.PlayerPaddle, GameAttributes.PongField, GameEngine.Direction.Up);

else

gameEngine.MovePaddle(GameAttributes.PlayerPaddle, GameAttributes.PongField, GameEngine.Direction.Down);

}

private void MoveNeuralPaddleUp()

{

gameEngine.MovePaddle(GameAttributes.NewralNetworkPaddle, GameAttributes.PongField, GameEngine.Direction.Up);

}

private void MoveNeuralPaddleDown()

{

gameEngine.MovePaddle(GameAttributes.NewralNetworkPaddle, GameAttributes.PongField, GameEngine.Direction.Down);

}

private void GameTimerElapsed(object sender, ElapsedEventArgs e)

{

if (NNState == NeuralNetworkState.Enabled)

neuralNetwork.Run(GameAttributes.PongBall.Y, GameAttributes.NewralNetworkPaddle.Y, MoveNeuralPaddleUp, MoveNeuralPaddleDown);

var winner = gameEngine.NextFrame(

GameAttributes.PongBall,

GameAttributes.PongField,

GameAttributes.PlayerPaddle,

GameAttributes.NewralNetworkPaddle);

switch (winner)

{

case GameEngine.Winner.P1:

GameAttributes.ScorePlayer += 1;

InitGame(GameAttributes.PongField.Width, GameAttributes.PongField.Height);

GameAttributes.PongBall.Angle = 0;

break;

case GameEngine.Winner.P2:

GameAttributes.ScoreNewralNetwork += 1;

InitGame(GameAttributes.PongField.Width, GameAttributes.PongField.Height);

GameAttributes.PongBall.Angle = 180;

break;

case GameEngine.Winner.NONE:

break;

}

Game\_Tick?.Invoke();

}

private void InitGame(double x, double y)

{

var paddleW = x \* 0.03;

GameAttributes.PongBall.X = x / 2.0;

GameAttributes.PongBall.Y = y / 2.0;

GameAttributes.PlayerPaddle.X = x \* 0.01;

GameAttributes.PlayerPaddle.Y = y / 3.0;

GameAttributes.NewralNetworkPaddle.X = (x \* 0.99) - paddleW;

GameAttributes.NewralNetworkPaddle.Y = y / 3.0;

GameAttributes.PongField.Width = x;

GameAttributes.PongField.Height = y;

}

public void ChangeGameState(GameCommands cmd)

{

switch (cmd)

{

case GameCommands.Start:

InitGame(GameAttributes.PongField.Width, GameAttributes.PongField.Height);

GameAttributes.ScorePlayer = 0;

GameAttributes.ScoreNewralNetwork = 0;

State = GameState.Started;

gameTimer.Enabled = true;

break;

case GameCommands.Pause:

State = GameState.Paused;

gameTimer.Enabled = false;

break;

case GameCommands.Stop:

State = GameState.Stopped;

gameTimer.Enabled = false;

InitGame(GameAttributes.PongField.Width, GameAttributes.PongField.Height);

break;

case GameCommands.Continue:

State = GameState.Started;

gameTimer.Enabled = true;

break;

default:

break;

}

}

public void ChangeNNState(NeuralNetworkState state)

{

NNState = state;

}

}

}

## Файл GameEngine.cs

using SmartPong.Model.GameObjects;

using System;

using System.Diagnostics;

namespace SmartPong.Model

{

class GameEngine

{

public enum Winner { P1, P2,NONE }

public enum Direction { Up,Down}

public void MovePaddle(Paddle paddle,Field field, Direction direction)

{

double hop = field.Height \* 0.05;

switch (direction)

{

case Direction.Up:

if (0 < paddle.Y - hop)

paddle.Y -= hop;

break;

case Direction.Down:

if (field.Height > paddle.Y + hop + paddle.Height)

paddle.Y += hop;

break;

default:

break;

}

}

public Winner NextFrame(Ball ball, Field field, Paddle playerPaddle,Paddle nnPadle)

{

double angleRag = (ball.Angle \* Math.PI) / 180;

double dx = Math.Cos(angleRag) \* (field.Width \* 0.008);

double dy = Math.Sin(angleRag) \* (field.Height \* 0.008);

ball.X += dx;

ball.Y += dy;

//Field physics

if (field.Height < ball.RightBottom.Y || 0 > ball.Y)

ball.Angle \*= -1;

//Win conditions

if (field.Width < ball.RightBottom.X)

return Winner.P1;

if (0 > ball.X)

return Winner.P2;

//Player paddle

if (playerPaddle.RightBottom.X > ball.X

&& playerPaddle.LeftTop.Y < ball.RightBottom.Y

&& playerPaddle.RightBottom.Y > ball.Y)

{

double k = (playerPaddle.Height - (ball.Y-playerPaddle.Y))/ playerPaddle.Height;

ball.Angle = 90 - 180\*k;

ball.X = playerPaddle.X + playerPaddle.Width;

}

//NN paddle

if (nnPadle.X < ball.RightBottom.X

&& nnPadle.Y < ball.RightBottom.Y

&& nnPadle.RightBottom.Y > ball.Y)

{

double k = (nnPadle.Height - (ball.Y - nnPadle.Y)) / nnPadle.Height;

ball.Angle = -270 + 180 \* k;

ball.X = nnPadle.X - ball.Width;

}

return Winner.NONE;

}

static int i = 0;

private void PaddleCollision(Ball ball, Paddle paddle)

{

if (IsIntoPaddle(ball, paddle))

{

ball.Angle = -ball.Angle;

ball.X = paddle.RightBottom.X+0.1;

Debug.WriteLine(i + ": Angle = " + ball.Angle + "\n Ball Coord: " + ball.X + "\n Paddle Coord: " +( paddle.X + paddle.Width));

i++;

}

}

private bool IsIntoPaddle(Ball ball, Paddle paddle) {

var blt = ball.LeftTop;

var brb = ball.RightBottom;

var plt = paddle.LeftTop;

var prb = paddle.RightBottom;

return blt.X < prb.X && blt.X > plt.X ;

}

}

}

## Файл NeuralNetwork.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SmartPong.Model

{

class NeuralNetwork

{

public void Run(double input1, double input2, Action output1, Action output2)

{

if (input2 > input1)

output1();

else

output2();

}

}

}