|  |
| --- |
| Министерство образования Республики Беларусь  Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники» |
| **Пояснительная записка**  к курсовому проекту по курсу «Объектно-ориентированное программирование»  Тема: «Игра Starship Seven» |
| Выполнил ст. гр. 052003  Смоляков К.Ю.  Проверил  Юзефович С.В. |

|  |
| --- |
| Минск, 2012 |

Оглавление

[1. Введение 2](#_Toc325910413)

[2. Анализ предметной области 3](#_Toc325910414)

[2.1. Проектирование программного продукта 3](#_Toc325910415)

[*Функциональные требования к программе* 3](#_Toc325910416)

[2.2. Обзор существующих аналогов 4](#_Toc325910417)

[2.3. Выбор языка программирования и среды разработки 5](#_Toc325910418)

[3. Подробности реализации 5](#_Toc325910419)

[3.1. Шаблон проектирования MVVM 5](#_Toc325910420)

[3.2. Основные интерфейсы 6](#_Toc325910421)

[3.3. Конкретная реализация 10](#_Toc325910422)

[3.4. Механизм добавления стратегий 23](#_Toc325910423)

[3.5. Реализация отображения карты в игровом окне 27](#_Toc325910424)

[3.6. Позднее связывание и инверсия управления 28](#_Toc325910425)

[4. Игровой процесс 30](#_Toc325910426)

[5. Заключение 32](#_Toc325910427)

[6. Список использованной литературы 32](#_Toc325910428)

# Введение

Казуальные игры - это небольшие, красочные и увлекательные компьютерные игры, отличающиеся простым управлением и легкостью в освоении. Неотъемлемой частью подобных игр являются интуитивно понятный игровой процесс, яркая графика, а также отсутствие элементов насилия и жестокости. Чаще всего пользователю нет необходимости тратить свое время на ознакомление с правилами, поэтому он может начать игру сразу же после ее инсталляции. При этом частота и длительность игровых сеансов может варьироваться в крайне широких пределах. Проще говоря, казуальные игры это приятное времяпрепровождение, отдых, возможность отвлечься от работы или рутинных занятий и с интересом провести несколько минут у экрана компьютера.

|  |  |
| --- | --- |
| http://www.kozlenko.info/wp-content/uploads/2009/10/Solitaire-Windows7.png | http://static-www.ec.popcap.com/www.popcap.com/sites/all/themes/popcap_2012/games/bejeweled2/screenshots/bejeweled21.jpg |

*Игра «Солитер» (слева) и «Bejeweled» (справа)*

К сожалению, в русскоязычных странах до сих пор не сложилось единого и понятного любому русскоязычному пользователю названия для этого вида платного контента. За время работы компании Alawar на российском рынке, было опробовано множество названий: "игры формата casual", "casual-игры", "shareware-игры" (по основному методу распространения), "скачиваемые", "загружаемые", "офисные", "casual-игры", "маленькие", "игры на пять минут", "игры на каждый день", "игры в стиле casual", "coffee-break-игры", "тайм-киллеры". Но рабочее название "казуальные игры" закрепилось как постоянное. Однако у него есть свои недостатки: поскольку образовано оно с помощью транслитерации, то зачастую не является понятным русскоязычному потребителю. К тому же в качестве перевода слова "casual" словарь общей лексики английского языка предлагает такие варианты как "случайный", "невольный", "бессистемный", "легкомысленный", "несерьезный" и т.д. Получается, что casual games - это игры для людей, нерегулярно посвящающих свое время играм и выбирающим эти игры бессистемно, что, по сути, не совсем верно. В итоге, прямой перевод не дает возможности одним словом описать данный игровой формат. Поэтому вопрос о более подходящем названии остается открытым.

Пасьянс «Косынка» от Microsoft, бесплатно распространявшийся с Windows, признается как первая успешная казуальная игра, в которую, с момента выхода, играло более 400 млн человек. В 1989 году компания Nintendo выпустила на Game Boy игру Тетрис. Тетрис на Game Boy оказался чрезвычайно популярен и стал одной из игр, которые принесли успех новой на тот момент консоли.

Казуальные игры дебютировали в онлайне в 1996 году на сайтах Gamesville и Uproar, которые предлагали мультиплеерные HTML-игры таких жанров, как карточные игры, бинго, пазлы. Появление технологии флеш вызвало бум в развитии онлайн-игр, но в то же время вынудило разработчиков ограничиваться использованием одной кнопки мыши и создавать простые игры, которые можно завершить за короткую игровую сессию. Одна из наиболее известных казуальных игр, Bejeweled, появилась в 2001 году как флеш-игра.

В 2005 году крупнейшие разработчики, издатели и дистрибьюторы казуальных игр объединились в Ассоциацию казуальных игр (Casual Games Association).

В развитии индустрии казуальных игр не последнюю роль сыграла общедоступность Интернета — зародившиеся технологии электронной коммерции (платёжные сервисы) и передача дистрибутивов через Интернет позволили сделать механизм покупки и получения игр быстрым и удобным.

# Анализ предметной области

## Проектирование программного продукта

Starship Seven – простая пошаговая стратегическая игра, суть которой заключается в том, чтобы захватить все планеты соперника. Справедливы следующие правила игры:

* Карта галактики делится на сектора, количество которых задается перед началом игры. На карте случайным образом расставляются планеты. В начале игры каждый игрок обладает одной планетой, остальные планеты принадлежат нейтральному сопернику. Карта генерируется случайным образом.
* Каждая планета обладает следующими характеристиками:

|  |  |
| --- | --- |
| Название планеты | В игре по умолчанию в качестве названий планет используются буквы латинского алфавита. |
| Производственная мощность | Количество кораблей, производимых планетой за один ход (варьируется от 3 до 15) |
| Боевая мощность | Число от 0,3 до 0,9 (30-90% соответственно) Характеризует шанс на победу у флотов, посланных с данной планеты. |
| Количество кораблей | Количество кораблей, находящихся в данный момент на планете. |

* Задача игрока – запускать корабли с одной планеты на другую. Флот перенимает боевую мощность планеты, с которой он был запущен. Расстояние между планетами заранее известно (количество секторов при движении от одной планеты к другой без возможности движения по диагонали). Игрок может видеть характеристики как своих, так и вражеских планет. Можно также пересылать корабли между своими планетами, в этом случае корабли, прибывшие на планету назначения, полностью перенимают характеристики новой планеты (при последующем запуске будут иметь боевую мощность новой планеты).
* После завершения хода отменить запуск флота невозможно.
* По умолчанию битвы проводятся по следующему принципу:

Атакующий флот имеет мощность планеты, с которой он был отправлен. Защищающийся флот перенимает эту характеристику у планеты, на которую совершается нападение. Каждый флот «выбрасывает кубик», который может принимать значения от 0 до 1,0. Если боевая мощность находится в пределах значения «кубика», то флот убивает один корабль противника. Так продолжается, пока один из флотов не проиграет.

* Игра считается выигранной, если захвачены все планеты противника (необязательно захватывать все нейтральные планеты)

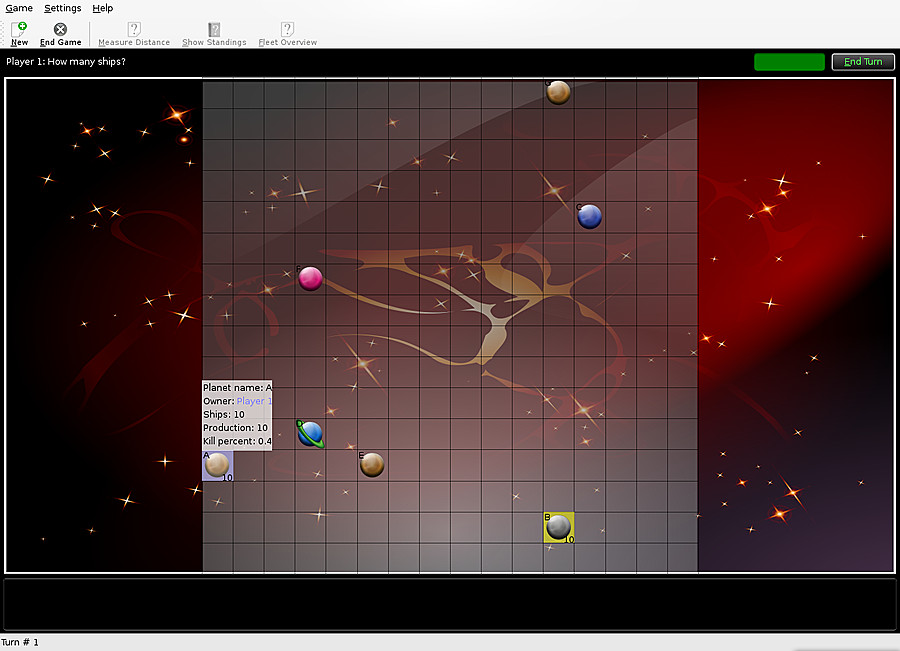
*Функциональные требования к программе*

* Необходимо разработать оконное Windows-приложение. Приложение будет содержать три окна: конструктор игры, основное игровое окно и меню.
* Конструктор позволяет задавать ширину и высоту карты, количество нейтральных планет, а также настраивать количество и тип игроков: «игрок-человек» и «игрок-компьютер».
* Основное игровое окно должно содержать карту галактики, а также должно предоставлять возможность просмотра характеристик планет. Возможность запускать корабли с планеты на планету. Также в окне показываются сообщения об игровом процессе: какая планета захвачена, какая смогла защититься, куда прибыло подкрепление.
* Окно игрового меню позволяет просмотреть список сохраненных игр, загрузить и сохранить игру, начать новую игру.
* Компьютерный противник нескольких уровней сложности. Возможность создания искусственного интеллекта компьютерного противника сторонними разработчиками (реализация через позднее связывание).
* Просмотр ходов компьютерного противника (данная функция полезна в образовательных целях, так как позволяет увидеть, как работает искусственный интеллект игрока-компьютера)
* Возможность сохранения и загрузки игры.

## Обзор существующих аналогов

Аналогом является игра Konquest, которая поставляется вместе с графической оболочкой KDE, используемой в операционных системах семейства Linux. Она обладает неплохим интерфейсом, тем не менее, он не очень удобен и недостаточно прост для казуальной игры. Konquest является open-source проектом и разрабатывается на языке C++ с использованием Qt.

*Игра «Konquest»*



## Выбор языка программирования и среды разработки

* Для казуальных игр важна стоимость и быстрота разработки. Кроме того данное приложение не требует сложной графики, поэтому можно обойтись без графических движков и библиотек для создания игр.
* Исходя из этих соображений, был выбран язык C#, один из языков платформы .NET. Это объектно-ориентированный язык, объединивший лучшие идеи всех современных языков программирования. Кроме того технология Reflection, входящая в .NET, позволит связать компоненты таким образом, чтобы обеспечить их изменяемость без ущерба для всего приложения. Позднее связывание сборок можно реализовать с помощью Unity IoC.
* Для создания оконного приложения решено использовать технологию WPF (Windows Presentation Foundation). Она позволяет с минимальными затратами создавать яркие, интуитивно понятные интерфейсы за счет широкого применения стилей и шаблонов. К этой технологии применим известный шаблон проектирования MVVM (Model-View-ViewModel), который позволит создать гибкую архитектуру.
* В качестве среды разработки будет использоваться Microsoft Visual Studio 2010. Это естественный выбор для разработки под Windows на языке C#.

# Подробности реализации

## Шаблон проектирования MVVM

Шаблон Model-View-ViewModel применяется при проектировании архитектуры приложения. Первоначально был представлен сообществу Джоном Госсманом (John Gossman) в 2005 году как модификация шаблона Presentation Model. MVVM ориентирован на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft, используется в качестве основной модели для Java-приложений на платформе Google Android.

MVVM используется для разделения модели и её представления, что необходимо, так как позволяет изменять их отдельно друг от друга. Например, программист задает логику работы с данными, а дизайнер соответственно работает с пользовательским интерфейсом.

Использование MVVM удобно использовать вместо классического MVC и ему подобных в тех случаях, когда в платформе, на которой ведется разработка, присутствует «связывание данных».

В MVC/MVP изменения в пользовательском интерфейсе не влияют непосредственно на модель, а предварительно идут через Контроллер/Presenter. В таких технологиях как WPF и Silverlight есть концепция «связывания данных», позволяющая связывать данные с визуальными элементами в обе стороны. Следовательно при использовании этого приема применение модели MVC становится крайне неудобным из-за того, что привязка данных к представлению напрямую не укладывается в концепцию MVC/MVP.

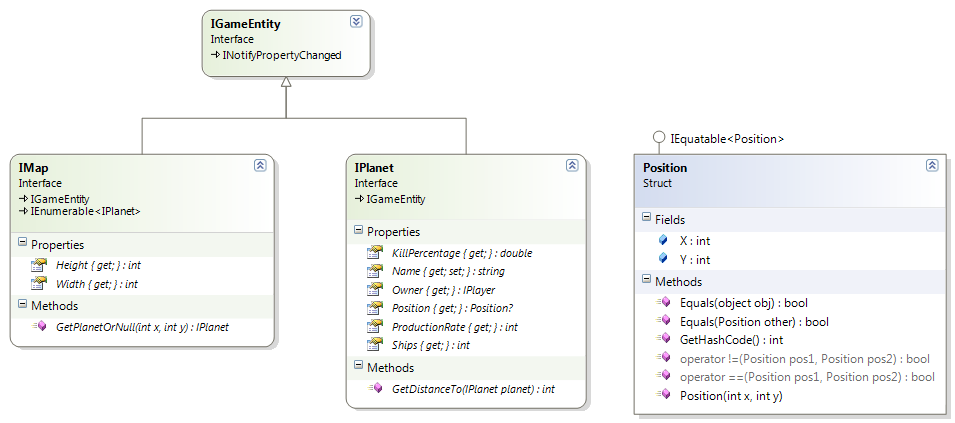
Паттерн MVVM делится на три части:

* Модель (Model), так же, как в классической MVC, Модель представляет собой фундаментальные данные, необходимые для работы приложения.
* Вид/Представление (View) так же, как в классической модели MVC, Вид — это графический интерфейс, то есть окно, кнопки и.т.п.
* Модель вида (ViewModel, что означает «Model of View») является с одной стороны абстракцией Вида, а с другой предоставляет обертку данных из Модели, которые подлежат связыванию. То есть она содержит Модель, которая преобразована к Виду, а также содержит в себе команды, которыми может пользоваться Вид, чтобы влиять на Модель.

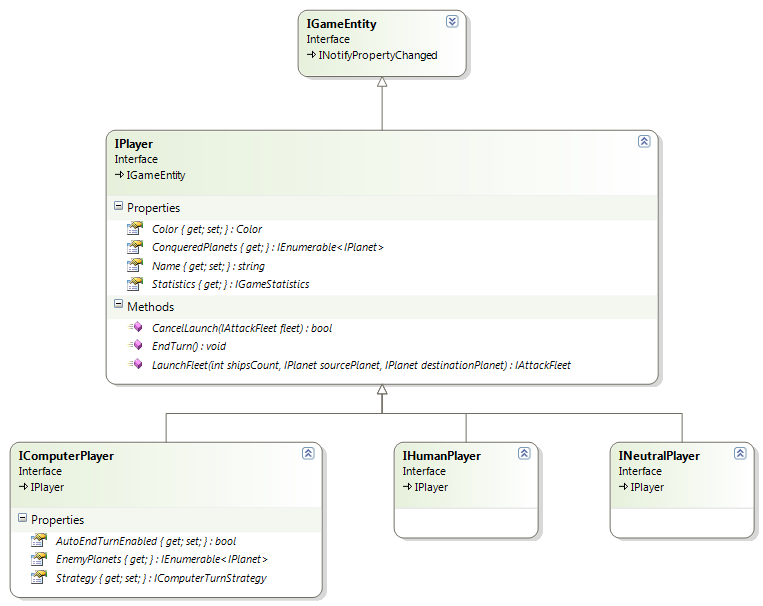
## Основные интерфейсы

В своем приложении я решил отказаться от использования на верхних уровнях архитектуры конкретных классов. Вместо этого, используются следующие интерфейсы.

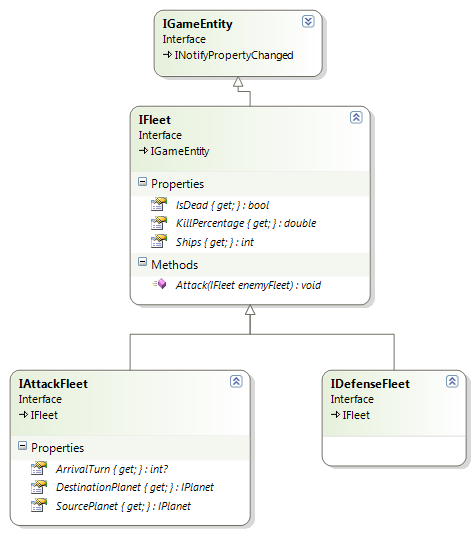
* *Игровые сущности «карта» и «планета».*

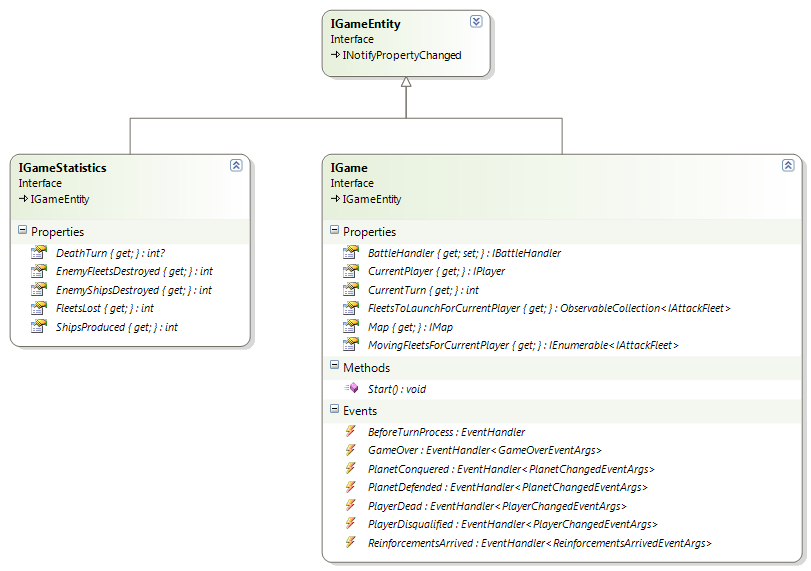


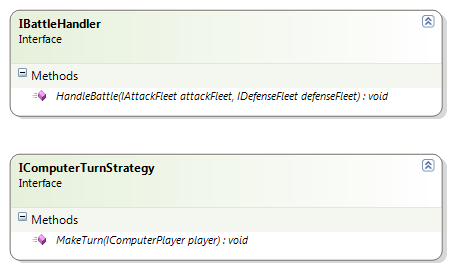
* *«Игрок» и его наследники».*

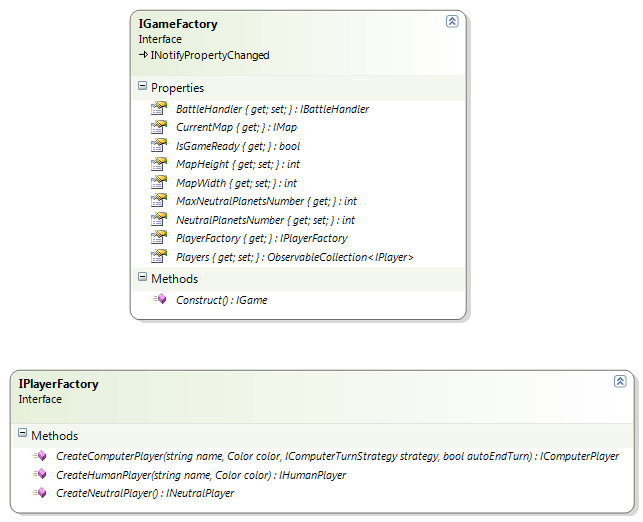


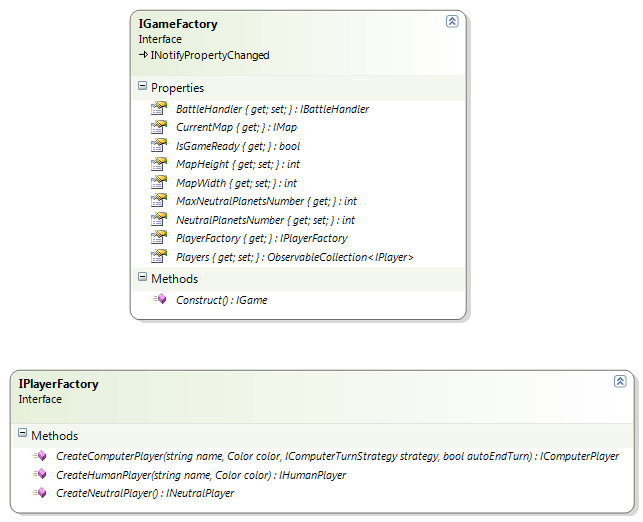
* *«Атакующий и защищающийся флот»*



* *Сущности «Игровая статистика» и «Игра»*
* *Интерфейсы, отвечающие за обработку битв и за стратегию компьютерного противника.*



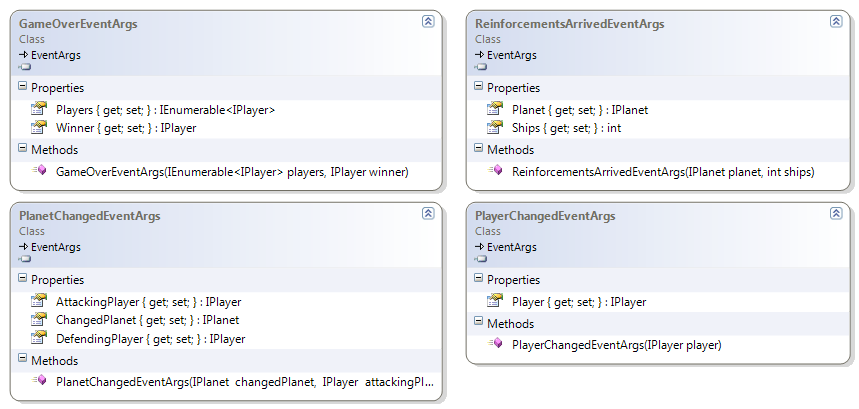
* *Интерфейсы-фабрики, отвечающие за конструирование игры с соответствующими параметрами*



* *Менеджер стратегий*

|  |  |
| --- | --- |
|  |  |

* *Игровые события*



* *Инфраструктура для сохранения и загрузки игры.*

**

**

## Конкретная реализация

* ***Базовый класс для всех игровых сущностей: игрок, планета и т.д.***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.ComponentModel;

using StarshipSeven.GameInterfaces;

namespace StarshipSeven.GameEntities

{

public class BaseGameEntity<TKey> : IGameEntity, IEquatable<BaseGameEntity<TKey>>

{

private TKey \_id;

public BaseGameEntity()

{

}

public BaseGameEntity(TKey id)

{

Id = id;

}

public TKey Id

{

get { return \_id; }

set { SetProperty<TKey>(ref \_id, value, "Id"); }

}

protected void SetProperty<T>(ref T field, T value, string propertyName)

{

if (!EqualityComparer<T>.Default.Equals(field, value))

{

field = value;

OnPropertyChanged(propertyName);

}

}

protected void OnPropertyChanged(string propertyName)

{

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

public event PropertyChangedEventHandler PropertyChanged;

public bool Equals(BaseGameEntity<TKey> other)

{

if (other == null)

return false;

return EqualityComparer<TKey>.Default.Equals(this.Id, other.Id);

}

}

}

* ***Класс «Player»***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces;

using System.Collections.ObjectModel;

using StarshipSeven.GameEntities.Exceptions;

using StarshipSeven.GameEntities.Events;

using System.Windows.Media;

namespace StarshipSeven.GameEntities

{

public class Player : BaseGameEntity<Guid>, IPlayer

{

private string \_name;

private Color \_color;

private GameStatistics \_statistics;

private Game \_game;

protected ObservableCollection<IAttackFleet> \_fleetsToLaunch = new ObservableCollection<IAttackFleet>();

public Player() : base(Guid.NewGuid())

{

\_statistics = new GameStatistics(this);

}

public Player(string name, Color color) : base(Guid.NewGuid())

{

\_statistics = new GameStatistics(this);

Name = name;

Color = color;

}

public Player(Guid id)

: base(id)

{

\_statistics = new GameStatistics(this);

}

#region IPlayer Members

public string Name

{

get { return \_name; }

set { SetProperty<string>(ref \_name, value, "Name"); }

}

public Color Color

{

get { return \_color; }

set { SetProperty<Color>(ref \_color, value, "Color"); }

}

public IEnumerable<IPlanet> ConqueredPlanets

{

get { return (Game.Map as Map).GetPlanetsForPlayer(this); }

}

public IAttackFleet LaunchFleet(int shipsCount, IPlanet sourcePlanet, IPlanet destinationPlanet)

{

if (!(sourcePlanet is Planet && destinationPlanet is Planet))

throw new TypeMismatchException(typeof(Planet), destinationPlanet);

if (!(Game.CurrentPlayer as Player).Equals(this))

throw new CheatingException(this);

AttackFleet fleet = (sourcePlanet as Planet).ProduceAttackFleet(shipsCount, destinationPlanet as Planet, this);

fleet.ArrivalTurn = Game.CurrentTurn + sourcePlanet.GetDistanceTo(destinationPlanet);

\_fleetsToLaunch.Add(fleet);

return fleet;

}

public bool CancelLaunch(IAttackFleet fleet)

{

if (!(fleet is AttackFleet))

throw new TypeMismatchException(typeof(AttackFleet), fleet);

(fleet.SourcePlanet as Planet).Ships += fleet.Ships;

return \_fleetsToLaunch.Remove(fleet as AttackFleet);

}

public virtual void EndTurn()

{

OnEndTurn();

}

public IGameStatistics Statistics

{

get { return \_statistics; }

}

#endregion

internal Game Game

{

get { return \_game; }

set { SetProperty<Game>(ref \_game, value, "Game"); }

}

internal ObservableCollection<IAttackFleet> FleetsToLaunch

{

get { return \_fleetsToLaunch; }

}

internal event EventHandler<EndTurnEventArgs> EndTurnEvent;

protected void OnEndTurn()

{

if (EndTurnEvent != null)

{

List<AttackFleet> orderedFleets = \_fleetsToLaunch.Select(x => x as AttackFleet).ToList();

\_fleetsToLaunch.Clear();

EndTurnEvent(this, new EndTurnEventArgs(orderedFleets));

}

}

}

}

* ***Класс «ComputerPlayer»***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces;

using StarshipSeven.GameInterfaces.Strategy;

using System.Windows.Media;

namespace StarshipSeven.GameEntities

{

public class ComputerPlayer : Player, IComputerPlayer

{

private IComputerTurnStrategy \_strategy;

private bool \_autoEndTurnEnabled;

public ComputerPlayer(string name, Color color, IComputerTurnStrategy strategy, bool autoEndTurn)

:base(name, color)

{

Strategy = strategy;

AutoEndTurnEnabled = autoEndTurn;

}

public ComputerPlayer(Guid id)

: base(id)

{

}

public virtual void Think()

{

Strategy.MakeTurn(this);

if (AutoEndTurnEnabled)

EndTurn();

}

public IEnumerable<IPlanet> EnemyPlanets

{

get { return (Game.Map as Map).GetEnemyPlanetsForPlayer(this); }

}

public bool AutoEndTurnEnabled

{

get { return \_autoEndTurnEnabled; }

set { SetProperty<bool>(ref \_autoEndTurnEnabled, value, "AutoEndTurnEnabled"); }

}

public IComputerTurnStrategy Strategy

{

get { return \_strategy; }

set { SetProperty<IComputerTurnStrategy>(ref \_strategy, value, "Strategy"); }

}

}

}

* ***Класс «Planet»***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces;

using StarshipSeven.GameEntities.Exceptions;

using StarshipSeven.GameEntities.Events;

using StarshipSeven.GameInterfaces.Events;

namespace StarshipSeven.GameEntities

{

public class Planet : BaseGameEntity<Guid>, IPlanet, IComparable<Planet>

{

private string \_name;

private Position? \_position;

private double \_killPercentage;

private int \_productionRate;

private int \_ships;

private Player \_owner;

public Planet(Player owner) : base(Guid.NewGuid())

{

SetProperty<Player>(ref \_owner, owner, "Player");

}

public Planet(string name) : base(Guid.NewGuid())

{

Name = name;

}

public Planet(Guid id)

: base(id)

{

}

#region IPlanet Members

public string Name

{

get { return \_name; }

set { SetProperty<string>(ref \_name, value, "Name"); }

}

public Position? Position

{

get { return \_position; }

set { SetProperty<Nullable<Position>>(ref \_position, value, "Position"); }

}

public int GetDistanceTo(IPlanet planet)

{

if (Position == null || planet.Position == null)

return int.MaxValue;

int distance = Math.Abs(Position.Value.X - planet.Position.Value.X) + Math.Abs(Position.Value.Y - planet.Position.Value.Y);

return distance;

}

public int ProductionRate

{

get { return \_productionRate; }

set { SetProperty<int>(ref \_productionRate, value, "ProductionRate"); }

}

public int Ships

{

get { return \_ships; }

set { SetProperty<int>(ref \_ships, value, "ShipsCount"); }

}

public double KillPercentage

{

get { return \_killPercentage; }

set { SetProperty<double>(ref \_killPercentage, value, "KillPercentage"); }

}

public IPlayer Owner

{

get { return \_owner; }

}

#endregion

internal void SetOwner(Player owner)

{

SetProperty<Player>(ref \_owner, owner, "Owner");

}

internal void Produce()

{

Ships += ProductionRate;

(Owner.Statistics as GameStatistics).ShipsProduced += ProductionRate;

}

internal void MeetAttackFleet(AttackFleet fleet)

{

if (fleet.Owner.Equals(this.Owner))

OnReinforcementsArrived(fleet.Ships);

else

OnBattle(fleet, ProduceDefenseFleet());

}

internal void SetupAttackFleet(AttackFleet fleet)

{

fleet.Dead += OnAttackDeath;

}

internal AttackFleet ProduceAttackFleet(int ships, Planet destinationPlanet, Player requestingPlayer)

{

if (!requestingPlayer.Equals(Owner as Player))

throw new CheatingException(requestingPlayer);

if (Ships < ships)

throw new GameLogicException("Not enough ships to form a fleet!");

Ships -= ships;

AttackFleet fleet = new AttackFleet(requestingPlayer, ships, this, destinationPlanet);

SetupAttackFleet(fleet);

return fleet;

}

internal DefenseFleet ProduceDefenseFleet()

{

DefenseFleet defense = new DefenseFleet(this);

defense.Dead += OnDefenseDeath;

return defense;

}

protected void OnBattle(AttackFleet attack, DefenseFleet defense)

{

if (Battle != null)

Battle(this, new BattleEventArgs(attack, defense));

}

private void OnAttackDeath(object sender, FleetDeadEventArgs args)

{

if (args.KillerFleet == null)

throw new CheatingException();

AttackFleet attack = (AttackFleet)args.DeadFleet;

DefenseFleet defense = (DefenseFleet)args.KillerFleet;

(attack.Owner.Statistics as GameStatistics).FleetsLost += 1;

(defense.Owner.Statistics as GameStatistics).EnemyFleetsDestroyed += 1;

defense.HomePlanet.Ships += defense.Ships;

OnDefensed(attack.Owner, defense.Owner, defense.HomePlanet);

}

private void OnDefenseDeath(object sender, FleetDeadEventArgs args)

{

if (args.KillerFleet == null)

throw new CheatingException();

AttackFleet attack = (AttackFleet) args.KillerFleet;

DefenseFleet defense = (DefenseFleet)args.DeadFleet;

(attack.Owner.Statistics as GameStatistics).EnemyFleetsDestroyed += 1;

(defense.Owner.Statistics as GameStatistics).FleetsLost += 1;

SetOwner(attack.Owner);

Ships += attack.Ships;

OnConquered(attack.Owner, defense.Owner);

}

internal event EventHandler<PlanetChangedEventArgs> Conquered;

internal event EventHandler<PlanetChangedEventArgs> Defensed;

internal event EventHandler<BattleEventArgs> Battle;

internal event EventHandler<ReinforcementsArrivedEventArgs> ReinforcementsArrived;

protected void OnReinforcementsArrived(int ships)

{

Ships += ships;

if (ReinforcementsArrived != null)

ReinforcementsArrived(this, new ReinforcementsArrivedEventArgs(this, ships));

}

protected void OnConquered(IPlayer attackingPlayer, IPlayer defendingPlayer)

{

if (Conquered != null)

Conquered(this, new PlanetChangedEventArgs(this, attackingPlayer, defendingPlayer));

}

protected void OnDefensed(IPlayer attackingPlayer, IPlayer defendingPlayer, IPlanet defensePlanet)

{

if (Defensed != null)

Defensed(this, new PlanetChangedEventArgs(defensePlanet, attackingPlayer, defendingPlayer));

}

public int CompareTo(Planet other)

{

int ydelta = Position.Value.Y - other.Position.Value.Y;

if (ydelta != 0)

return ydelta;

int xdelta = Position.Value.X - other.Position.Value.X;

if (xdelta != 0)

return xdelta;

else

return 0;

}

}

}

* ***Класс «Fleet»***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces;

using StarshipSeven.GameEntities.Events;

using StarshipSeven.GameEntities.Exceptions;

namespace StarshipSeven.GameEntities

{

public abstract class Fleet : BaseGameEntity<Guid>, IFleet

{

private int \_ships;

private Player \_owner;

public Fleet(int shipsCount, Player owner) : base(Guid.NewGuid())

{

Ships = shipsCount;

SetProperty<Player>(ref \_owner, owner, "Owner");

}

#region IFleet Members

public abstract double KillPercentage { get; }

public void Attack(IFleet enemyFleet)

{

if (!(enemyFleet is Fleet))

throw new TypeMismatchException(typeof(Fleet), enemyFleet);

Fleet enemy = (Fleet)enemyFleet;

if (!enemy.IsDead)

{

(Owner.Statistics as GameStatistics).EnemyShipsDestroyed += 1;

enemy.SetShipsCount(enemy.Ships - 1, this);

}

}

public bool IsDead

{

get { return Ships == 0; }

}

public int Ships

{

get { return \_ships; }

set { SetShipsCount(value, null); }

}

#endregion

internal Player Owner

{

get { return \_owner; }

}

internal event EventHandler<FleetDeadEventArgs> Dead;

protected void OnDead(Fleet killer)

{

if (Dead != null)

Dead(this, new FleetDeadEventArgs(this, killer));

}

internal void SetShipsCount(int count, Fleet possibleKiller)

{

int value = (count < 0) ? 0 : count;

SetProperty<int>(ref \_ships, value, "ShipsCount");

if (value == 0)

OnDead(possibleKiller);

}

}

}

* ***Класс «AttackFleet»***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces;

namespace StarshipSeven.GameEntities

{

public class AttackFleet : Fleet, IAttackFleet

{

private Planet \_sourcePlanet;

private Planet \_destinationPlanet;

private int? \_arrivalTurn;

public AttackFleet(Player owner, int shipsCount, Planet sourcePlanet, Planet destinationPlanet)

:base(shipsCount, owner)

{

SetProperty<Planet>(ref \_sourcePlanet, sourcePlanet, "SourcePlanet");

SetProperty<Planet>(ref \_destinationPlanet, destinationPlanet, "DestinationPlanet");

}

#region IAttackFleet Members

public IPlanet SourcePlanet

{

get { return \_sourcePlanet; }

}

public IPlanet DestinationPlanet

{

get { return \_destinationPlanet; }

}

public int? ArrivalTurn

{

get { return \_arrivalTurn; }

set { SetProperty<Nullable<int>>(ref \_arrivalTurn, value, "ArrivalTurn"); }

}

#endregion

public override double KillPercentage

{

get { return SourcePlanet.KillPercentage; }

}

}

}

* ***Класс «Game»***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections.ObjectModel;

using StarshipSeven.GameInterfaces;

using StarshipSeven.GameInterfaces.Strategy;

using StarshipSeven.GameInterfaces.Events;

using StarshipSeven.GameEntities.Events;

using StarshipSeven.GameEntities.Exceptions;

namespace StarshipSeven.GameEntities

{

public class Game : BaseGameEntity<Guid>, IGame

{

private int \_currentTurn;

private bool \_isGameOver = false;

private Player \_currentPlayer;

private Map \_map;

private List<Player> \_players = new List<Player>();

private List<Player> \_deadPlayers = new List<Player>();

private List<AttackFleet> \_movingFleets = new List<AttackFleet>();

private Queue<Player> \_waitingPlayers = new Queue<Player>();

private IBattleHandler \_battlerHandler;

#region IGame Members

public Game() : base(Guid.NewGuid())

{

}

public Game(Map map) : base(Guid.NewGuid())

{

SetProperty<Map>(ref \_map, SetupMap(map), "Map");

}

public IPlayer CurrentPlayer

{

get { return \_currentPlayer; }

}

public int CurrentTurn

{

get { return \_currentTurn; }

private set { SetProperty<int>(ref \_currentTurn, value, "CurrentTurn"); }

}

public void Start()

{

if (\_currentPlayer is ComputerPlayer)

(\_currentPlayer as ComputerPlayer).Think();

}

public IMap Map

{

get { return \_map; }

}

public IEnumerable<IAttackFleet> MovingFleetsForCurrentPlayer

{

get { return \_movingFleets.Where(x => x.Owner.Equals(\_currentPlayer)); }

}

public ObservableCollection<IAttackFleet> FleetsToLaunchForCurrentPlayer

{

get { return \_currentPlayer.FleetsToLaunch; }

}

public IBattleHandler BattleHandler

{

get { return \_battlerHandler; }

set { SetProperty<IBattleHandler>(ref \_battlerHandler, value, "BattleHandler"); }

}

public event EventHandler BeforeTurnProcess;

public event EventHandler<GameInterfaces.Events.PlanetChangedEventArgs> PlanetConquered;

public event EventHandler<GameInterfaces.Events.PlanetChangedEventArgs> PlanetDefended;

public event EventHandler<PlayerChangedEventArgs> PlayerDead;

public event EventHandler<PlayerChangedEventArgs> PlayerDisqualified;

public event EventHandler<GameOverEventArgs> GameOver;

public event EventHandler<ReinforcementsArrivedEventArgs> ReinforcementsArrived;

#endregion

internal Game(int turn, Player currentPlayer, IEnumerable<Player> players, Map map, IEnumerable<AttackFleet> attackingFleets)

{

CurrentTurn = turn;

SetMap(map);

foreach (var player in players)

{

AttachPlayer(player);

if (IsPlayerDead(player))

{

\_deadPlayers.Add(player);

\_players.Remove(player);

}

}

\_movingFleets = new List<AttackFleet>(attackingFleets);

foreach(var player in \_players.SkipWhile(x => !x.Equals(currentPlayer)).Skip(1))

\_waitingPlayers.Enqueue(player);

SetProperty<Player>(ref \_currentPlayer, currentPlayer, "CurrentPlayer");

}

internal void Initialize()

{

CurrentTurn = 1;

foreach (Player player in \_players)

\_waitingPlayers.Enqueue(player);

\_map.ProcessNextTurn();

SetProperty<Player>(ref \_currentPlayer, \_waitingPlayers.Dequeue(), "CurrentPlayer");

}

internal void SetMap(Map map)

{

if (\_map != null)

ReleaseMap(\_map);

SetProperty<Map>(ref \_map, SetupMap(map), "Map");

}

private void OnBeforeTurnProcess()

{

if (BeforeTurnProcess != null)

BeforeTurnProcess(this, new EventArgs());

}

private void OnPlanetConquered(object sender, PlanetChangedEventArgs args)

{

if (PlanetConquered != null)

PlanetConquered(this, args);

}

private void OnPlanetDefended(object sender, PlanetChangedEventArgs args)

{

if (PlanetDefended != null)

PlanetDefended(this, args);

}

private void OnReinforcementsArrived(object sender, ReinforcementsArrivedEventArgs args)

{

if (ReinforcementsArrived != null)

ReinforcementsArrived(this, args);

}

private Map SetupMap(Map map)

{

foreach (Planet p in map)

{

p.Conquered += OnPlanetConquered;

p.Defensed += OnPlanetDefended;

p.ReinforcementsArrived += OnReinforcementsArrived;

p.Battle += OnBattle;

}

return map;

}

private Map ReleaseMap(Map map)

{

foreach (Planet p in map)

{

p.Conquered -= OnPlanetConquered;

p.Defensed -= OnPlanetDefended;

p.ReinforcementsArrived -= OnReinforcementsArrived;

p.Battle -= OnBattle;

}

return map;

}

private bool IsPlayerDead(Player player)

{

if (!player.ConqueredPlanets.Any() && !\_movingFleets.Any(x => x.Owner.Equals(player)))

return true;

return false;

}

protected void OnEndTurn(object sender, EndTurnEventArgs args)

{

if (\_isGameOver)

return;

foreach (AttackFleet fleet in args.LaunchedFleets)

AddToMoving(fleet);

if (\_waitingPlayers.Count == 0)

ProcessNextTurn();

else

SetProperty<Player>(ref \_currentPlayer, \_waitingPlayers.Dequeue(), "CurrentPlayer");

if (\_currentPlayer is ComputerPlayer && !\_isGameOver)

{

try

{

(\_currentPlayer as ComputerPlayer).Think();

}

catch (CheatingException ex)

{

OnDisqualify(ex.Cheater);

}

}

OnPropertyChanged("MovingFleetsForCurrentPlayer");

OnPropertyChanged("Game");

}

internal IEnumerable<Player> AllPlayers

{

get { return \_players.Union(\_deadPlayers); }

}

internal IEnumerable<AttackFleet> MovingFleets

{

get { return \_movingFleets; }

}

private void OnBattle(object sender, BattleEventArgs args)

{

if (args.AttackFleet.Ships == 0)

throw new Exception();

BattleHandler.HandleBattle(args.AttackFleet, args.DefenseFleet);

}

private void OnDisqualify(Player cheater)

{

if (cheater == null)

throw new GameLogicException("Cheating detected!");

var neutralPlayer = \_players.SingleOrDefault(x => x is NeutralPlayer);

if (neutralPlayer == null)

{

neutralPlayer = \_deadPlayers.SingleOrDefault(x => x is NeutralPlayer);

\_deadPlayers.Remove(neutralPlayer);

\_players.Add(neutralPlayer);

}

foreach (var planet in cheater.ConqueredPlanets)

(planet as Planet).SetOwner(neutralPlayer);

\_players.Remove(cheater);

\_deadPlayers.Add(cheater);

(cheater.Statistics as GameStatistics).DeathTurn = CurrentTurn;

if (PlayerDisqualified != null)

PlayerDisqualified(this, new PlayerChangedEventArgs(cheater));

}

private void ProcessNextTurn()

{

CurrentTurn += 1;

OnBeforeTurnProcess();

\_map.ProcessNextTurn();

Func<AttackFleet, bool> arrivesThisTurn = f => f.ArrivalTurn.Value == CurrentTurn;

foreach (AttackFleet fleet in \_movingFleets.Where(x => arrivesThisTurn(x)))

(fleet.DestinationPlanet as Planet).MeetAttackFleet(fleet);

\_movingFleets.RemoveAll(x => arrivesThisTurn(x));

foreach (Player player in \_players.ToList())

{

if (IsPlayerDead(player))

OnPlayerDead(player);

else

\_waitingPlayers.Enqueue(player);

}

if (!\_isGameOver)

SetProperty<Player>(ref \_currentPlayer, \_waitingPlayers.Dequeue(), "CurrentPlayer");

}

private void OnPlayerDead(Player player)

{

\_players.Remove(player);

\_deadPlayers.Add(player);

(player.Statistics as GameStatistics).DeathTurn = CurrentTurn;

if (PlayerDead != null)

PlayerDead(this, new PlayerChangedEventArgs(player));

if (\_players.Count == 2 && \_players.Any(x => x is NeutralPlayer))

OnGameOver(\_players.First(x => !(x is INeutralPlayer)));

}

private void OnGameOver(Player winner)

{

\_isGameOver = true;

if (GameOver != null)

GameOver(this, new GameOverEventArgs(\_players.Union(\_deadPlayers), winner));

}

private void AddToMoving(AttackFleet fleet)

{

int arrivalTurn = CurrentTurn + fleet.SourcePlanet.GetDistanceTo(fleet.DestinationPlanet);

fleet.ArrivalTurn = arrivalTurn;

\_movingFleets.Add(fleet);

}

public void AttachPlayer(Player player)

{

if (\_players.Contains(player))

throw new ItemDuplicateException(player, this);

player.EndTurnEvent += OnEndTurn;

player.Game = this;

\_players.Add(player);

}

}

}

## Механизм добавления стратегий

Разработчику искусственного интеллекта для игры «Starship Seven» поставляется сборка StarshipSeven.GameInterfaces.dll с основными интерфейсами и атрибутами. Необходимо наследоваться от интерфейса IComputerTurnStrategy и реализовать метод MakeTurn. Его сигнатура имеет следующий вид:

void MakeTurn(IComputerPlayer player);

Вызывая методы интерфейса IComputerPlayer – LaunchFleet и CancelLaunch, необходимо выбрать куда и сколько кораблей отправить.

Список вражеских планет можно узнать из свойства EnemyPlanets, список своих – ConqeredPlanets.

К сборке нужно применить атрибут StrategySetAttribute, в котором дать имя стратегии и указать автора стратегии. В сборке может находиться несколько классов – стратегий. К каждому такому классу нужно применить атрибут StrategyAttribute, в котором указывается уровень сложности (Easy, Medium, Hard). Также можно указать строку – тэг.

Скомпилированную сборку со стратегиями нужно поместить в папку AI, которая находится в каталоге с игрой. После этого должна появиться возможность выбора этой стратегии для компьютерного соперника в игре.

***Пример (стандартные стратегии):***

* *Базовый класс*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces.Strategy;

using StarshipSeven.GameInterfaces;

using StarshipSeven.GameInterfaces.Strategy.Attributes;

[assembly: StrategySet("Default AI", Author = "K. Smolyakov")]

namespace StarshipSeven.DefaultAI

{

public abstract class BaseAI : IComputerTurnStrategy

{

private IComputerPlayer \_player;

public void MakeTurn(IComputerPlayer player)

{

\_player = player;

foreach (IPlanet myPlanet in \_player.ConqueredPlanets)

{

var attackTarget = ChooseTargetPlanet(myPlanet);

if (attackTarget == null)

{

var reinforcementsTarget = ChoosePlanetForReinforcements(myPlanet);

if (reinforcementsTarget != null)

{

int shipsToLaunch = CanLaunchForReinforcements(myPlanet);

if (shipsToLaunch > 0)

player.LaunchFleet(shipsToLaunch, myPlanet, reinforcementsTarget);

}

}

else

{

int shipsToLaunch = CanLaunch(myPlanet);

if (shipsToLaunch > 0)

player.LaunchFleet(shipsToLaunch, myPlanet, attackTarget);

}

}

}

private IPlanet ChooseTargetPlanet(IPlanet myPlanet)

{

var targets = GetVisibleEnemyPlanets(myPlanet)

.Where(x => CanWin(myPlanet, x))

.OrderBy(x => myPlanet.GetDistanceTo(x));

var neutral = targets.FirstOrDefault(x => x.Owner is INeutralPlayer);

return (neutral == null) ? targets.FirstOrDefault() : neutral;

}

private IPlanet ChoosePlanetForReinforcements(IPlanet myPlanet)

{

var target = GetVisibleConqueredPlanets(myPlanet).OrderByDescending(x => x.KillPercentage).FirstOrDefault();

if (target == null)

return null;

return (target.KillPercentage < myPlanet.KillPercentage) ? null : target;

}

private int EnemyPlanetsCount

{

get { return \_player.EnemyPlanets.Count(); }

}

private IEnumerable<IPlanet> GetVisibleConqueredPlanets(IPlanet myPlanet)

{

int visiblePlanets = Convert.ToInt32(EnemyPlanetsCount \* VisionFactor);

return \_player.ConqueredPlanets.Where(x => !x.Equals(myPlanet)).OrderBy(x => myPlanet.GetDistanceTo(x)).Take(visiblePlanets);

}

private IEnumerable<IPlanet> GetVisibleEnemyPlanets(IPlanet myPlanet)

{

int visiblePlanets = Convert.ToInt32(EnemyPlanetsCount \* VisionFactor);

return \_player.EnemyPlanets.OrderBy(x => myPlanet.GetDistanceTo(x)).Take(visiblePlanets);

}

private bool CanWin(IPlanet sourcePlanet, IPlanet enemyPlanet)

{

int distance = sourcePlanet.GetDistanceTo(enemyPlanet);

return GetMyPower(sourcePlanet) / ExpectedEnemyPower(enemyPlanet, distance) >= StupidityFactor;

}

private double GetMyPower(IPlanet sourcePlanet)

{

return CanLaunch(sourcePlanet) \* sourcePlanet.KillPercentage;

}

private double ExpectedEnemyPower(IPlanet planet, int distance)

{

return (planet.Ships + distance \* planet.ProductionRate) \* planet.KillPercentage;

}

private int CanLaunch(IPlanet planet)

{

return Convert.ToInt32(planet.Ships \* RiskFactor);

}

private int CanLaunchForReinforcements(IPlanet planet)

{

return Convert.ToInt32(planet.Ships \* ReinforcementsFactor);

}

protected abstract double StupidityFactor { get; }

protected abstract double RiskFactor { get; }

protected abstract double VisionFactor { get; }

protected abstract double ReinforcementsFactor { get; }

}

}

* Уровень Easy

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces.Strategy.Attributes;

namespace StarshipSeven.DefaultAI

{

[Strategy(DifficultyLevel.Easy)]

public class EasyAI : BaseAI

{

protected override double RiskFactor

{

get { return 0.8; }

}

protected override double VisionFactor

{

get { return 0.3; }

}

protected override double ReinforcementsFactor

{

get { return 0.1; }

}

protected override double StupidityFactor

{

get { return 0.2; }

}

}

}

* Уровень Medium

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces.Strategy.Attributes;

namespace StarshipSeven.DefaultAI

{

[Strategy(DifficultyLevel.Medium)]

public class MediumAI : BaseAI

{

protected override double RiskFactor

{

get { return 0.8; }

}

protected override double VisionFactor

{

get { return 0.7; }

}

protected override double ReinforcementsFactor

{

get { return 0.6; }

}

protected override double StupidityFactor

{

get { return 0.5; }

}

}

}

* Уровень Hard

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces.Strategy.Attributes;

namespace StarshipSeven.DefaultAI

{

[Strategy(DifficultyLevel.Hard)]

public class HardAI : BaseAI

{

protected override double RiskFactor

{

get { return 0.7; }

}

protected override double VisionFactor

{

get { return 1; }

}

protected override double ReinforcementsFactor

{

get { return 0.9; }

}

protected override double StupidityFactor

{

get { return 1; }

}

}

}

## Реализация отображения карты в игровом окне

Для отображения карты используется элемент управления ListBox с установленным UniformGrid в качестве ItemPanel. Источником данных для ListBox является коллекция, в которой содержатся объекты вспомогательного класса MapTile. Приведем здесь его код:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using StarshipSeven.GameInterfaces;

using System.ComponentModel;

namespace StarshipSeven.ViewModel.Wrappers

{

public class MapTile : BaseViewModel

{

private IPlanet \_planet;

private bool \_isSourcePlanet;

private bool \_isDestinationPlanet;

public MapTile(IPlanet planet)

{

Planet = planet;

}

public IPlanet Planet

{

get { return \_planet; }

set

{

if (value != null)

value.PropertyChanged += OnPlanetChanged;

SetProperty<IPlanet>(ref \_planet, value, "Planet");

OnPropertyChanged("IsEmpty");

}

}

public void OnPlanetChanged(object sender, PropertyChangedEventArgs args)

{

OnPropertyChanged("Planet");

}

public bool IsSourcePlanet

{

get { return \_isSourcePlanet; }

set

{

SetProperty<bool>(ref \_isSourcePlanet, value, "IsSourcePlanet");

if (IsSourcePlanet && IsDestinationPlanet)

{

\_isDestinationPlanet = false;

OnPropertyChanged("IsDestinationPlanet");

}

}

}

public bool IsDestinationPlanet

{

get { return \_isDestinationPlanet; }

set

{

SetProperty<bool>(ref \_isDestinationPlanet, value, "IsDestinationPlanet");

if (IsSourcePlanet && IsDestinationPlanet)

{

\_isSourcePlanet = false;

OnPropertyChanged("IsSourcePlanet");

}

}

}

public bool IsEmpty

{

get { return Planet == null; }

}

public bool HasPlanet

{

get { return !IsEmpty; }

}

}

}

Используются различные стили и шаблоны для ячеек: ItemTemplate и ItemContainerStyle. Привожу здесь разметку XAML для шаблона данных MapItemTemplate. Триггеры в данном шаблоне отвечают за отображение ячейки как исходной или как планеты назначения.

<DataTemplate x:Key="MapItemTemplate">

<Grid>

<Grid.Background>

<SolidColorBrush Color="{Binding Planet.Owner.Color}" Opacity="0.3"/>

</Grid.Background>

<Border Name="mapItemBorder" BorderBrush="{StaticResource ButtonBorderColor}">

<Grid Margin="4,2" Name="itemGrid">

<TextBlock Text="{Binding Planet.Name}"

Style="{DynamicResource PlanetNameTextBoxStyle}"/>

</Grid>

</Border>

</Grid>

<DataTemplate.Triggers>

<DataTrigger Binding="{Binding IsSourcePlanet}" Value="true">

<Setter Property="BorderThickness" TargetName="mapItemBorder" Value="2"/>

<Setter Property="Background" TargetName="itemGrid">

<Setter.Value>

<ImageBrush ImageSource="/StarshipSeven.Presentation;component/Images/starship.png" Stretch="Uniform"/>

</Setter.Value>

</Setter>

</DataTrigger>

<DataTrigger Binding="{Binding IsDestinationPlanet}" Value="true">

<Setter Property="BorderThickness" TargetName="mapItemBorder" Value="2"/>

<Setter Property="BorderBrush" TargetName="mapItemBorder" Value="{DynamicResource DefaultBorderColor}"/>

<Setter Property="Background" TargetName="itemGrid">

<Setter.Value>

<ImageBrush ImageSource="/StarshipSeven.Presentation;component/Images/sight.png" Stretch="Uniform"/>

</Setter.Value>

</Setter>

</DataTrigger>

</DataTemplate.Triggers>

</DataTemplate>

Также используются другие шаблоны и стили, которые из-за громоздкости кода приводить здесь не буду.

## Позднее связывание и инверсия управления

Инверсия управления (Inversion of Control, IoC) — важный принцип объектно-ориентированного программирования, используемый для уменьшения связанности в компьютерных программах.

Одной из реализаций IoC является внедрение зависимостей (англ. Dependency Injection). Dependency Injection используется во многих framework'ах, которые называются IoC контейнерами.

В моем проекте используется Unity IoC Framework - проект с открытым исходным кодом от группы **Patterns & Practices** в Microsoft.

Далее приведу пример использования данной технологии. Напомним интерфейс IBattleHandler

namespace StarshipSeven.GameInterfaces.Strategy

{

public interface IBattleHandler

{

void HandleBattle(IAttackFleet attackFleet, IDefenseFleet defenseFleet);

}

}

Для создания игры фабрикой необходим объект класса, реализующий данный интерфейс. Unity считывает информацию о типе класса, которым необходимо проинициализировать свойство BattleHandler фабрики GameFactory. Такая информация хранится в конфигурационном файле приложения. Далее приведен конфигурационный файл, жирным выделена строка, которой по сути осуществляется связь интерфейса IBattleHandler с классом DefaultBattleHandler.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="unity" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection, Microsoft.Practices.Unity.Configuration"/>

<section name="strategyManager" type="StarshipSeven.StrategyManager.Configuration.StrategyManagerConfiguration, StarshipSeven.StrategyManager"/>

<section name="savegameManager" type="StarshipSeven.SavegameManager.Configuration.SavegameManagerConfiguration, StarshipSeven.SavegameManager"/>

</configSections>

<unity xmlns="http://schemas.microsoft.com/practices/2010/unity">

<container>

<register type="StarshipSeven.GameInterfaces.Factories.IGameFactory, StarshipSeven.GameInterfaces" mapTo="StarshipSeven.GameEntities.Factories.GameFactory, StarshipSeven.GameEntities"/>

**<register type="StarshipSeven.GameInterfaces.Strategy.IBattleHandler, StarshipSeven.GameInterfaces" mapTo="StarshipSeven.DefaultBattleHandler.DefaultBattleHandler, StarshipSeven.DefaultBattleHandler"/>**

<register type="StarshipSeven.DataInterfaces.Managers.IStrategyManager, StarshipSeven.DataInterfaces" mapTo="StarshipSeven.StrategyManager.NativeStrategyManager, StarshipSeven.StrategyManager"/>

<register type="StarshipSeven.DataInterfaces.Managers.ISavegameManager, StarshipSeven.DataInterfaces" mapTo="StarshipSeven.SavegameManager.NativeSavegameManager, StarshipSeven.SavegameManager"/>

<register type="StarshipSeven.DataInterfaces.Managers.IGameStateManager, StarshipSeven.DataInterfaces" mapTo="StarshipSeven.GameEntities.Managers.GameStateManager, StarshipSeven.GameEntities"/>

</container>

</unity>

<strategyManager assemblySource="AI"/>

<savegameManager savegameCatalog="Saves"/>

</configuration>

Связь эта настраивается в классе GameConstructorModel. Приведу здесь конструктор этого класса.

public GameConstructorModel()

{

Players = new ObservableCollection<PlayerWrapper>();

Players.CollectionChanged += OnPlayersCollectionChanged;

**\_container = new UnityContainer().LoadConfiguration();**

\_factory = \_container.Resolve<IGameFactory>();

**\_factory.BattleHandler = \_container.Resolve<IBattleHandler>();**

\_factory.PropertyChanged += OnFactoryPropertyChanged;

\_strategyManager = \_container.Resolve<IStrategyManager>().LoadConfiguration();

MapWrapper = new MapWrapper(\_factory.CurrentMap);

AddDefaultPlayer();

}

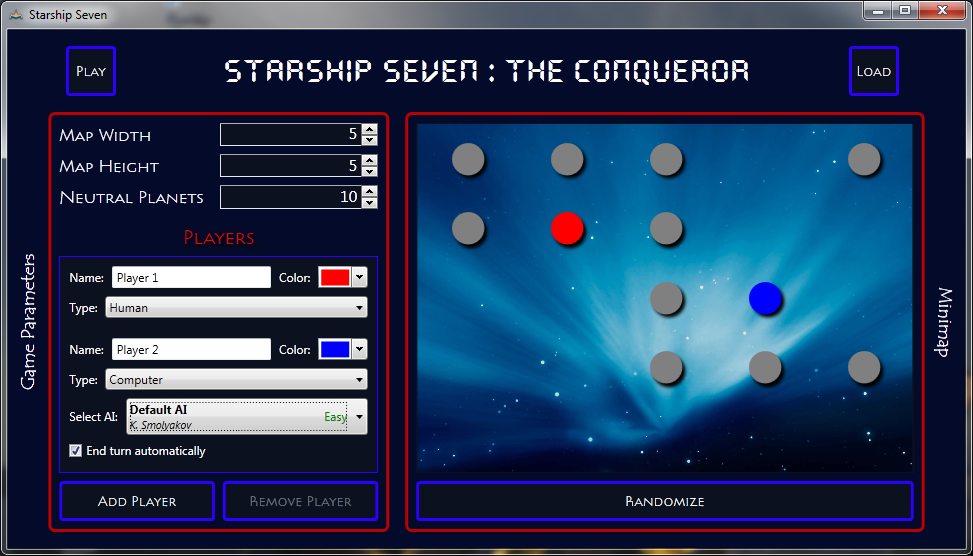
Жирным выделена загрузка конфигурации контейнера Unity, а также инициализация обработчика битв фабрики.

Таким образом, правила игры можно легко поменять, если написать новый обработчик битв и поменять строчку в конфигурационном файле.

# Игровой процесс

После запуска файла StarshipSeven.Presentation.exe видим окно конструктора игры. Здесь можно начать новую игру, задав ширину и высоту карты, а также добавив необходимое количество соперников. Можно также перейти к окну загрузки сохраненной игры.

*Первое окно – «конструктор игры»*



После нажатия на кнопку «Play» переходим к основному игровому окну.

*Основное игровое окно*



Здесь отображается информация о текущем игроке, текущем ходе. Нажимая на планеты на карте можно увидеть информацию о них в секции «Control Panel». Фон сектора соответствует игроку – владельцу планеты.

Нажав по принадлежащей Вам планете два раза левой кнопкой мыши можно установить ее как исходную. Планета назначения устанавливается правой кнопкой мыши. Также эти функции дублируются в панели управления (есть кнопки рядом с информацией о планете). Затем игрок устанавливает количество кораблей и нажимает на красную кнопку «Push» До нажатия кнопки «End Turn» любой запуск можно отменить. Сделав все необходимые манипуляции, игрок нажимает кнопку «End Turn» и инициатива переходит к другому игроку.

В левой нижней зоне игрового экрана отображается информация о событиях, произошедших в игре. Причем первыми отображаются события, которые относятся к текущему игроку.

В левом верхнем углу – кнопка «Меню»

*Игровое меню*



Здесь отображается список сохраненных игр, можно сохранить текущую игру или загрузить старую. Доступен предпросмотр карты для сохраненных игр. Можно также удалять сохраненные игры.

После того, как один из игроков выиграл, отображается окно со статистикой.

*Окно статистики.*



# Заключение

В ходе работы над курсовым проектом были закреплены навыки программирования на языке C#. Изучены основные методы работы с технологией WPF в связке с шаблоном MVVM. Обеспечена необходимая расширяемость программного продукта. Игра получилась довольно простой и понятной, что и требуется для рядовой казуальной игры.

Для повышения эффектности можно было бы добавить звуковое сопровождение. Также можно провести рефакторинг кода, использовать в некоторых ситуациях более подходящие структуры данных для повышения производительности. Тогда данный код можно было бы использовать для просчета карт больших размеров с огромным количеством планет (например, один миллион).

В целом игра неплохо смотрится на фоне своих конкурентов.

# Список использованной литературы

1. А.А. Волосевич «Базовые технологии платформы .NET». Минск, 2012 г.
2. А.А. Волосевич «Технология Windows Presentation Foundation». Минск, 2012 г.
3. Мэтью Мак-Дональд «WPF. Windows Presentation Foundation в .NET 4 с примерами на С# 2010». Москва, 2011 г.
4. Gary McLean Hall «Pro WPF and Silverlight MVVM Effective Application Development with Model-View-ViewModel» Apress, 2010
5. Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford «Patterns of Enterprise Application Architecture» Publisher: Addison Wesley, 2002
6. Веб-ресурс msdn.microsoft.com