Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего профессионального образования

«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет . информационных технологий

Кафедра . прикладная математика .

Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_

Руководитель

проекта (работы) \_\_\_\_\_\_\_ Старолетов М.С.\_\_\_\_\_\_

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2013 г.

**Реализация многопоточного симулятора стратегической игры**

Пояснительная записка

к курсовой работе

по дисциплине Проектирование сетевых и многопоточных приложений

**КР 230105.20.000ПЗ**

Студент группы \_\_\_\_\_\_\_ПОВТ-81\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Яковлев В.А.\_\_

Руководитель

проекта (работы) \_\_ \_\_\_\_\_\_\_\_\_\_ Старолетов С.М.\_\_

Барнаул 2013

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Задание на курсовую работу**  Учебная дисциплина: Проектирование сетевых и многопоточных приложений  ФИО студента: Яковлев Василий Андреевич  Группа: ПОВТ-81  Задание: Реализация многопоточного симулятора стратегической игры.  Руководитель: Старолетов С. М. | | | | | | | | | | |
|  |  |  |  |  | **КР 230105.20.000ПЗ** | | | | | |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подпись | Дата |
| Разработал | | Яковлев В.А. |  |  | Реализация многопоточного симулятора стратегической игры | Лит. | | | Лист | Листов |
| Пров. | | Старолетов С.М. |  |  |  | У |  | 2 | 31 |
|  | |  |  |  | АлтГТУ, ФИТ  *гр. ПОВТ-81* | | | | |
| Н.контролер | |  |  |  |
| Утв. | | Кантор С.А. |  |  |

**Аннотация**

В данной работе представлены сведения о проектировании и разработке приложения "Многопоточный симулятор стратегической игры". Приложение позволяет моделировать противостояние двух игроков, а именно постройка баз, борьба за ресурсы и симуляция битв.

Программное обеспечение написано на языке программирования С# с использованием Microsoft Visual Studio 2012 и XNA 4.0. Исходный код приложения, а также данный отчет представлены на прилагаемом CD диске.

**Содержание**

[Введение 7](#_Toc345686078)

[1 Постановка задачи 8](#_Toc345686079)

[1.1 Развернутая постановка задачи 8](#_Toc345686080)

[2 Проектирование программного обеспечения 9](#_Toc345686081)

[2.1 Диаграмма вариантов использования 9](#_Toc345686082)

[2.2 Диаграмма развертывания 9](#_Toc345686083)

[2.3 Временная диаграмма 10](#_Toc345686084)

[2.4 Диаграмма деятельности 10](#_Toc345686085)

[3 Описание программного обеспечения 12](#_Toc345686086)

[3.1 Общий подход к реализации 12](#_Toc345686087)

[3.2 Описание приложения 13](#_Toc345686088)

[Заключение 17](#_Toc345686089)

[Список использованных источников 18](#_Toc345686090)

[Приложение А Руководство пользователя 19](#_Toc345686091)

[Приложение Б Текст программы 21](#_Toc345686093)

**Введение**

В современном мире технические и технологические достижения стали неотъемлемой частью жизни. Один из важнейших показателей прогресса - это внедрение информационных технологий**,** которое повлекло за собой переход общества в новую фазу развития – так называемое «информационное общество». Развитие информационных технологий явилось результатом широкого использования компьютерной техники в разных сферах человеческой жизни.

Компьютерные технологии дают нам огромное количество преимуществ. Они удовлетворяет наши потребности, облегчают наш труд и повышают его эффективность. Компьютерная техника рационализирует деятельность человека, организует её в соответствии с целесообразностью. Она расширяет доступ к информации. Компьютеры освобождают нас от утомительной работы, выполняя с высокой скоростью разнообразные операции. Скорость эта постоянно увеличивается.

Сфера применения компьютерных технологий очень широка и практически не ограниченна. В частности использование компьютеров, повлекло за собой бурное развитие индустрии компьютерных игр, одним из направлений которой были стратегии реального времени. Эти игры могут сильно отличатся друг от друга, но обычно суть остается неизменной. В таких приложениях некоторое количество игроков пытается уничтожить все юниты (объекты) противника.

# 1 Постановка задачи

## 1.1 Развернутая постановка задачи

Спроектировать и реализовать многопоточный симулятор стратегической игры.

Сделать AI для управлением игроком в игре наподобие Duna, Age of Emperies, Starcraft. Строятся базы, юниты, добываются ресурсы. Атака на другого игрока. Все сущности в игре - потоки. Реализовать графическое отображение хода компании.

# 2 Проектирование программного обеспечения

## 2.1 Диаграмма вариантов использования

На рисунке 2.1 представлена диаграмма использования для нашего приложения.

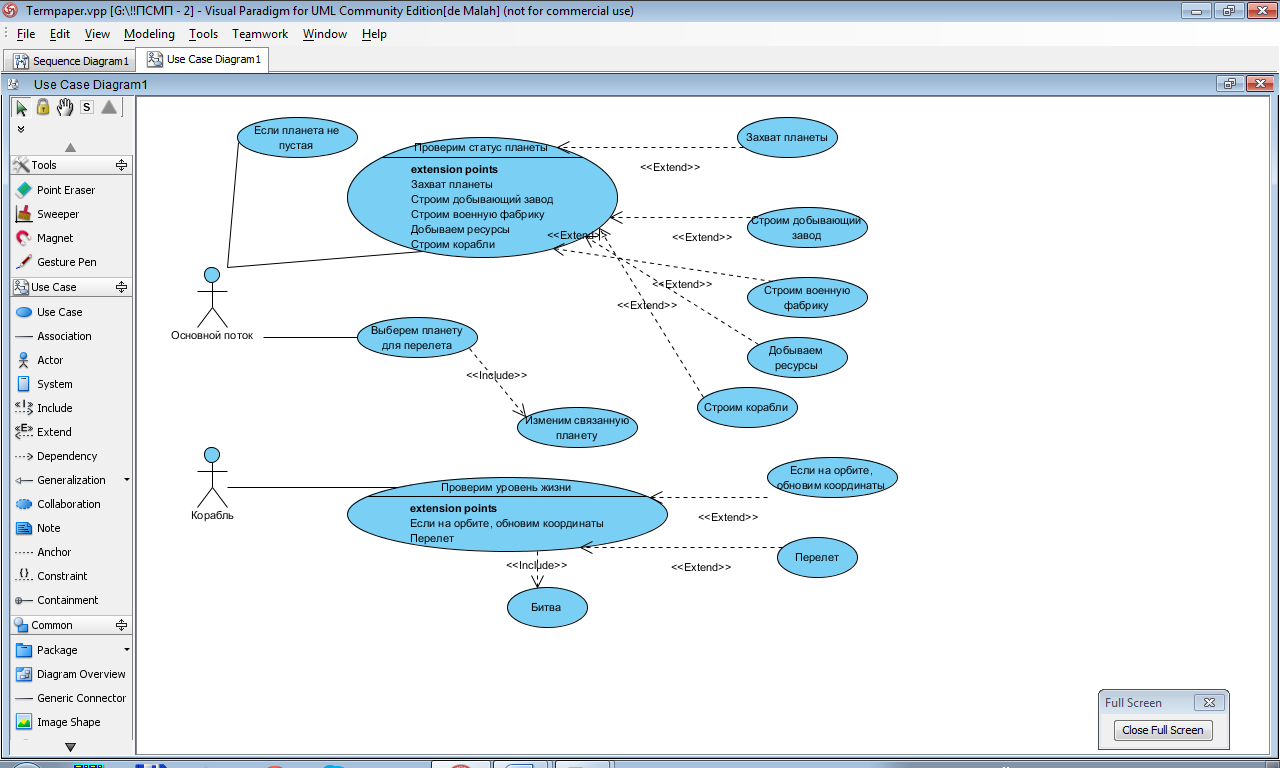


Рисунок 2.1 - Диаграмма вариантов использования

## 2.2 Диаграмма развертывания

На рисунке 2.2 представлена диаграмма развертывания приложения. В ней отраженны необходимые файлы и библиотеки для функционирования программы.

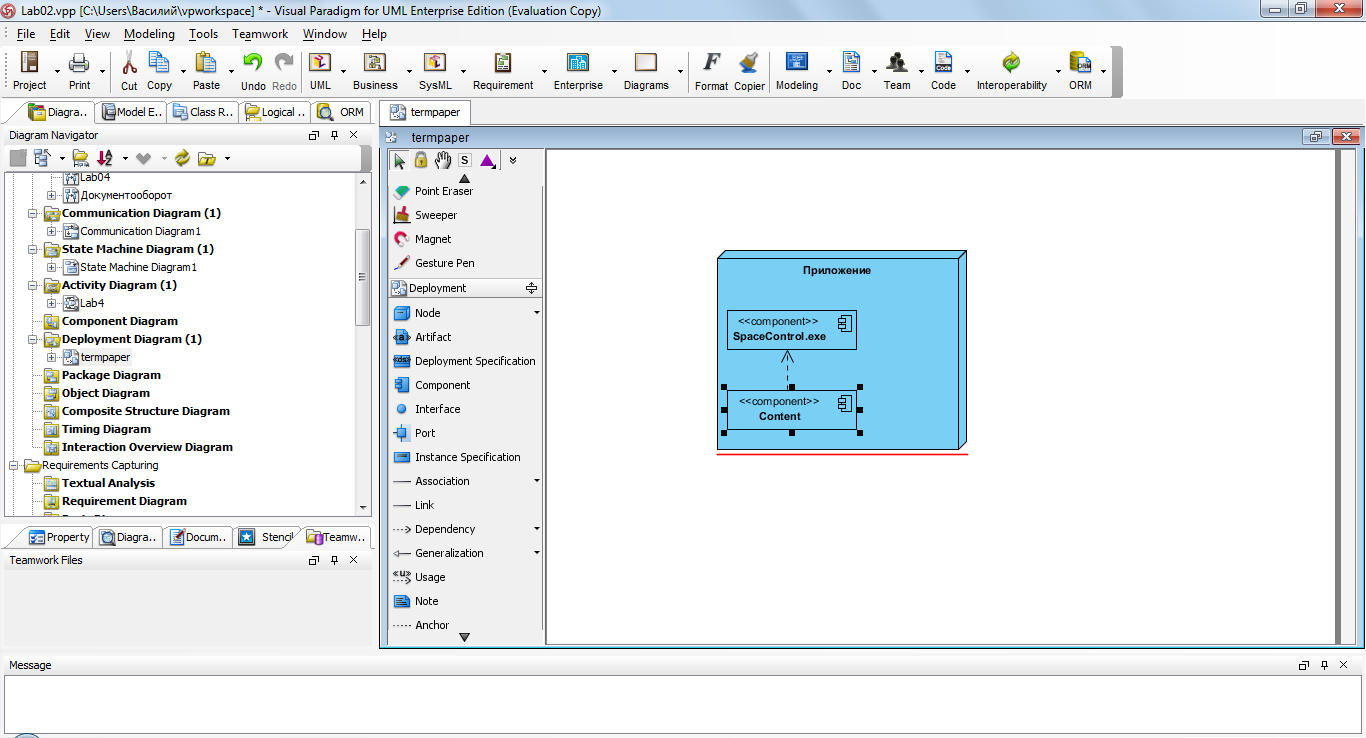


Рисунок 2.2 - Диаграмма развертывания

## 2.3 Временная диаграмма

На рисунке 2.3 представлена временная диаграмма. Данная диаграмма отображает наиболее полный вариант работы программы.

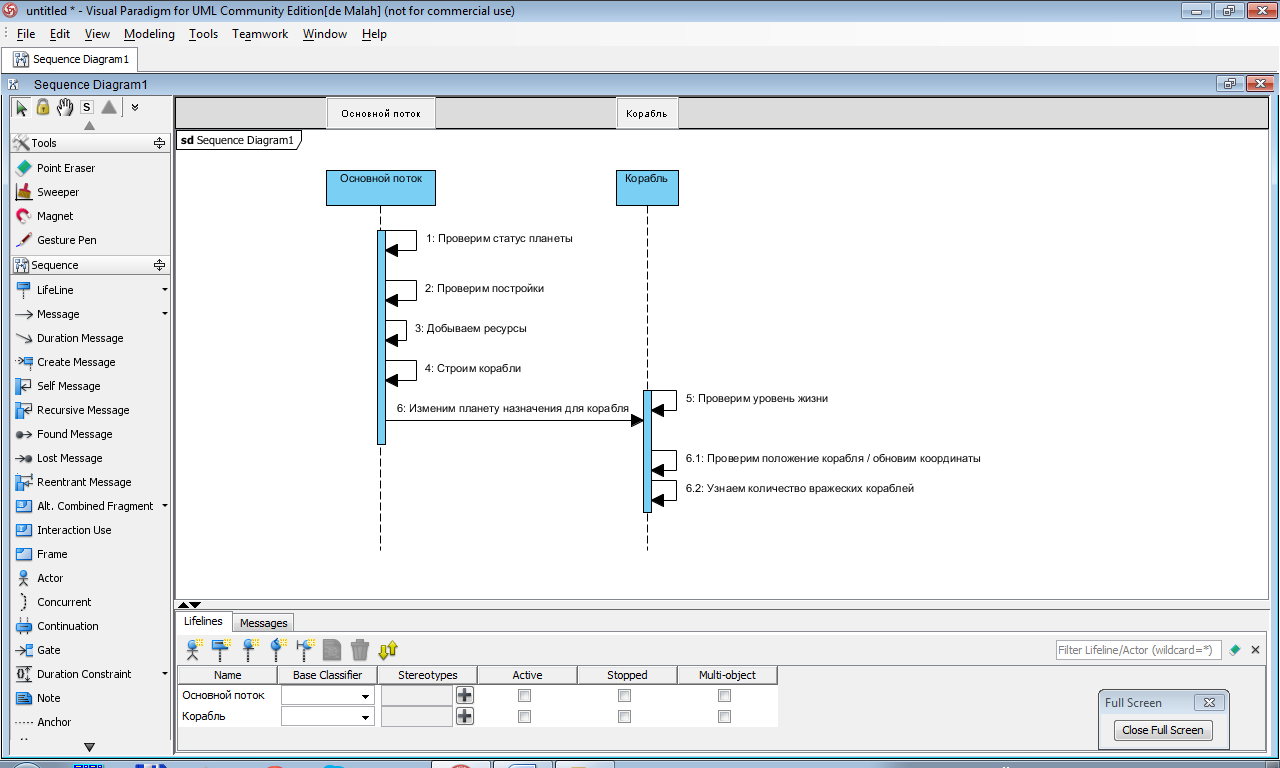


Рисунок 2.3 - Временная диаграмма

## 2.4 Диаграмма деятельности

На рисунке 2.4 представлена диаграмма деятельности.

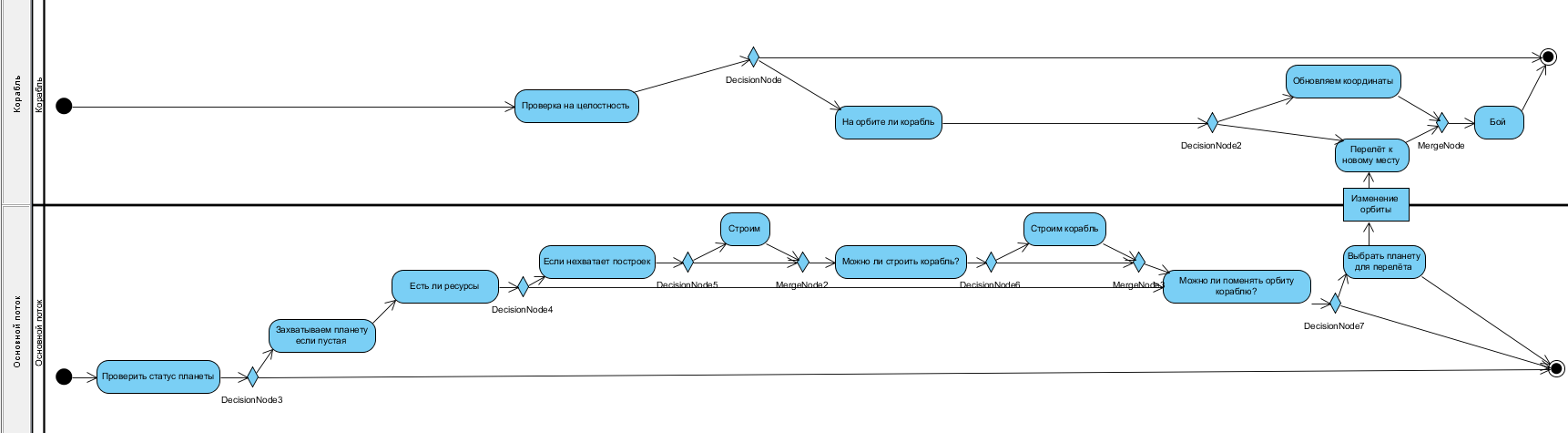


Рисунок 2.4 - Диаграмма деятельности

# 3 Описание программного обеспечения

## 3.1 Общий подход к реализации

Данный программный продукт представляет собой приложение, написанное с использованием XNA 4.0 на языке C#. Приложение были написано в среде Microsoft Visual Studio 2012 и протестировано в операционной системе Windows 7 64 bit и Windows Vista 32 bit.

Приложение XNA имеет определённую структуру. Основные действия в программе реализованы в двух циклах. Один цикл необходим для обновления логики программы – т.е. вычислений которые не связаны с обрисовкой объекта на экране. Именно в этом цикле создаются новые объекты, юниты и изменяются их характеристики. Следующий цикл необходим для отображения объектов на экране и содержит только методы обрисовки всех объектов.

C# — язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML. На C# легко реализуется графический пользовательский интерфейс, поэтому он был выбран для реализации приложения.

## 3.2 Описание приложения

На рисунке 3.1 приведена диаграмма основных классов, реализующих взаимодействие объектами.

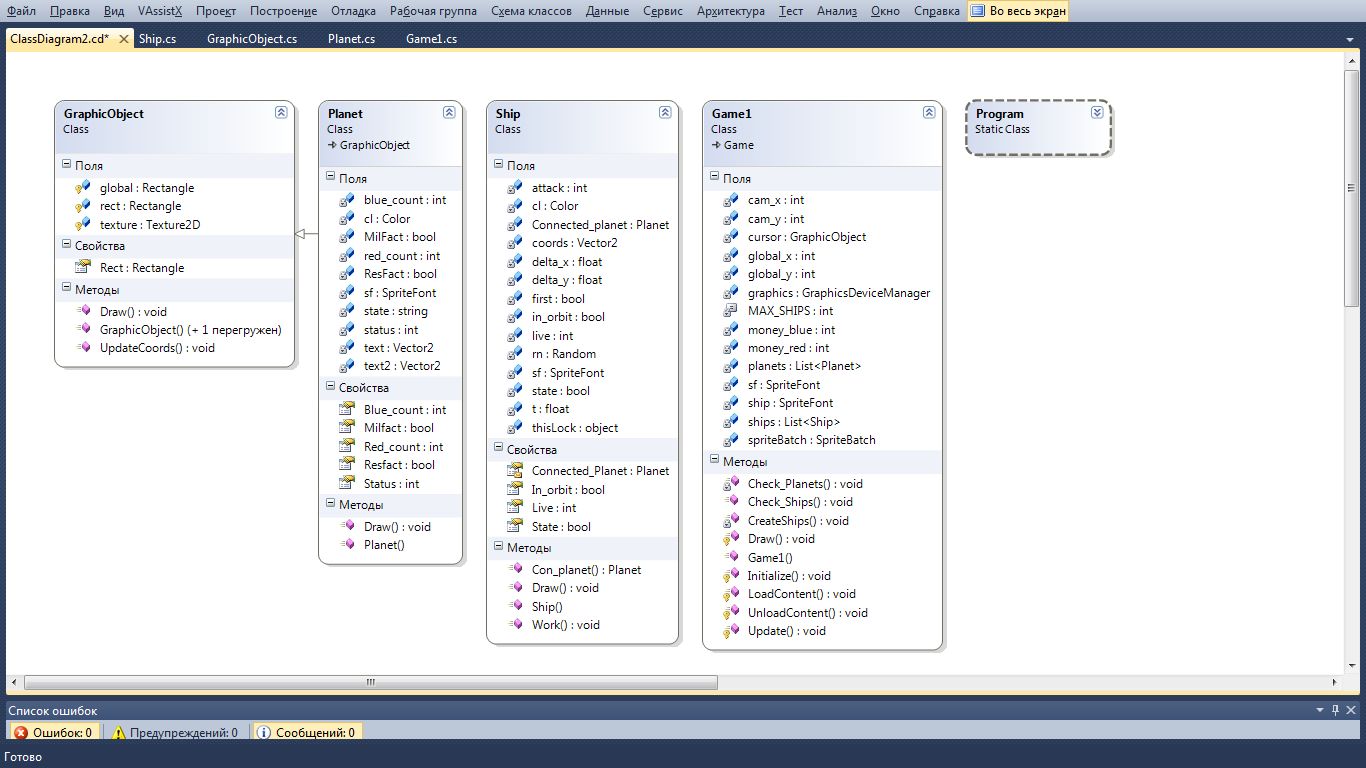


Рисунок 3.1 - Диаграмма основных классов

Так как на диаграмме представлены поля и методы классов, то ниже опишем их функциональное назначение.

Класс GraphicObject - класс реализующий простейший графический примитив.

Поля класса:

* protected Texture2D texture - переменная хранящая текстуру объекта;
* protected Rectangle rect – переменная хранящая координаты примитива;
* protected Rectangle global – переменная хранящая глобальные координаты объекта;

Методы:

* public GraphicObject(Texture2D text, Rectangle rect) - конструктор класса;
* public void UpdateCoords(int x, int y) - обновление координат;
* public virtual void Draw(SpriteBatch sb) – отрисовка примитива;

Класс Ship – класс моделирующий поведение космического корабля.

Поля класса:

* int live – целостность брони корабля;
* bool in\_orbit – на орбите ли корабль;
* Planet Connected\_planet – планета на орбите которой находится корабль;
* bool state – принадлежность корабля определённой фракции;
* int attack – сила атаки корабля;
* float t – начальное положение на орбите;
* SpriteFont sf – отображение корабля;
* Vector2 coords – координаты корабля;
* Color cl – цвет корабля;

Методы:

* public Ship(…) – конструктор корабля
* public void Work(…) – основной рабочий цикл
* public void Draw(…) – отрисовка корабля

Класс Planet – моделирует поведение планеты.

Поля класса:

* int status – кому принадлежит планета;
* Color cl – цвет текста.
* Vector2 text – координаты текста.
* Vector2 text2 – координаты текста.
* SpriteFont sf – текст.
* String state – строка хранящая информацию о состоянии.
* blue\_count – количество синих кораблей на планете
* red\_count – колчество красных кораблей на планете
* bool MilFact – наличие военного завода на планете
* bool ResFact – наличие добывающей фабрики на планете

Методы класса:

* public Planet(...) – конструктор планеты;
* public override void Draw(...) – отрисовка планеты;

Класс Game – основной класс игры.

Поля класса:

* const int MAX\_SHIPS – константа, максимальное количество кораблей;
* int money\_blue – количество ресурсов у синих;
* int money\_red – количество ресурсов у красных;
* int cam\_x – координата камеры;
* int cam\_y – координата камеры;
* GraphicsDeviceManager graphics – объект графики;
* SpriteBatch – spriteBatch – объект отрисовки;
* List<Planet> planets – список планет;
* List<Ship> ships – список кораблей;
* GraphicObject cursor – курсор;
* SpriteFont sf – основной шрифт;
* SpriteFont ship – вспомогательный шрифт;

Методы класса:

* public Game(...) - конструктор класса;
* protected override void Initialize() – инициализация логики;
* protected override void LoadContent() – загрузка контента;
* protected override void UnloadContent() – выгрузка контента;
* protected override void Update(…) – обновление логики;
* void CreateShips() – создание кораблей;
* void Check\_Planets() – проверяем статус планеты;
* void Check\_Ships() – проверка статуса кораблей;
* public override void Draw() – отрисовка всех объектов;

# Заключение

В результате работы над проектом были изучены примитивы синхронизации языка программирования C#. Также изучены возможности этого языка для реализации многопоточных приложений.

Все основные требования к приложению были реализованы.

В дальнейшем можно было бы добавить в программу пользовательский интерфейс, улучить баланс, добавить несколько других юнитов.

# Список использованных источников

1. Дейтел, Х. С# [Текст] : Учебный курс. / Дейтел Х., Дейтел П., Листфилд Дж., Нието Т., Йегер Ш., Златкина М. - СПб.: БХВ-Петербург, 2006. - 1056 с.: ил.
2. Шилдт, Г. C# [Текст] : Учебный курс. / Г. Шилдт – СПб: Питер, 2003 – 471 с.
3. Microsoft Developer Network Library [Электронный ресурс ] Режим доступа: http://msdn.microsoft.com/ru-ru/default.aspx

# Приложение А

# Руководство пользователя

Симуляция стартует сразу же после запуска программы. В левом верхнем углу отображаются ресурсы каждой из сторон.

Над каждой планетой указано:

* Принадлежность планеты определённой стороне.
* Наличие построек на поверхности (MF – завод кораблей, RF – фабрика ресурсов)
* Количество кораблей каждой фракции на орбите планеты

Плюсом обозначается местоположение корабля. Крестом – место где корабль был уничтожен. Цифрами возле корабля обозначается целостность брони корабля.



Рисунок А.1 – Работа приложения

# Приложение Б

# Текст программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework;

using System.Threading;

namespace SpaceControl

{

/// <summary>

/// Класс корабль

/// </summary>

class Ship

{

//целостность корпуса

int live;

public int Live

{

get { return live; }

set { live = value; }

}

//на орбите?

bool in\_orbit;

private object thisLock = new object();

public bool In\_orbit

{

get { return in\_orbit; }

set { in\_orbit = value; }

}

//связанная планета

Planet Connected\_planet;

internal Planet Connected\_Planet

{

get { return Connected\_planet; }

set { Connected\_planet = value; }

}

//статус корабля (true - синий, false - красный)

bool state;

public bool State

{

get { return state; }

set { state = value; }

}

//сила атаки

int attack;

//начальное положение на орбите

float t;

//отображение корабля

SpriteFont sf;

//координаты корабля

Vector2 coords;

//цвет корабля

Color cl;

Random rn;

bool first = true;

float delta\_x;

float delta\_y;

/// <summary>

/// конструктор корабля

/// </summary>

/// <param name="sf">шрифт для отображения корабля</param>

/// <param name="rect">положение в пространстве</param>

/// <param name="state">true - синий, false - красный</param>

/// <param name="connected">планета вокруг которой вращается корабль</param>

/// <param name="t">начальное положение корабля на орбите</param>

public Ship(SpriteFont sf, Rectangle rect, bool state, Planet connected, float t)

{

live = 100;

in\_orbit = true;

this.state = state;

Connected\_planet = connected;

if (state == true)

{

Connected\_planet.Blue\_count += 1;

}

else

{

Connected\_planet.Red\_count += 1;

}

this.t = t;

attack = 2;

rn = new Random();

this.sf = sf;

coords.X = rect.X;

coords.Y = rect.Y;

Thread td = new Thread(Work);

td.IsBackground = true;

td.Start();

}

/// <summary>

/// основоной рабочий цикл

/// </summary>

public void Work()

{

while(live > 0)

{

//если корабль находится на орбите

if (in\_orbit)

{

if (t >= 6.283f)

t = 0.0f;

coords.X = Connected\_planet.Rect.X + 45 + 70 \* (float)Math.Cos(t);

coords.Y = Connected\_planet.Rect.Y + 45 + 70 \* (float)Math.Sin(t);

t += 0.01f;

}

else

{

if (first)

{

delta\_x = (float)Connected\_planet.Rect.X - coords.X;

delta\_y = (float)Connected\_planet.Rect.Y - coords.Y;

delta\_x = delta\_x / 100.0f;

delta\_y = delta\_y / 100.0f;

first = false;

}

coords.X += delta\_x;

coords.Y += delta\_y;

//уже долетели

if (Math.Abs(coords.X - Connected\_planet.Rect.X) < 50 && Math.Abs(coords.Y - Connected\_planet.Rect.Y) < 50)

{

in\_orbit = true;

first = true;

}

}

//битва

if (state == true)

{

live -= attack \* Connected\_planet.Red\_count;

if (live <= 0) Connected\_planet.Blue\_count -= 1;

}

else

{

live -= attack \* Connected\_planet.Blue\_count;

if (live <= 0) Connected\_planet.Red\_count -= 1;

}

Thread.Sleep(20);

}

}

/// <summary>

/// отрисовка корабля

/// </summary>

/// <param name="sb">объект XNA для рисованя</param>

public void Draw(SpriteBatch sb)

{

//чей корабль?

if (state)

cl = Color.DarkBlue;

else

cl = Color.Red;

//живой или нет?

if(live > 0)

{

sb.DrawString(sf,"+" + live,coords, cl);

}

else

sb.DrawString(sf, "X", coords, cl);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework;

namespace SpaceControl

{

/// <summary>

/// Графический примитив

/// </summary>

class GraphicObject

{

protected Texture2D texture;

protected Rectangle rect;

protected Rectangle global;

public Rectangle Rect

{

get { return rect; }

set { rect = value; }

}

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public GraphicObject()

{

}

/// <summary>

/// Конструктор графического примитива

/// </summary>

/// <param name="text">Текстура</param>

/// <param name="rect">положение объекта</param>

public GraphicObject(Texture2D text, Rectangle rect)

{

this.texture = text;

this.rect = rect;

this.global.X = rect.X;

this.global.Y = rect.Y;

}

/// <summary>

/// Обновление координат

/// </summary>

/// <param name="x">Новый X</param>

/// <param name="y">Новый Y</param>

public void UpdateCoords(int x, int y)

{

rect.X = global.X + x;

rect.Y = global.Y + y;

}

/// <summary>

/// Отрисовка графического примитива

/// </summary>

/// <param name="sb">Объект XNA для отрисовки</param>

public virtual void Draw(SpriteBatch sb)

{

sb.Draw(texture, rect, Color.White);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework;

namespace SpaceControl

{

/// <summary>

/// Класс планета

/// </summary>

class Planet : GraphicObject

{

//чья планета

int status;

public int Status

{

get { return status; }

set { status = value; }

}

//цвет текста

Color cl;

//коордианты текста

Vector2 text;

Vector2 text2;

//объект текст

SpriteFont sf;

//строка состояния планеты

String state;

//количество кораблей на планете

int blue\_count = 0;

public int Blue\_count

{

get { return blue\_count; }

set { blue\_count = value; }

}

int red\_count = 0;

public int Red\_count

{

get { return red\_count; }

set { red\_count = value; }

}

//постройки

bool MilFact = false; //военный завод

public bool Milfact

{

get { return MilFact; }

set { MilFact = value; }

}

bool ResFact = false; //добывающая фабрика

public bool Resfact

{

get { return ResFact; }

set { ResFact = value; }

}

/// <summary>

/// Конструктор планеты

/// </summary>

/// <param name="tex">текстура</param>

/// <param name="rec">положение</param>

/// <param name="sp">шрифт</param>

/// <param name="stat">статтус планеты (0 - нейтральная, 1 - синяя, 2 - красная)</param>

public Planet(Texture2D tex, Rectangle rec, SpriteFont sp, int stat) : base(tex, rec)

{

text.X = rec.X;

text.Y = rec.Y - 20;

text2.X = rec.X;

text2.Y = rec.Y - 40;

this.sf = sp;

this.status = stat;

}

/// <summary>

/// Отрисовка планеты

/// </summary>

/// <param name="sb">Объект XNA для отрисовки</param>

public override void Draw(SpriteBatch sb)

{

state = "";

//установка статуса планеты

switch (status)

{

case 0: { state = "Empty "; cl = Color.Gray; break; }

case 1: { state = "UNSC "; cl = Color.DarkBlue; break; }

case 2: { state = "Covenant "; cl = Color.Red; break; }

default: break;

}

//есть ли постройки

if (MilFact)

state += "MF ";

if (ResFact)

state += "RF ";

String state2 = "B: " + blue\_count;

state2 += " R: " + red\_count;

base.Draw(sb);

sb.DrawString(sf,state,text,cl);

sb.DrawString(sf,state2,text2,cl);

}

}

}

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using System.Threading;

using System.Collections.Generic;

using System;

namespace SpaceControl

{

/// <summary>

/// This is the main type for your game

/// </summary>

public class Game1 : Microsoft.Xna.Framework.Game

{

//ограничение по количеству кораблей

const int MAX\_SHIPS = 20;

int money\_blue;

int money\_red;

/// <summary>

/// Задаем глобальные координаты

/// </summary>

int global\_x = 0;

int global\_y = 0;

/// <summary>

/// координаты камеры

/// </summary>

int cam\_x = 0;

int cam\_y = 0;

GraphicsDeviceManager graphics;

SpriteBatch spriteBatch;

//список планет

List<Planet> planets = new List<Planet>();

//список кораблей

List<Ship> ships = new List<Ship>();

//изображение курсора

GraphicObject cursor;

//основной шрифт

SpriteFont sf;

//отображение планет

SpriteFont ship;

public Game1()

{

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

graphics.PreferredBackBufferWidth = 800;

graphics.PreferredBackBufferHeight = 600;

//graphics.IsFullScreen = true;

Window.Title = "Space Control";

//ресурсы на момент начала игры

money\_blue = money\_red = 1500;

}

/// <summary>

/// Allows the game to perform any initialization it needs to before starting to run.

/// This is where it can query for any required services and load any non-graphic

/// related content. Calling base.Initialize will enumerate through any components

/// and initialize them as well.

/// </summary>

protected override void Initialize()

{

// TODO: Add your initialization logic here

base.Initialize();

}

/// <summary>

/// LoadContent will be called once per game and is the place to load

/// all of your content.

/// </summary>

protected override void LoadContent()

{

// Create a new SpriteBatch, which can be used to draw textures.

spriteBatch = new SpriteBatch(GraphicsDevice);

sf = Content.Load<SpriteFont>("font");

ship = Content.Load<SpriteFont>("ship");

// TODO: use this.Content to load your game content here

//создаем планеты

planets.Add(new Planet(Content.Load<Texture2D>("Earth"),new Rectangle(300,300,100,100),sf,1));

planets.Add(new Planet(Content.Load<Texture2D>("Mars"), new Rectangle(30, 65, 100, 100),sf,2));

planets.Add(new Planet(Content.Load<Texture2D>("Mercury"), new Rectangle(80, 400, 100, 100),sf,0));

planets.Add(new Planet(Content.Load<Texture2D>("Venus"), new Rectangle(600, 500, 100, 100),sf,0));

planets.Add(new Planet(Content.Load<Texture2D>("Uranus"), new Rectangle(500, 300, 100, 100),sf,0));

//загрузим курсор

cursor = new GraphicObject(Content.Load<Texture2D>("curs"), new Rectangle(0, 0, 20, 20));

}

/// <summary>

/// UnloadContent will be called once per game and is the place to unload

/// all content.

/// </summary>

protected override void UnloadContent()

{

// TODO: Unload any non ContentManager content here

}

/// <summary>

/// Allows the game to run logic such as updating the world,

/// checking for collisions, gathering input, and playing audio.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Update(GameTime gameTime)

{

// Allows the game to exit

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

this.Exit();

// TODO: Add your update logic here

MouseState ms = Mouse.GetState();

cursor.UpdateCoords(ms.X, ms.Y);

foreach (var item in planets)

{

item.UpdateCoords(cam\_x,cam\_y);

}

//работа с планетами

Check\_Planets();

//работа с кораблями

Check\_Ships();

KeyboardState keys = Keyboard.GetState();

if (keys.IsKeyDown(Keys.Escape))

this.Exit();

/\*

if (keys.IsKeyDown(Keys.W))

cam\_y -= 10;

if (keys.IsKeyDown(Keys.A))

cam\_x -= 10;

if (keys.IsKeyDown(Keys.S))

cam\_y += 10;

if (keys.IsKeyDown(Keys.D))

cam\_x += 10;

\*/

base.Update(gameTime);

}

/// <summary>

/// Метод создания кораблей

/// </summary>

void CreateShips()

{

Random rn = new Random();

for (int i = 0; i < 10; i++)

{

ships.Add(new Ship(ship, new Rectangle(rn.Next(0, 400), rn.Next(0, 400), 10, 10),true,planets[rn.Next(0,2)],(float)(rn.NextDouble() \* 6)));

}

}

/// <summary>

/// Проверяем статус планеты

/// </summary>

void Check\_Planets()

{

Random rn = new Random();

for(int i = 0; i < planets.Count; i++)

{

//захват планеты

if(planets[i].Blue\_count == 0 && planets[i].Red\_count > 0)

{

planets[i].Status = 2;

}

else if (planets[i].Blue\_count > 0 && planets[i].Red\_count == 0)

{

planets[i].Status = 1;

}

//если планета не пустая

if (planets[i].Status != 0)

{

if (planets[i].Status == 1)

{

if (money\_blue >= 500)

{

//строим добывающий завод

if (planets[i].Resfact == false && rn.Next(0, 200) < 1)

{

money\_blue -= 500;

planets[i].Resfact = true;

break;

}

//строим военный завод

if (planets[i].Milfact == false && rn.Next(0, 200) < 1)

{

money\_blue -= 500;

planets[i].Milfact = true;

break;

}

}

//добываем ресурсы

if (planets[i].Resfact && rn.Next(0,200) < 1)

{

money\_blue += 10;

break;

}

//строим корабли

if (money\_blue > 50 && rn.Next(0,200) < 1 && planets[i].Milfact)

{

if (ships.Count < MAX\_SHIPS)

{

ships.Add(new Ship(ship, new Rectangle(0, 0, 10, 10), true, planets[i], (float)(rn.NextDouble() \* 6)));

}

money\_blue -= 50;

break;

}

}

else

{

if (money\_red >= 500)

{

//строим добывающий завод

if (planets[i].Resfact == false && rn.Next(0, 200) < 1)

{

money\_red -= 500;

planets[i].Resfact = true;

break;

}

//строим военный завод

if (planets[i].Milfact == false && rn.Next(0, 200) < 1)

{

money\_red -= 500;

planets[i].Milfact = true;

break;

}

}

//добываем ресурсы

if (planets[i].Resfact && rn.Next(0, 200) < 1)

{

money\_red += 10;

break;

}

//строим корабли

if (money\_red > 50 && rn.Next(0, 200) < 1 && planets[i].Milfact && ships.Count < MAX\_SHIPS)

{

ships.Add(new Ship(ship, new Rectangle(0, 0, 10, 10), false, planets[i], (float)(rn.NextDouble() \* 6)));

money\_red -= 50;

break;

}

}

}

}

}

/// <summary>

/// Проверяем статус корабля

/// </summary>

public void Check\_Ships()

{

Random rn = new Random();

for (int i = 0; i < ships.Count; i++ )

{

if (ships[i].Live > 0 && rn.Next(0, 500) < 1)

{

Planet n\_plan = planets[rn.Next(0, planets.Count)];

if(!ships[i].Connected\_Planet.Equals(n\_plan))

{

//изменили счётчик кораблей на планете

if (ships[i].State == true)

{

n\_plan.Blue\_count += 1;

ships[i].Connected\_Planet.Blue\_count -= 1;

//ships[i].ChangeOrbit();

}

else

{

n\_plan.Red\_count += 1;

ships[i].Connected\_Planet.Red\_count -= 1;

//ships[i].ChangeOrbit();

}

ships[i].Connected\_Planet = n\_plan;

}

}

}

}

/// <summary>

/// This is called when the game should draw itself.

/// </summary>

/// <param name="gameTime">Provides a snapshot of timing values.</param>

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.Black);

// TODO: Add your drawing code here

spriteBatch.Begin();

//отобразим планеты

foreach (var item in planets)

{

item.Draw(spriteBatch);

}

//отобразим корабли

foreach (var item in ships)

{

item.Draw(spriteBatch);

}

//отобразим строку состояния

spriteBatch.DrawString(sf, "Resources UNSC: " + money\_blue, Vector2.Zero, Color.BlueViolet);

spriteBatch.DrawString(sf, "Resources Covenant: " + money\_red, new Vector2(0, 15), Color.Red);

//отобразим курсор

cursor.Draw(spriteBatch);

spriteBatch.End();

base.Draw(gameTime);

}

}

}