МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Руководитель отдела информационных технологий и связи
ООО «Модерн Гласс»
_____ А.Г. Григорьев
“___”_____ 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой,
д.ф.-м.н., профессор
_____ Л.Б. Соколинский
“___”_____ 2020 г.

Разработка компьютерной 2D-игры «Space Shifter» на платформе Unity

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ –09.03.04.2020.308-045.ВКР

Научный руководитель,
к.ф.-м.н., доцент кафедры СП
_____ С.А. Иванов

Автор работы,
студент группы КЭ-403
_____ Д.В. Колобанов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
“___”_____ 2020 г.

Челябинск-2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
09.02.2020

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Колобанову Дмитрию Вячеславовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. Тема работы (утверждена приказом ректора от 24.04.2020 № 627)
Разработка компьютерной 2D-игры «Space Shifter» на платформе Unity.
2. Срок сдачи студентом законченной работы: 05.06.2020.
3. Исходные данные к работе
Thorn, A. (2013). Learn Unity for 2D Game Development. In Learn Unity for 2D Game Development. https://doi.org/10.1007/978-1-4302-6230-5
4. Перечень подлежащих разработке вопросов
4.1. Провести обзор аналогов.
4.2. Спроектировать приложение.
4.3. Реализовать и протестировать приложение.
5. Дата выдачи задания: 08.02.2020.

Научный руководитель
к.ф.-м.н., доцент кафедры СП С.А. Иванов

Задание принял к исполнению Д.В. Колобанов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В ходе развития электроники в 1971 году появляется первая компьютерная игра Spacewar. Данное событие ознаменовало появление нового направления индустрии развлечений. Компьютерные игры быстро завоевали популярность. Уже в 1981 год индустрия игр в США заработала $5 млрд.

На данный момент индустрия компьютерных игр сравнима с индустрией музыки и кино. На 2019 год объём рынка составил 152,1 млрд. долларов.

Многие служат не только для развлечения. Некоторые из них служат для отработки технологии машинного обучения к примеру «Alphastar» от компании «DeepMind» в игре «Starcraft 2» обученный бот достиг ранга грандмастера и имел более 94% побед[1]. Научных исследований к примеру проект «Discovery» в игре «Eve online» [2], который был создан для помощи в поиске экзопланет.

Объём рынка стремительно растёт. На данный момент появляются множество «indie» проектов. Это проекты с низким бюджетом, разрабатывающиеся небольшой группой людей. Такие проекты разрабатываются небольшой командой от 1-10 человек и легко окупаются. Такие игры часто экспериментируют с жанрами и имеют большую свободу в выборе игровых механик и способов монетизации. Что и обуславливает популярность данного жанра. К таким играм относятся «Rimworld», «Minecraft», «Terraria» и многие другие.

Постановка задачи

Целью данной работы является создание мобильной игры «Space shifter» на платформе Unity.

Для решения поставленной цели необходимо выполнить следующие задачи:

1) обзор существующих аналогов игрового приложения;

2) обзор программных средств реализации игрового приложения;

3) проектирование игрового приложения;

4) реализация игрового приложения;

5) тестирование игрового приложения.

Структура и объем работы

Работа состоит из введения, трех глав, заключения, списка литературы и приложения. Объем работы составляет 39 страниц, список литературы содержит 18 источников и объём приложения три страницы.

Содержание работы

Введение раскрывает актуальность выбранной темы для выпускной квалификационной работы.

В первой главе проведен анализ существующих решений для постановки задачи по реализации приложения и средств разработки и наиболее популярных подходов к разработке игровых приложений.

Во второй главе проведено проектирование приложения с учетом программных средств и необходимых компонентов.

В третьей главе представлена файловая структура приложения, приведены реализации алгоритмов, и разработана диаграмма UML для игрового приложения.

В заключении описываются основные результаты, полученные при выполнении работы.

1. ОБЗОР АНАЛОГОВ

1.1. Обзор аналогичных проектов

В ходе выполнения дипломной работы были изучены несколько игр со схожей тематикой в магазине приложений Steam.

Космические рейнджеры

Космические рейнджеры – это серия двухмерных ролевых игр о космических приключениях, в которых присутствует возможность свободного путешествия по звездным системам, связанным между собой в небольшую виртуальную галактику. Основные механики игры - это торговля, квесты, космические бои и исследование. Оценки игры в steam составляют 94 процента положительных отзывов. Игра представлена на рисунке 1.

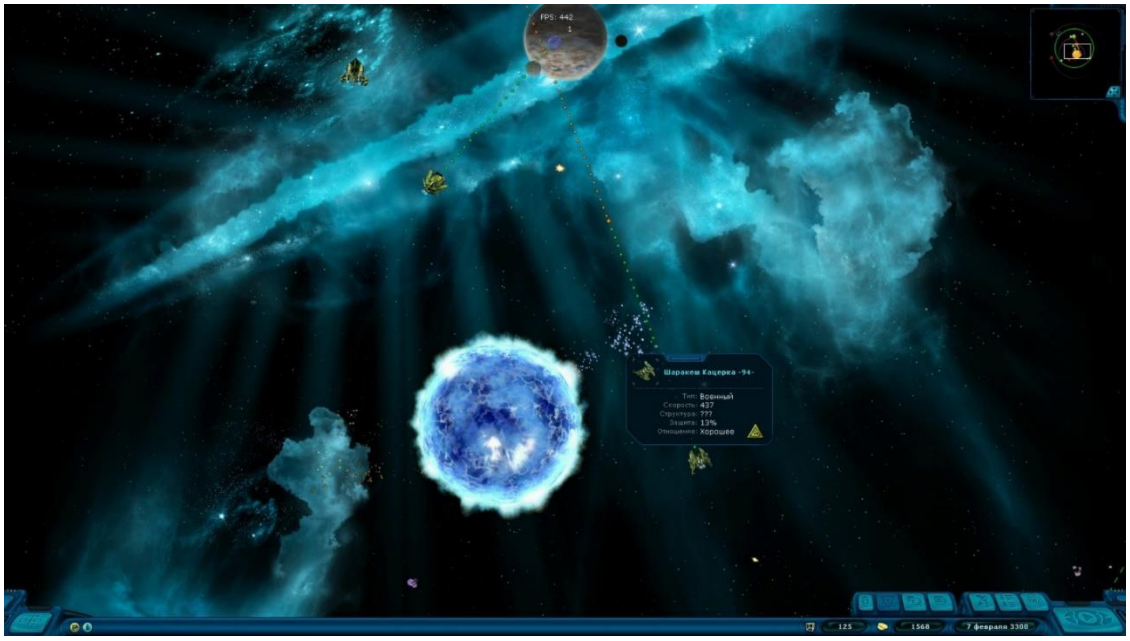

Рис. 1. Игра «Космические рейнджеры»

NO MAN’S SKY

NO MAN’S SKY – это игра в жанре песочница, получившая известность благодаря своей огромной виртуальной вселенной, генерируемой случайным образом. Цель игры – исследование этой самой вселенной, заключающееся в полете к ее центру с посадками на встречающихся по пути планетах и космических станциях. Делать такие посадки необходимо для

сбора ресурсов, торговли, создания предметов и поиска информации о мире. Игра понравилась многим пользователям из-за своего необычного дизайна, масштаба и ощущения безграничной свободы, что представлена на рисунке 2.

Рис. 2. Игра «NO MAN’S SKY»

ELITE DANGEROUS

ELITE DANGEROUS – онлайн игра про космос с открытым миром, с огромной вселенной размером 400 миллиардов звездных систем, современной графикой и множеством разных видов кораблей.

Рис. 3. Игра «ELITE DANGEROUS»

Выбранный корабль можно модифицировать по своему вкусу и в дальнейшем всячески совершенствовать и улучшать. Также развитая система торговли и добычи ресурсов и исследования мира, представленная на рисунке 3. Приятной особенностью игры является и довольно регулярный выход обновлений с новым контентом.

СЕРИЯ X

СЕРИЯ X – это линейка игр, которая ведет свою историю аж с 1999 года. Правда, не все ее выпуски получались качественными, но, тем не менее, у франшизы много поклонников.

Игры серии X предлагают изучать открытый мир, добывать ресурсы, торговать, сражаться, строить космические станции и даже руководить звездной империей. Игрокам доступно множество типов космических кораблей, от небольших разведчиков до огромных авианосцев, и каждым из них можно управлять и изменять на свой вкус (рисунок 4).

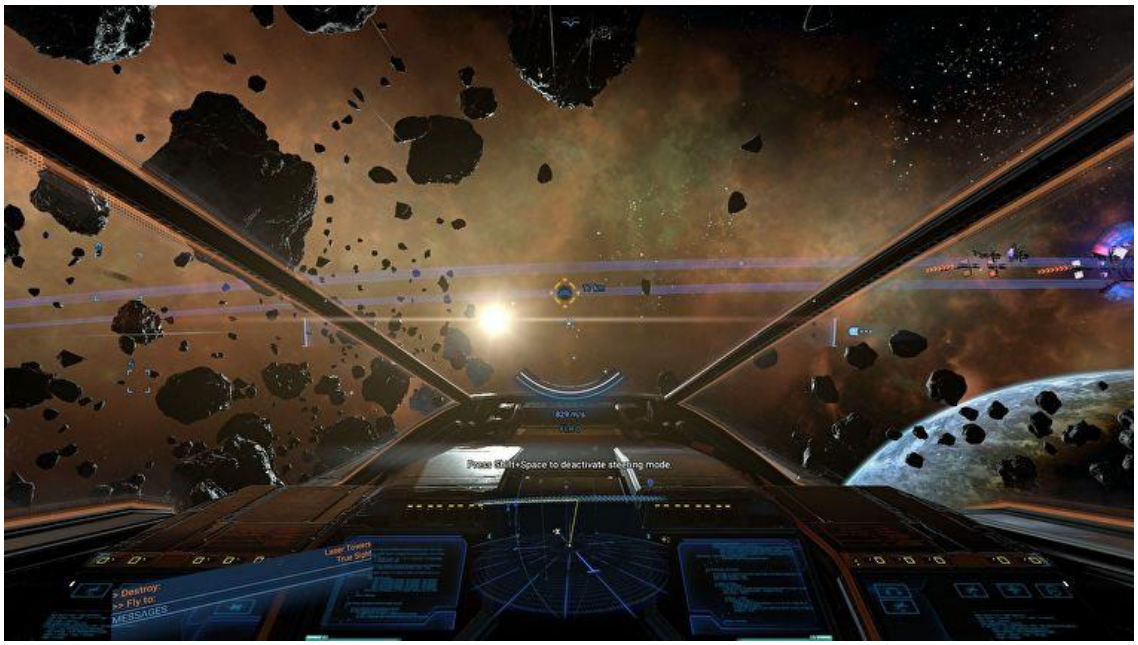

Рис. 4. Игра «ELITE DANGEROUS»

Вывод по главе обзор аналогичных проектов

Были проанализированы основные аналоги разрабатываемого приложения. Из полученных данных был сделан вывод касательно необходи-

мых игровых механик и основной цели игры. С учётом полученных данных был составлен план разработки и выбраны механики к реализации. Основные механики, встречаемые в данном типе игр, торговля, перемещение по космосу, улучшение и покупка новых космических кораблей.

Также как игровые особенности включен открытый мир и система развития мира.

Открытый мир – это термин в компьютерных играх, обозначающий виртуальный мир, который игрок может свободно исследовать и свободно достигать в нем своих целей. Обычно противопоставляется играм с более линейным геймплеем[4].

1.2. Анализ существующих решений для реализации проекта Unity

Unity - межплатформенная среда разработки компьютерных игр[15], разработанная американской компанией Unity Technologies. Unity позволяет создавать приложения, работающие на различных платформах, включающих персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие[16].

В отличие от многих сред разработки, у Unity имеется два основных преимущества: наличие визуальной среды разработки и межплатформенная поддержка. Первый фактор включает не только инструментарий визуального моделирования, но и интегрированную среду, цепочку сборки, что направлено на повышение производительности разработчиков, в частности, этапов создания прототипов и тестирования.

Преимуществами выделяющие Unity это простой и информативной визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. К недостаткам относят появление сложностей при работе с многокомпонентными схемами и затруднения при подключении внешних библиотек.

Плюсы:

1) низкая стоимость;

2) простота освоение;

3) мультиплатформенность;

4) большое сообщество разработчиков;

5) постоянное совершенствование игрового движка;

6) большой магазин ассетов.

Минусы:

1) сложность при работе с многокомпонентными системами;

2) затруднение подключения внешних библиотек.

Unreal Engine

Unity -среда разработки, разрабатываемый и поддерживаемый компанией Epic Games. Первой игрой на этом движке был шутер от первого лица Unreal, выпущенный в 1998 году. Хотя движок первоначально был предназначен для разработки шутеров от первого лица, его последующие версии успешно применялись в играх самых различных жанров. В прошлом движок распространялся на условиях оплаты ежемесячной подписки; с 2015 года Unreal Engine бесплатен, но разработчики использующих его приложений обязаны перечислять 5% роялти от общемирового дохода с некоторыми условиями. Язык написания C++ и визуальный язык программирования [3].

Плюсы:

1) большое количество инструментов для разных целей;

2) мультиплатформенность;

3) визуальное программирование Blueprint;

4) возможность создавать хорошую графику.

Минусы:

1) сложность в освоении.

CryEngine

CryEngine - среда разработки, созданная немецкой частной компанией Crytek в 2002 году и первоначально используемый в шутере от первого лица Far Cry. «CryEngine» - коммерческий движок, который предлагается для лицензирования другим компаниям. С 30 марта 2006 года все права на движок принадлежат компании Ubisoft.

Весной 2016 года компания перешла на модель распространения «плати сколько хочешь», предполагающую возможность свободно использовать движок для разработки игр, код CryEngine был опубликован (под проприетарной лицензией) на GitHub.

Плюсы:

1) модульность;

2) поддержка визуального программирования;

3) хорошая графика;

4) редактор игры в реальном времени.

Минусы:

1) плохая документация;

2) маленькое сообщество.

Исходя из произведенного обзора, для разработки был выбрана среда разработки Unity [17]. Он популярен, прост в освоении, имеет множество средств для работы двухмерной графикой. Язык C# [16], на котором пишутся скрипты в Unity, обладает большим функционалом и крайне удобен для разработки подобных проектов. Так же большой магазин готовых материалов позволяет уменьшить время на поиск необходимых текстур, звуков и анимаций.

Вывод по главе анализ существующих решений для реализации проекта

В ходе анализа предметной области был произведен обзор существующих аналогов, в ходе которого были выявлены основные принципы формирования жанра и правила построения пользовательского интерфейса.

После этого был произведен обзор существующих решений, выявлены их недостатки и достоинства. На основе этих данных были выбраны инструменты, которые будут использоваться в разработке.

Все материалы, полученные на основе обзора, позволяют определить требования к проектированию и реализации игрового приложения.

2. ПРОЕКТИРОВАНИЕ

2.1. Цель игры

Основной идея игры выживание и развитие своего персонажа путём покупки более совершенного оборудования и улучшения отношений с фракциями. Условием победы является выполнение задачи в событии. Существует несколько различных событий, приводящих к концу игры.

2.2. Основная концепция

Игра является игровым приложением на платформе PC.

Камера в игре расположена в формате «вид сверху».

Управление в игре реализовывается посредством нажатия заранее настроенных клавиш на клавиатуре, а также нажатием на кнопки интерфейса мышкой.

Экраны:

1) станции/планеты;

2) открытый космос;

3) карта;

4) главное меню.

Игровые механики:

1) сражения;

2) торговля;

3) перемещение;

4) аномалии;

5) модификация корабля.

Сражения

Сражения – это вид игровой активности, в которой игрок или искусственный интеллект пытается уничтожить корабль своего оппонента. Происходит на экране «Открытый космос».

Торговля

Торговля – это вид игровой активности, в которой игрок может обменивать игровая валюту на различные предметы и наоборот. Происходит на станциях или на планетах. Основная цель активности получения нового оборудования и зарабатывание валюты путем перепродажи разных товаров.

Перемещение

Перемещение – возможность игрока перемещаться ка по глобальной карте (Звёздной карте) так и по локальной (Карте системы).

Аномалии

Игровые локации, вход в которые расположен в открытом космосе. При активации данной локации вас переносят на специальную карту со сгенерированными противниками или другими активностями.

Модификация корабля

Игрок владеет собственным кораблём, который он может улучшать или заменять. Модификация происходит на планете или станции. Для покупки и установки новых деталей требуется игровая валюта.

Виртуальная валюта или игровая валюта - частные электронные деньги, которые используются для приобретения и продажи виртуальных товаров в различных сетевых сообществах: социальных сетях, виртуальных мирах и онлайн-играх[11].

2.3. Игровой мир

Игровой мир состоит из множества локаций, на которых расположены множество объектов, таких как станции и планеты, корабли, управляемые ИИ и аномалии, представляющие собой точку перемещения в другую локацию. Перемещение между локациями осуществляется с помощью карты. Существует ограничение на перемещение.

1)Расстояние между системой, в которой находится игрок и системой, куда необходимо переместиться, должно быть меньше силы двигателя.

2) Перемещение можно совершить, только находясь достаточно далеко от центра системы.

3) Перемещение занимает некоторое время.

Планеты и станции доступны для взаимодействия. С помощью них можно модифицировать и починить корабль, продать, купить товары.

Мир функционирует без игрока. Планеты и станции производят и потребляют ресурсы. Корабли переносят ресурсы между разными планетами и станциями, атакуют друг друга, и улучшают своё оборудование.

Оборудование

Всё оборудование разделено на несколько типов:

1) генератор – генерирует энергию для питания корабля;

2) щит – увеличивает защитное поле корабля;

3) броня – увеличивает броню корабля;

4) орудия – даёт возможность корабля стрелять во время боя;

5) двигатель – увеличивает скорость и мощность корабля.

2.4. Интерфейс

Интерфейс состоит из главного меню, меню карты, меню модификации, меню торговли, меню космоса и меню настроек, игровое меню, панель смерти.

В главном меню находятся кнопки старта игры, а также перехода в меню настроек.

Меню карты содержит кнопки выбора системы, в которую игрок хочет переместиться и информацию о максимальной дальности перемещения.

В меню модификации находятся кнопки выбора, покупки, продажи и установки оборудования. В нём также находится информация о характеристиках корабля.

Игровое меню открывается при нажатии кнопки esc. Игровое меню может быть вызвано только во время игры. При вызове игровое меню приостанавливается игровой процесс до выхода из него. Во игровом меню расположены кнопки «Продолжить игру», «Выйти в главное меню» и «Открыть настройки».

В меню торговли находятся кнопки покупки и продажи ресурсов. В нём же хранится информация о ценах на ресурсы и их количестве у игрока и на станции.

Меню космоса появляется во время нахождения игрока на корабле в космосе и содержит кнопки «Открыть карту», напасть на корабль, сесть на станцию/планету, активировать аномалию.

Панель смерти открывается при смерти персонажа. В панели смерти расположены кнопки «Выйти в главное меню».

3. РЕАЛИЗАЦИЯ

3.1. Диаграммы

Проект игрового приложения состоит из набора каталогов, содержащих файлы анимации и звукового сопровождения; изображения; игровые скрипты и готовые объекты. Файловая структура приложения представлена на рисунке 5.

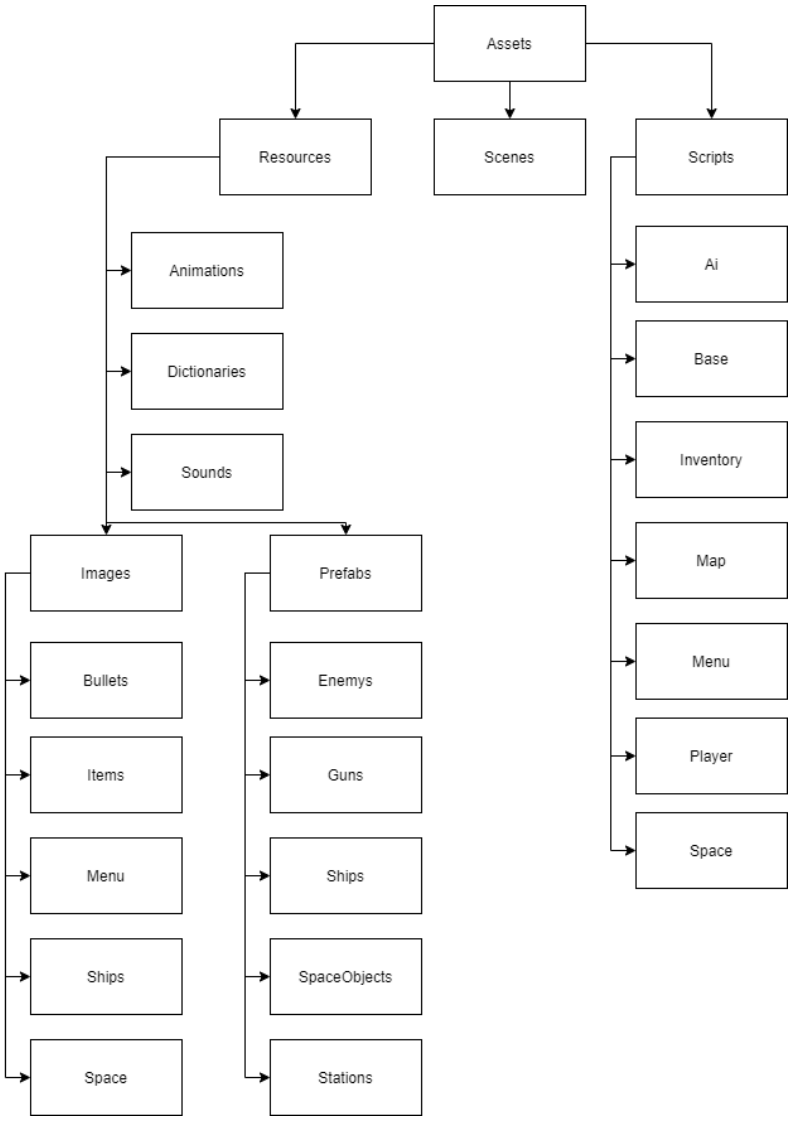

Рис. 5. Файловая структура приложения

С помощью языка графического моделирования UML была построена диаграмма классов системы карты, представленная на рисунке 6. Диаграммы остальных блоков представлены в приложении на рисунках А.1 - А.5.

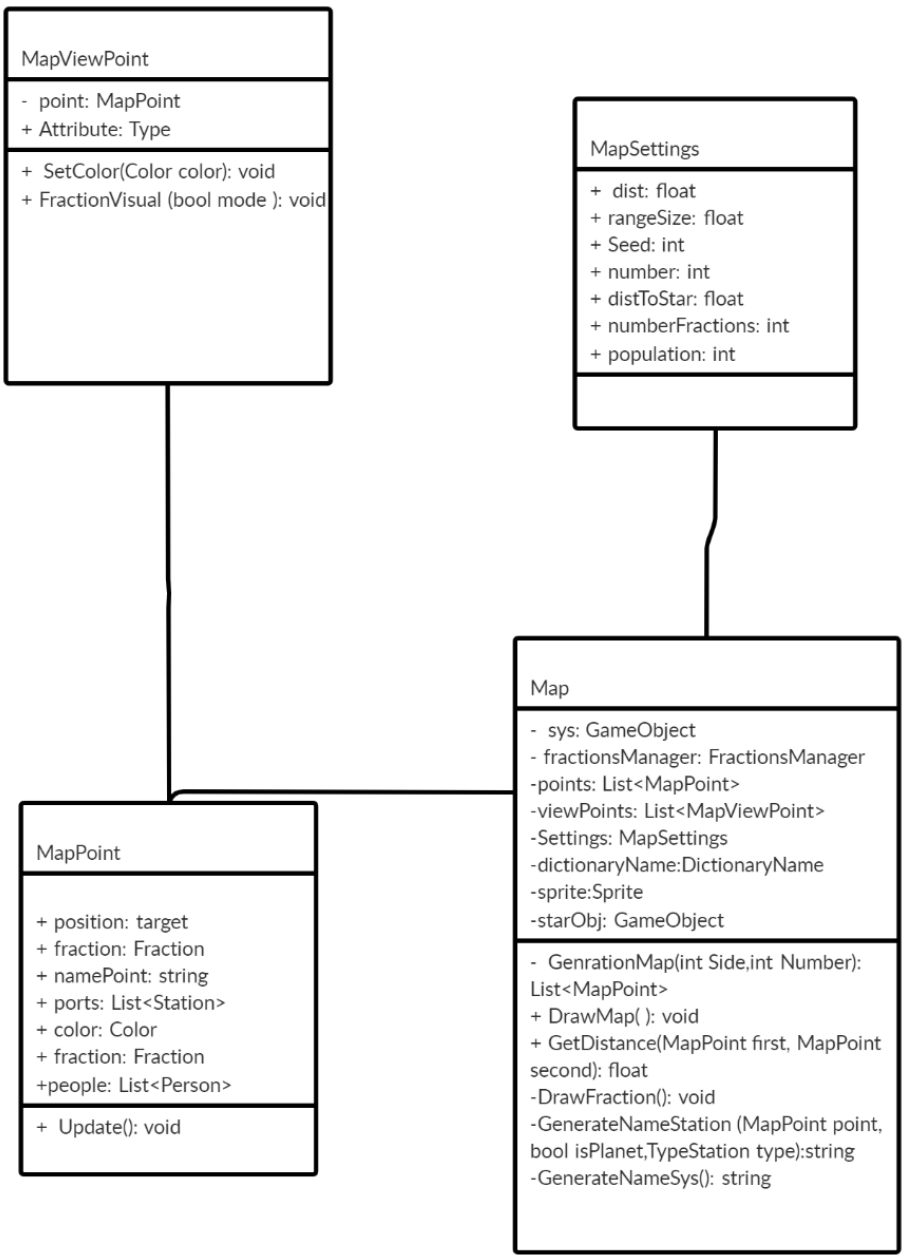

Рис. 6. UML диаграмма классов карты

3.2. Карта

Вся игра разделена на четыре основных системы:

1) карта;
2) планеты и станции;
3) открытый космос;
4) общие.

Карта – система генерации мира и симуляции жизни в игре. Представляет собой систему с множеством классов, каждый из которых изображает персонажа игры и отыгрывает роль торговца, пирата или военного. У каждого персонажа есть свои цели, оборудование и средства. Количество персонажей непостоянно и может падать с течением времени, так как персонажи могут умереть. Но системы следит за балансом классов и количеством персонажей.

Система также включает производство и потребление, находящихся на планете, ресурсов раз в определённое время. Количество и тип производимых ресурсов определяет тип планеты, что создаёт потребность в обмене одних на другие. Планеты не могут обеспечивать себя всеми ресурсами и при их нехватке ресурсов цены на них растут, а при избытке падают, это создаёт возможность получение прибыли от торговли.

В данную систему включены генерация мира, в котором существует игрок и Ии, система взаимодействия с картой, переходов по локациям и визуализацию доступных мест рисунке 7.

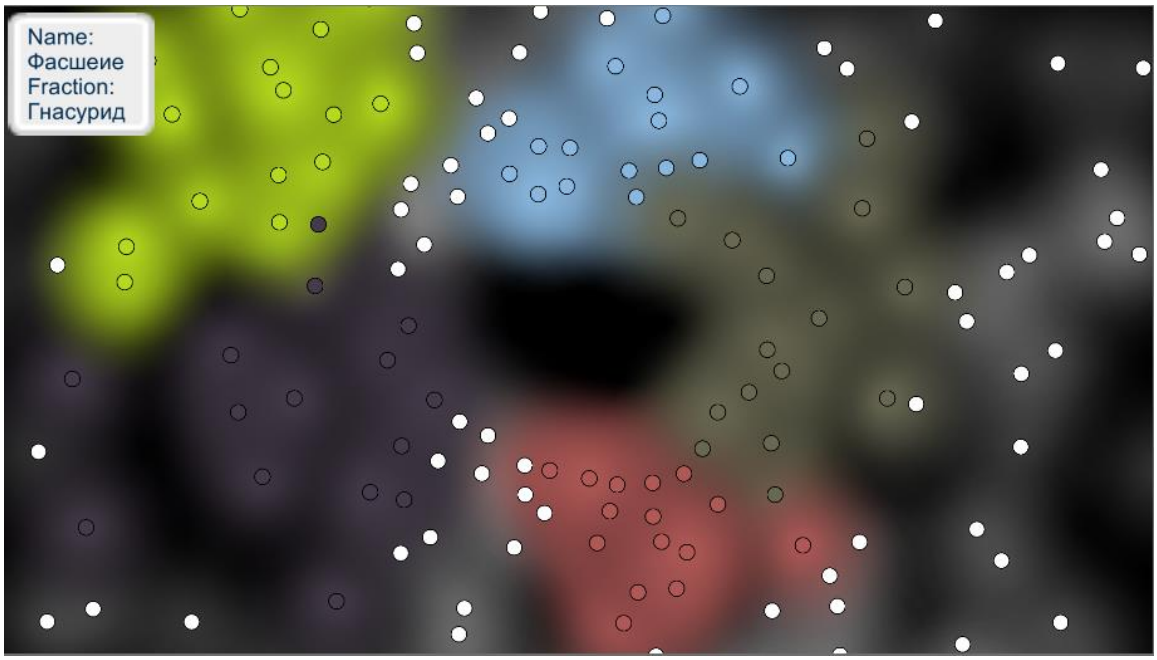

Рис. 7. Визуализация карты.

Данная система состоит из нескольких классов.

MapPoint – класс, отвечающий за хранение информации о системе и взаимодействие с ней. Из данных классов состоит карта игры, представленная на рисунке 8.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


[Serializable]
public class MapPoint : Target
{
    public Vector2 position { get; set; }
    public Fraction fraction;
    public string namePoint;
    public List<Station> ports { get; set; }
    public Color color { get; set; }

    public List<Person> people {get; set;}


    public Station GetStation()
    {
        return null;
    }

    public MapPoint GetMapPoint()
    {
        return this;
    }

    public Person GetPerson()
    {
        return null;
    }
}
```

Рис. 8. Класс MapPoint

MapViewPoint – класс, который отвечает за визуализацию класса MapPoint в среде Unity.

Map – это класс, отвечающий за взаимодействие с картой, генерацию и хранение классов MapPoint, а также построение маршрутов между точками.

MapSettings – класс, хранящий настройки карты. Содержит основные настройки для генерации карты.

Генерация карты – это многоступенчатый процесс создания игровых локаций, фракций и станций. На первом этапе происходит создание звёздных систем, определяется их положение и название. Игра имеет множество настроек по созданию карты. Такие настройки как количество систем, расстояние между системами, распределение систем по удалённости от центра галактики влияют непосредственно на первый этап создания карты. На рисунке 9 приведён первый этап создания карты.

```
List<MapPoint> GenrationMap(int Side,int Number)
    {
        List<MapPoint> points = new List<MapPoint>();

                  for (int i = 0;i<Number; i++)
        {
            Vector2 pos = new Vector2();
            for (int j = 0; j < 500; j++)
            {
                pos = RandomPos();

                if (Check(pos, points))
                {
                    break;
                }

            }
            var comp = new MapPoint();
            comp.position = pos;
            comp.namePoint = GenerateNameSys();
            points.Add(comp);
        }

        return points;
    }
```

Рис. 9. Первый этап генерации

На втором этапе создаются различные фракции и выбираются системы, которые они контролируют. Генератор пытает подобрать место для фракции чтобы расстояния между системами фракции было не очень большое генерация фракций и выбор систем указан на рисунке 10. Фракции размещаются в центре карты, тогда как по краям появляются пираты. На генерацию фракции влияют параметры количество фракций, количество планет фракций.

```
void SetFraction(List<MapPoint> map,List<Fraction> fractions)
    {
        List<MapPoint> mapNear = new List<MapPoint>();

        foreach (MapPoint point in map)
        {
            if (Vector2.Distance(point.position,Vector2.zero) < Set-
tings.rangeSize * MapSettings.ZONE)
            {
                mapNear.Add(point);
            }
        }

        foreach (Fraction fraction in fractions)
        {
            for (int i = 0; i < Settings.population;i++)
            {
                if (mapNear.Count > 0)
                {
                    if (fraction.myStars.Count == 0)
                    {
                        int num = Random.Range(0, mapNear.Count -
1);
                        mapNear[num].fraction = fraction;
                        mapNear[num].color = (Col-
or)fraction.colorFraction;
                        fraction.myStars.Add(mapNear[num]);
                        mapNear.RemoveAt(num);

                    }
                    else
                    {
                        int num = Random.Range(0, frac-
tion.myStars.Count - 1);
                        MapPoint mapPoint = fraction.myStars[num];

                        num = FindNearPlanet(mapNear, mapPoint);
                        mapNear[num].fraction = fraction;
                        mapNear[num].color = fraction.colorFraction;
                        fraction.myStars.Add(mapNear[num]);
                        mapNear.RemoveAt(num);
                    }
                }
                else
                {
                    break;
                }

            }
        }
    }
```

Рис. 10. Создание фракций

Следующий этап генерации мира - это создания планет и станций в звёздных системах. На мир наложено ограничение, что планеты могут появиться только в системах с фракциями. В каждой системе создаётся от

одного до трёх планет или станций. Планеты от станций отличаются лишь населением, которое влияет на выработку и трату ресурсов. Генератор создаёт различные типы планет в определённом соотношении. Иначе будет возникать глобальный дисбаланс по производству и потреблению ресурсов. На рисунке 11 представлен код данного метода.

Person – класс, хранящий данный о персонаже управляемом ИИ. Может торговать перемещаться и улучшать свой корабль.

PersonManager – класс, который отвечает за функционал активностей. Следит за состоянием и количеством персонажей.

Activity – это структура, хранящая данные о действиях персонажа.

Target – это интерфейс для получения необходимых данных об объекте. Данный интерфейс содержит методы доступа к определённым объектам, таким как класс MapPoint, Person и Station.

```
void GenerateStations()
    {
        int station = 0;
        //population
        List<int> stationNumType = new List<int>() {0 ,0 ,0 ,0 ,0
,0};
        foreach (MapPoint point in points)
        {
            point.ports = new List<Station>();
            int num = 1;
            if (point.fraction == null)
            {
                num = Random.Range(1, 3);
                for (int i = 0; i < num; i++)
                {
                    bool isPlanet = Random.Range(0, 100) > 50;
                    TypeStation type;
                    if (isPlanet)
                    {
                         type = (TypeStation.Close);
                    }
                    else
                    {
                        int n = findMinimumAr-
ray(stationNumType.ToArray());
                        type = (TypeStation)n;
                        stationNumType[n] = stationNumType[n] + 1;
                    }
                    point.ports.Add(new Station(Settings.Seed,
type, isPlanet, Random.Range(1, 6 - i), i, GenerateNameSta-
tion(point, isPlanet, type)));
                }
            }
            else
            {
                num = Random.Range(1, 5);
                for (int i = 0; i < num; i++)
                {
                    bool isPlanet = Random.Range(0, 100) > 50;
                    TypeStation type;
                    if (isPlanet)
                    {
                        int n = findMinimumAr-
ray(stationNumType.ToArray());
                        type = (TypeStation)n;
                        stationNumType[n] = stationNumType[n] + 1;
                    }
                            else
                    {
                        int n = findMinimumAr-
ray(stationNumType.ToArray());
                        type = (TypeStation)n;
                        stationNumType[n] = stationNumType[n] + 1;
                    }
                    point.ports.Add(new Station(Settings.Seed,
type, isPlanet, Random.Range(1, 6 - i), i, GenerateNameSta-
tion(point, isPlanet, type)));
                }
            }
        }
```

Рис. 11. Создание станций

3.3. Планеты и станции

Планеты и станции – включают в себя систему торговли и модификации корабля. Система торговли даёт возможность игроку покупать и продавать ресурсы на планетах. Данная механика служит как для получения игроком прибыли, так и создаёт видимость живого мира с изменяющимися ценами рисунке 13.

Система модификации - это система по улучшению вашего корабля. С её помощью вы можете не только создать более эффективный корабль, но и настроить корабль под себя путём установки одного из пяти видов модулей. Максимальное количество модулей зависит от размера корабля и размера модулей. Система модулей предполагает необходимость соблюдение определённых правил. Необходимо следить за количеством энергии и свободным местом. Не допускается понижение этих параметров до отрицательного значения. Сам корабль так же является модулем, но в отличие от остальных модулей его наличие обязательно.

Данная система состоит из классов:

Item – класс хранит основную информацию о модуле и ссылку на него, представленную на рисунке 12.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Item
{
    float weight;
    float size;
    float price;
    float energy;
    TypeModule type;
    Module module;
}
```

Рис. 12. Класс Item

Module – базовый компонент, дающий доступ к разному функционалу модулей.

Inventory – класс, хранящий данные о содержащихся предметах у игрока или станции.

Station – класс, реализующий функционал магазина с товарами, представлен на рисунке 13. Он хранит данные о товарах и ресурсах, которые может купить игрок. Каждая станция постоянно тратит одни ресурсы и производит другие (рисунок 15). В зависимости от количества ресурсов на станции меняются цены на них (представлены на рисунке 16). Данный класс отвечает за модификацию кораблей (рисунок 14).

Рис. 13. Интерфейс покупки

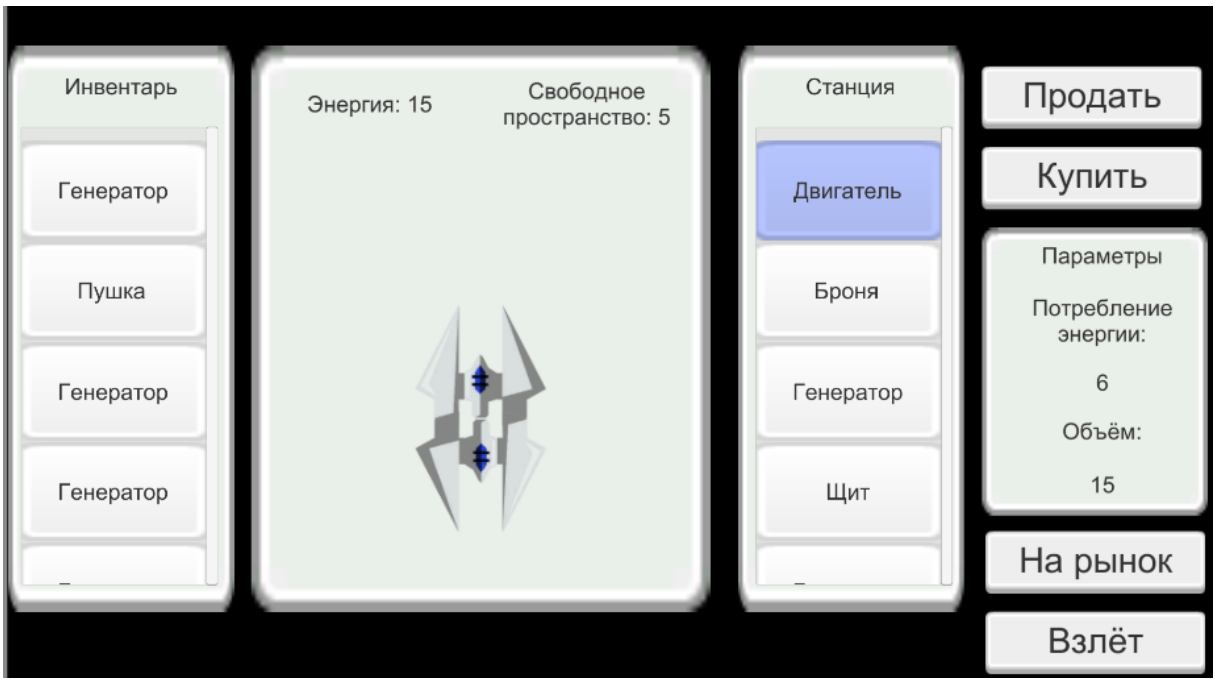

Рис. 14. Интерфейс модификации

StationsManager – следит за состоянием станций, при необходимости отправляет персонажа с ресурсами к станции. Контролирует обновление предметов и ресурсов.

```
blic MapPoint starSystem;
blic float speed;
blic float distToStar;
blic static float time;
erializeField]
blic TypeStation Type;
blic bool isPlanet;
blic bool active;
erializeField]
blic string name;

blic Vector3 position;
blic float techLevel = 0;//Уровень технологий
blic float dTechLevel = 0;//Уровень разницы технологий

atic readonly float[] basePrice = new float[] {15, 30, 45, 30, 45,

st<Item> items = new List<Item>();//Предметы в продаже
st<float> partsPrice = new List<float>();//Цена на ресурсы
st<int> partsCount = new List<int>();//Количество ресурсов
st<float> partsProdaction = new List<float>();//Производство
oat population = 1;
```

Рис. 15. Данные в классе Station

```
    public void UpdateItems()
    {
        items.Clear();
        for (int i =0; i< UnityEngine.Random.Range(0, 15); i++)
        {
            items.Add(new Item());
        }
    }

    void UpdateTrade()
    {
        for (int i = 0; i < partsProdaction.Count; i++)
        {
            partsPrice[i] = Mathf.Clamp((basePrice[i]) /
((partsCount[i] / (population * Mathf.Abs(partsProdaction[i]) *
20))) * UnityEngine.Random.Range(0.95f, 1.05f), basePrice[i] *
0.3f, basePrice[i] * 1.7f);
        }
    }

    void UpdateProdaction()
    {
        for (int i = 0; i < partsProdaction.Count; i++)
        {
            partsCount[i] += (int)((partsProdaction[i] * popula-
tion) * UnityEngine.Random.Range(0.9f, 1.1f));
            partsCount[i] = (int)(Mathf.Clamp(partsCount[i], 0,
(partsProdaction[i] * population * UnityEngine.Random.Range(90,
100))));
        }

        UpdateTrade();
    }
```

Рис. 16. Методы обновления ассортимента.

3.4. Открытый космос

Открытый космос – это система включает в себя путешествие по системе и сражения с ИИ. Вы можете встретить несколько видов поведения ИИ. Агрессивный будет нападать на вас при удобно случае. Нейтральный будет нападать только в ответ на ваше нападение, и пассивный будет пытаться сбежать.

Система включает в себя:

Gun – компонент, реализующий механику оружия. Хранит в себе необходимые параметры для стрельбы и создаёт пули по команде.

Bullet – класс, реализующий механику пули. Хранит данные об уроне и скорости. Передвигается в заданную точку и наносит урон при столкновении с врагом.

Harbour – класс реализующий, возможность посадки на станции или планеты.

SpaceObject – базовый класс, хранящий информацию об объектах в системе.

SpaceShip – компонент управления кораблём. Имеет параметры скорости и выживаемости, такие как жизни, броня и щит. Игрок или Ии может управлять им и передвигаться по системе. При значении жизней корабля ниже нуля он уничтожается, и игрок проигрывает. Сам корабль продемонстрирован на рисунке 17.

SpaceAi – Ии корабля, определяет поведение корабля.

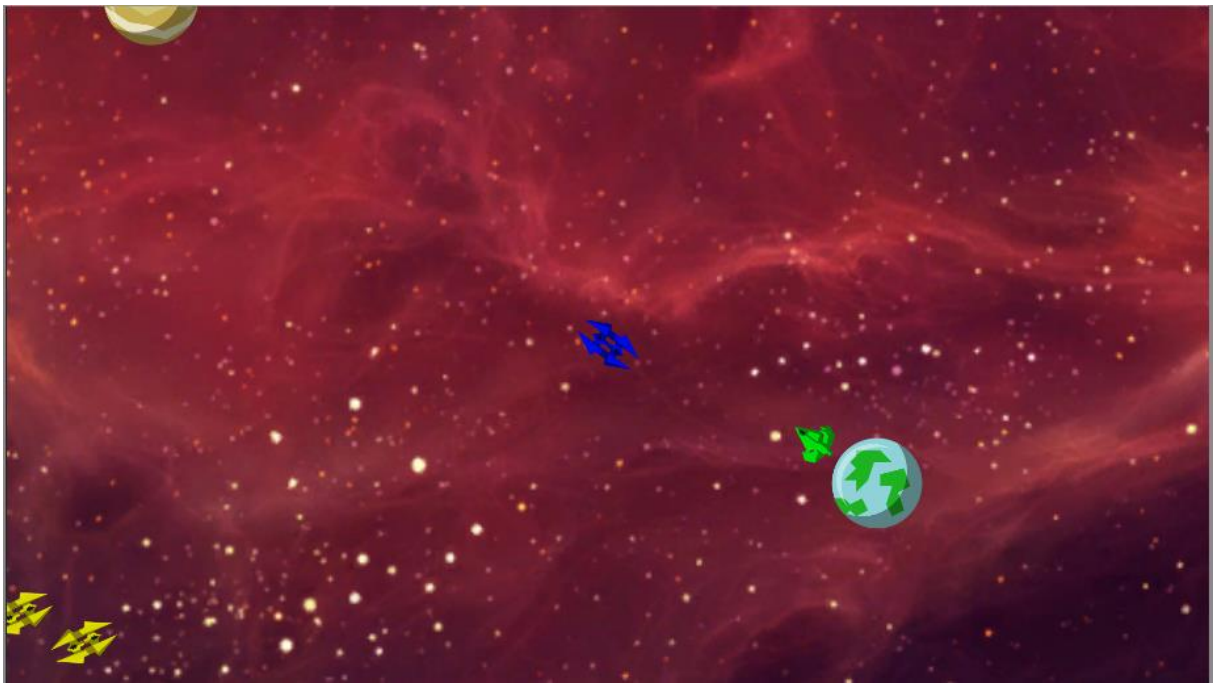

Рис.17. Полёт в системе

3.5. Общие

В общих содержатся классы, участвующие во многих процессах, но не имеющие строгой привязки к какой-то конкретной системе. Такие классы как DictionaryName.

DictionaryName – это класс, генерирующий и резервирующий имена для персонажей, планет, станций, систем и фракций. Она следит за недопущением повторов в названиях рисунке 18.

```
public string GetSystemName()
{
    string newName = "";
    int num = Random.Range(0, namesSystemsLoad.Count - 1);
    newName = namesSystemsLoad[num];
    return newName;
}

public string GetPersonName()
{
    string newName = "";
    int num = Random.Range(0, namesPersonsLoad.Count - 1);
    newName = namesPersonsLoad[num];
    namesPersonsLoad.RemoveAt(num);

    return newName;
}

public string GetPlanetName()
{
    string newName = "";
    int num = Random.Range(0, namesPlanetsLoad.Count - 1);
    newName = namesPlanetsLoad[num];
    namesPlanetsLoad.RemoveAt(num);

    return newName;
}

public string GetFractionName()
{
    string newName = "";
    int num = Random.Range(0, namesFractionsLoad.Count - 1);
    newName = namesFractionsLoad[num];

    return newName;
}
```

Рис. 18. Методы генерации имени

Menu – содержит основные методы для контроля графического интерфейса на всех этапах игры. Меню приведено на рисунке 19. Меню контролирует нажатия игрока на кнопки и отслеживает нажатия на клавиши. Данная система необходима как для работы пользовательского интерфейса, так и для работы некоторых функций.

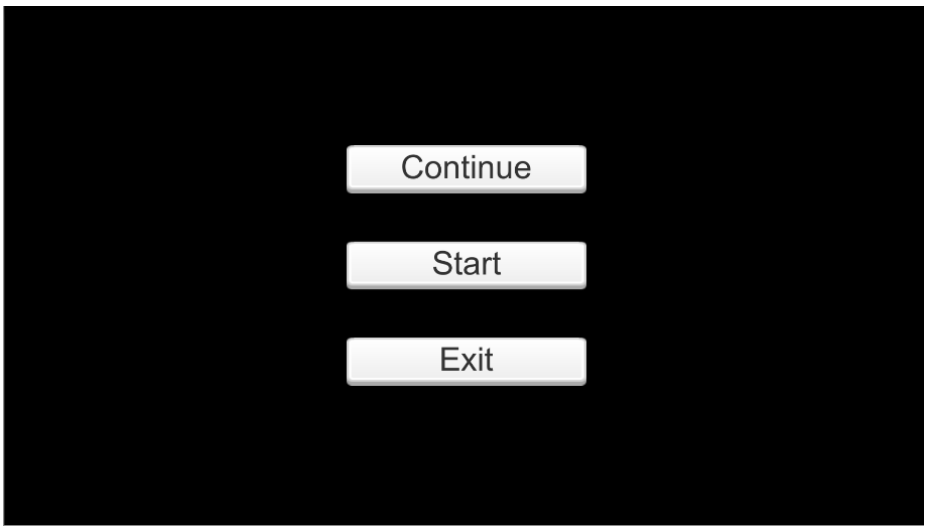

Рис. 19. Главное меню

Вывод

Реализация подобного приложения требует больших трудозатрат. Для качественного и полного создания подобных приложений требуется не только много времени, но и профессиональная команда разработки, включающая в себя не только программистов, но и дизайнеров, сценаристов, композиторов-музыкантов. Тем не менее, при использовании сторонних ресурсов реализовать данный проект на базовом уровне можно и в одиночку.

4.ТЕСТИРОВАНИЕ

4.1. Функциональное тестирование

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных условиях решать задачи, нужные пользователям.

Функциональные требования определяют, что именно делает программное обеспечение, и какие задачи оно решает.

По результатам функционального тестирования была составлена таблица с результатами.

Табл. 1. Функциональное тестирование

№	Название теста	Ожидаемый результат	Полученный результат	Тест пройден?
1	Отображение главного меню игры	На экране отображается главное меню игры с кнопками «Continue», «Start», «Exit».	На экране отображается главное меню игры с кнопками «Continue», «Start», «Exit».	Да
2	Работа кнопки «Start»	При нажатии кнопки «Start» начинается игровой процесс.	При нажатии кнопки «Start» начинается игровой процесс.	Да
3	Нажатие клавиши «W»	При нажатии клавиши «W» корабль ускоряется.	При нажатии клавиши «W» корабль ускоряется.	Да
4	Нажатие клавиши «E» рядом с планетой	Корабль садится на планету. Экран заменяется на экран станции.	Корабль садится на планету. Экран заменяется на экран станции.	Да
5	Нажатие клавиши «D».	Корабль поворачивает на право.	Корабль поворачивает на право.	Да
6	Ожидание на одном месте.	Мимо корабля пролетают различные другие корабли.	Мимо корабля пролетают различные другие корабли.	Да
7	Нажатие клавиши «A».	Корабль поворачивает на влево.	Корабль поворачивает на влево.	Да
8	Ожидание на планете.	Количество ресурсов и цены меняется.	Количество ресурсов и цены меняется.	Да

Окончание табл. 1

№	Название теста	Ожидаемый результат	Полученный результат	Тест пройден?
9	Нажатие на другую систему.	Корабль перемещается в другую систему.	Корабль перемещается в другую систему.	Да
10	Нажатие левую клавишу мыши.	Корабль стреляет в указанную сторону.	Корабль стреляет в указанную сторону.	Да
11	Нажатие клавиши «S».	Корабль замедляется.	Корабль замедляется.	Да

4.2. Юзабилити-тестирование

Юзабилити-тестирование – это способ тестирование, при котором определяется удобство, понятность и привлекательность продукта для-пользователя. Выполняется с помощью привлечения пользователей с целью прохождения тестов, с целью получения в дальнейшем информации и выводов от тестировщиков.

Участникам юзабилити-тестирования было необходимо выполнить следующие задачи:

1)начать игру;

2)выйти из игры;

3)сесть на станцию;

4)посмотреть цены;

5)выстрелить;

6)заменить оборудование.

Все задачи были выполнены успешно, без проблем. Юзабилити-тестирование пройдено успешно. Количество участников тестирование 10 человек.

Вывод

Для реализованного игрового приложения было проведено функциональное и юзабилити-тестирование. Все тесты были выполнены успешно, проблем выявлено не было.

ЗАКЛЮЧЕНИЕ

На базе игрового движка Unity было разработано приложение.

При этом были решены следующие задачи:

1) проанализированы различные игровые механики и способы создания игровых миров;

2) спроектировано и реализовано игровое приложение;

3) использована библиотека Unity для реализации базовых возможностей игрового приложения;

4) приложение было протестировано на наличие критических ошибок.

ЛИТЕРАТУРА

1.AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. [Электронный ресурс] URL:

https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii (дата обращения: 29.05.2020).

2.Project Discovery - Citizen science. [Электронный ресурс] URL: https://www.eveonline.com/ru/discovery(дата обращения: 11.04.2020).

3.Project Discovery - Citizen science. [Электронный ресурс] URL: https://www.unrealengine.com/ (дата обращения: 15.04.2020).

4.Open-World Games Are Changing the Way We Play. [Электронный ресурс] URL: https://www.wired.com/2015/12/open-world-games-2015/ (дата обращения: 25.05.2020).

5.Björk S., Lundgren S., Holopainen J. Game Design Patterns 2003.

6.Ciardiello T. Unity 2005.

7.Halpern J. Developing 2D Games with Unity / J. Halpern, 2019.

8.Hunicke R., Leblanc M., Zubek R. MDA: A formal approach to game design and game research 2004.

9.Lewis J.P., McGuire M., Fox P. Mapping the mental space of game genres 2007.

10.Mauger V. Interface 2014.

11.Neumann J. von, Morgenstern O. Theory of games and economic behavior / J. von Neumann, O. Morgenstern, 2007.

12.Rouse R. Game design 2014.

13.Shemmings D. Game // Community Care. 2011.

14.Unity UNITY: Store // Unity Technologies.

15.Unity Technologies Game engine, tools and multiplatform // Unity Technologies.

16.Unity Technologies Unity Manual // 5.3-N. 2016.

17.Unity Technologies Unity Documentation // 2019.2-003D.

18.Unity Technologies [и др.]. Unity Manual / Unity Technologies, J. Halpern, Unity Technologies, R. Rouse, R. Leonard [и др.]., 2019.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А - UML диаграмма классов

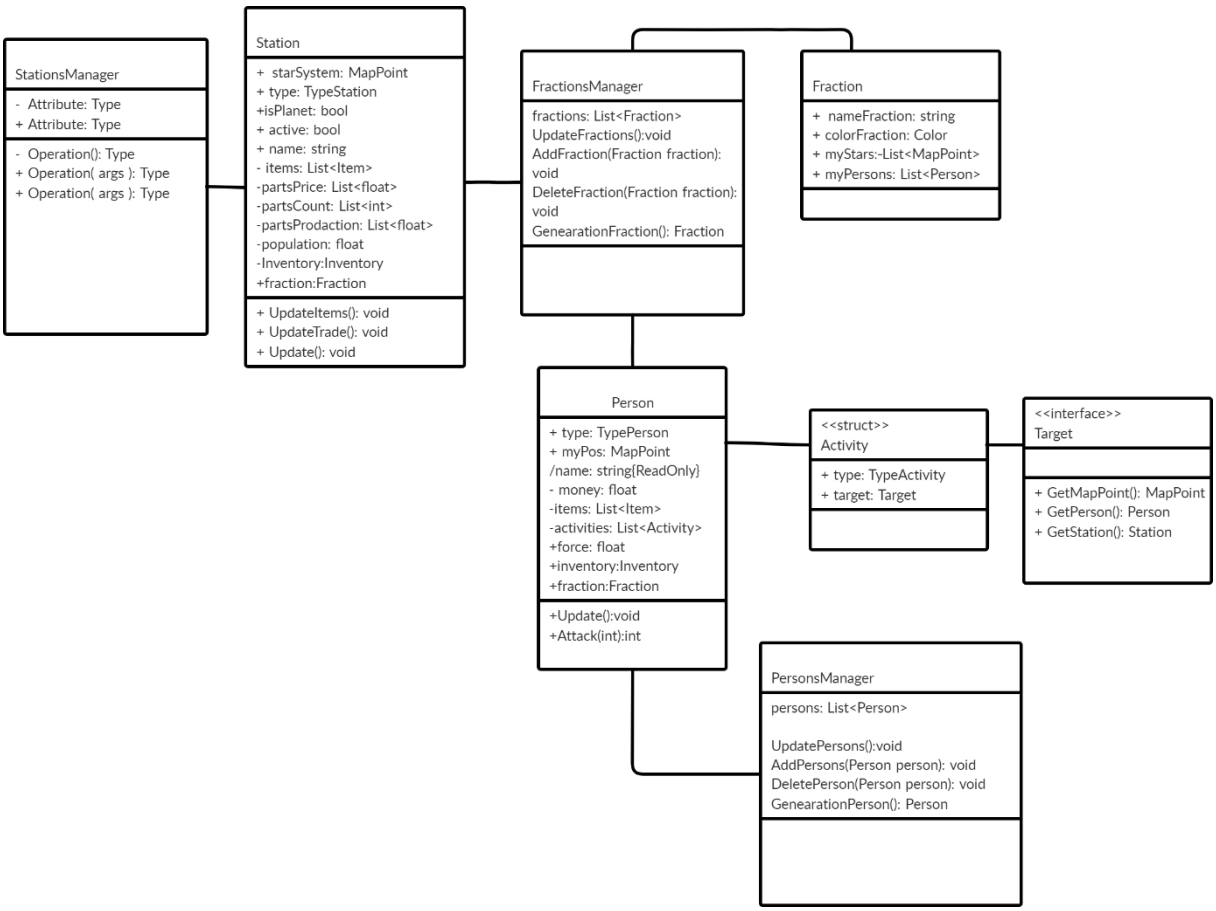

Рис. А.1. UML диаграмма классов блока живой мир

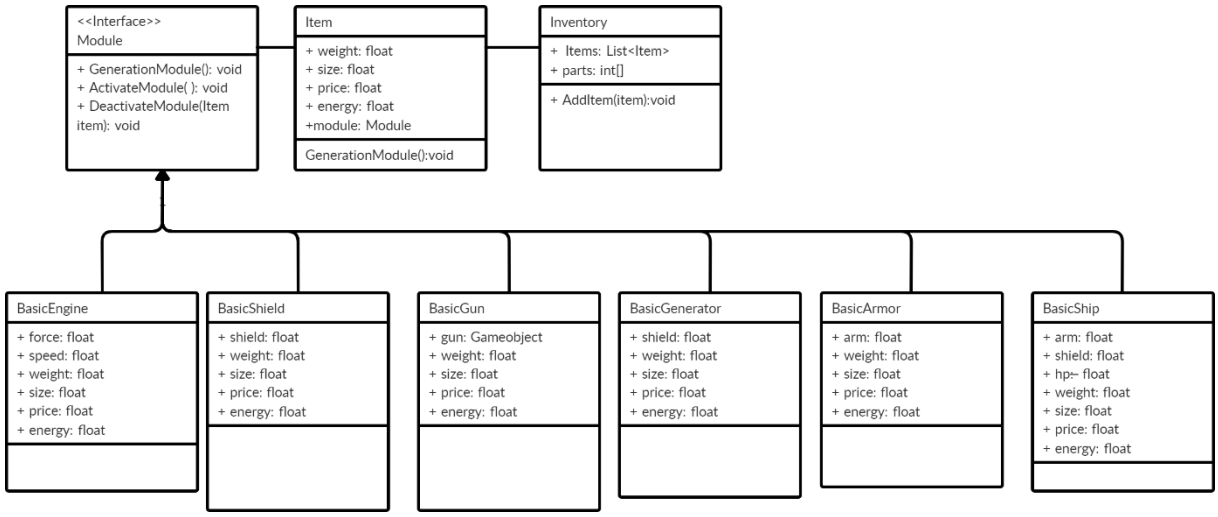

Рис. А.2. UML диаграмма классов инвентаря

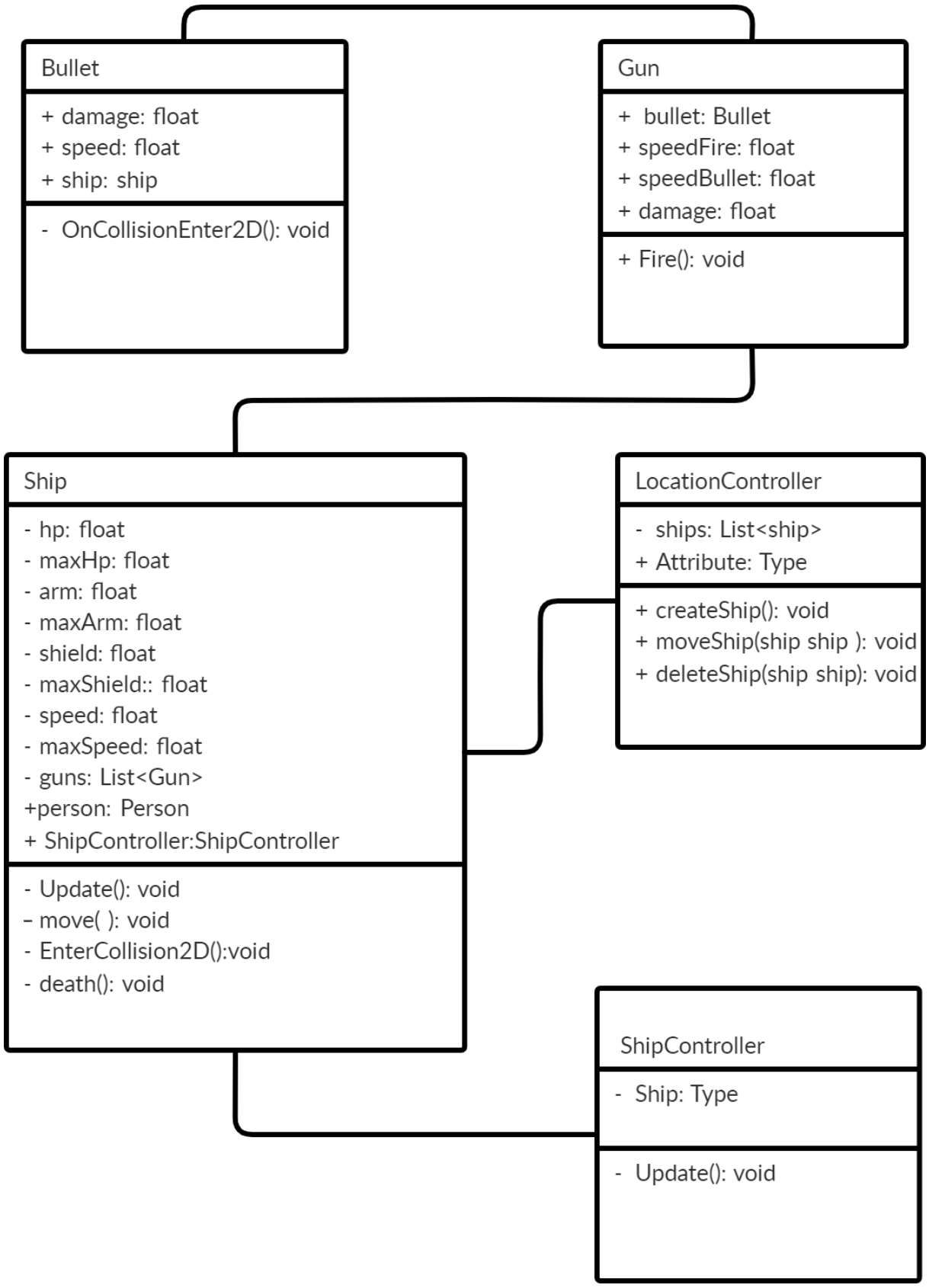

Рис. А.3. UML диаграмма классов блока космоса

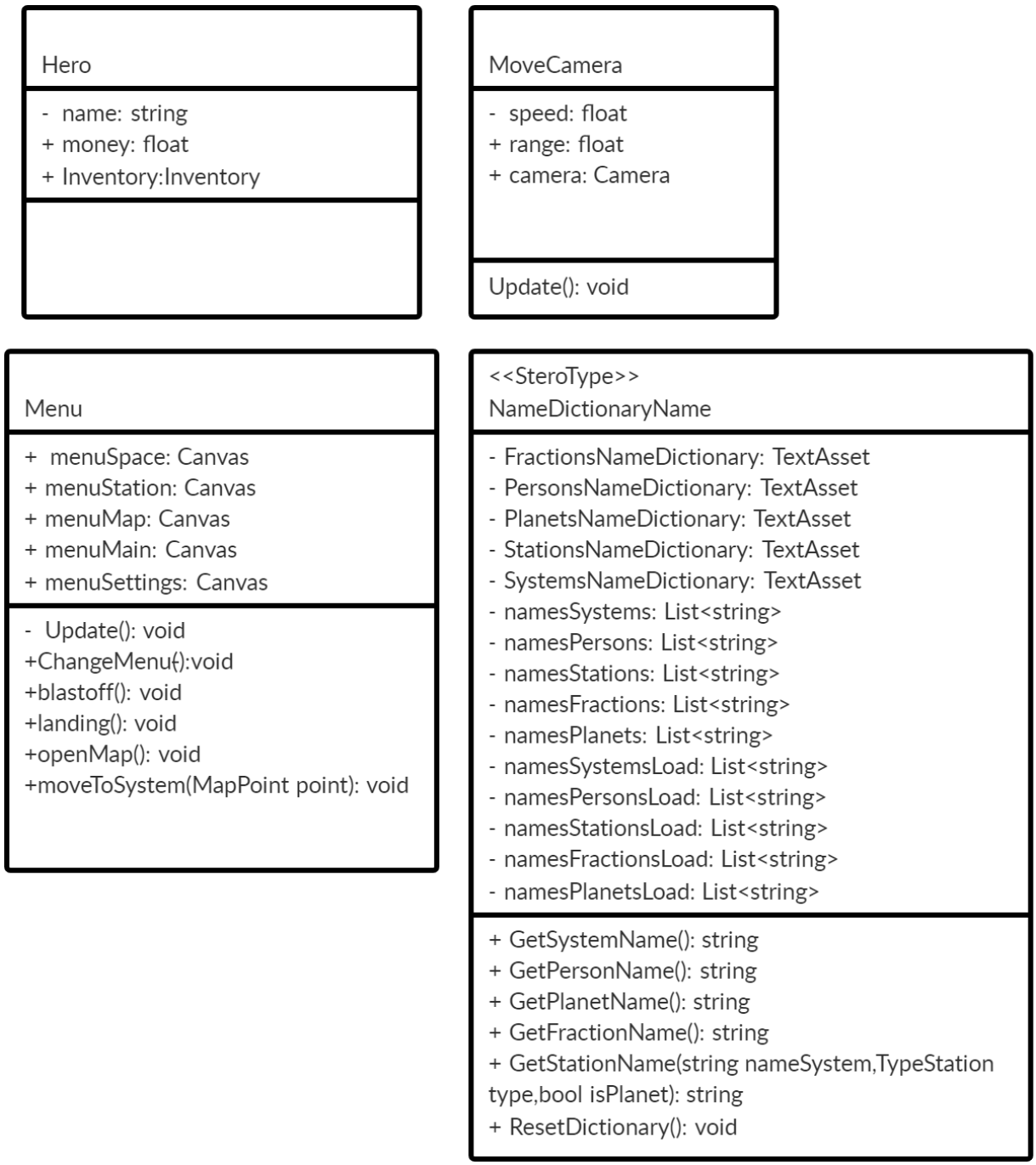

Рис. А.4. UML диаграмма классов общего блока