|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и система управления

КАФЕДРА Информационная безопасность

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***Разработка игры***

***в жанре RTS – «Планетные захватчики»***

Студент ИУ8Ц-51 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Кардаш К. Г.

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2019 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой ИУ8

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Басараб М.А.

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине «Технологии и методы программирования»

Студент группы ИУ8Ц-51

Кардаш Кирилл Григорьевич

(Фамилия, имя, отчество)

Тема курсового проекта «Разработка игры “Планетные захватчики”»

Направленность КП (учебный, исследовательский, практический, производственный, др.)

*учебный*

Источник тематики (кафедра, предприятие, НИР)

*кафедра*

График выполнения проекта: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

***Задание:*** Разрабатываемая игра – стратегия в реальном времени. В начале игры появятся некоторое количество неколонизированных планет и две империи, расположенных отдаленно друг от друга и имеющих по 1 планете. Колонизированные планеты производят корабли во времени. Корабли можно отправить с планеты на планету для колонизации или захват планеты. Игра заканчивается, если одна из империй утратила все планеты.

***Оформление курсового проекта:***

Расчетно-пояснительная записка на 2-х листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания «1» сентября 2019 г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бородин А.А.

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Кардаш К.Г.

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

[Введение 4](#_Toc27176692)

[Основные определения 4](#_Toc27176693)

[Требование к проекту 5](#_Toc27176694)

[Проектирование игры 5](#_Toc27176695)

[Механика игры 6](#_Toc27176696)

[Выбор игрового движка 24](#_Toc27176697)

[Описание технических решений 25](#_Toc27176698)

[Список используемых источников 26](#_Toc27176699)

## Введение

Цель проекта заключается в создании двумерной игры в жанре RTS в Unity под MasOS и Windows для ознакомления с игровым движком Unity и освоения навыков разработки двумерной игры.

Жанр RTS (real-time strategy) в игре довольно популярен среди современных жанров компьютерных игр.

Данная игра называется «Planet Invaders» либо «Планетные захватчики», представляет собой простое управление и рассчитана как стратегия в реальном времени, то есть в ней отсутствует очередность ходов. Игра является одиночной и подходит для любителей данного жанра и любителей космической экспансии.

Задача игрока – захватить как можно больше нейтральных или вражеских планет. Игрок станет победителем, если захватит все планеты соперника. Или, другими словами, игра заканчивается, если одна из двух империй утратила контроль над своими планетами и у неё больше нет кораблей для экспансии.

## Основные определения

* Unity – межплатформенная среда разработки компьютерных игр
* RTS (real-time strategy) – стратегия в реальном времени. Она является жанром стратегических компьютерных игр, в которых отсутствует очередность ходов
* ПО ­– программное обеспечение
* ОС ­­– операционная система
* PC (personal computer) – персональный компьютер
* Ассеты — это элементы, которые представляют собой графику, звуковое сопровождение или скрипты. Они прикрепляются к объектам и составляют важную часть игры.
* Префаб - это особый тип ассетов, позволяющий хранить весь GameObject со всеми компонентами и значениями свойств. Префаб выступает в роли шаблона для создания экземпляров хранимого объекта в сцене. Любые изменения в префабе немедленно отражаются и на всех его экземплярах, при этом вы можете *переопределять* компоненты и настройки для каждого экземпляра в отдельности.
* Спрайт – двумерные графические объекты.

## Требование к проекту

* Отсутствие ошибок в исходном коде
* Отсутствие багов в игре
* Устойчивая работа в игровом процессе, т.е. отсутствие вылетов или лагов

## Проектирование игры

Перед созданием игры документация Unity была изучена. Unity позволяет создать приложения, работающие под более чем 20 различными ОС, включающими PC, игровые консоли, мобильные устройства, интернет-приложения и другие. Для написания кода игры этот движок использует скриптовый язык С#.

Логика игры реализуется с помощью 7 скриптовых файлов:

* класс «PlayScene», отвечающий за случайное размещение планет и стартовое расположение планет игрока и бота в начале игры, а также за условия окончания игры
* класс «PopulationCounter», обеспечивающий отображение размеров популяций на планетах
* класс «BehaviourScript», отвечающий за свойства планет и выбор игроком планет для отправки кораблей
* класс «Fleet», контролирующий производство кораблей игрока
* класс «Ship», управляющий перемещением кораблей игрока и изменением свойств планет после захвата
* класс «BotFleet», контролирующий производство кораблей бота и стратегию захвата планет ботом
* класс «BotShip», управляющий перемещением кораблей бота и изменением свойств планет после захвата.

## Механика игры

Создание игровой сцены

Изображение cцены и рисунок планеты были скачаны из Интернета, в редакторе Unity открыл их и добавил в папку /Assets/Scenes в нижнем окне Project. Затем перетащил изображение космоса для сцены на сцену и установил подходящий размер.

Создание спрайта корабля

Создал спрайт-треугольник, затем немного изменил форму.

Создание префабов планет

Добавил на сцену четыре 2D объекта Спрайт Planet1, Planet2, Planet3 и Planet4. Прикрепил к ним спрайт планеты. Изменил размер на нужный. Добавил компонент Circle Collider 2D. Добавил компонент скрипт BehaviourScript. Добавил внутренний пустой игровой объект PopulationCounter. К объекту PopulationCounter добавил компоненты TextMesh и скрипт PopulationCounter. Перетащил каждую из 4-х планет в панель Project.

Создание префабов кораблей

Добавил на сцену 2D объекты Спрайт Ship и botShip. Прикрепил к ним спрайт корабля. Добавил к объектам компоненты скрипт Ship и botShip соответственно. Для корабля бота поменял цвет рендера на красный, а для корабля игрока – на зелёный. Перетащил оба корабля в панель Project.

Создание префабов поражения и победы

Добавил на сцену пустые игровые объекты Lose и Win. Добавил к ним внутренний пустой игровой объект Text. К объекту Text добавил компонент TextMesh и поражения написал красным цветом слово «Поражение», а для победы – жёлтым цветом слово «Победа». Перетащил оба объекта в панель Project.

Создание игрового объекта флота игрока

Добавил на сцену пустой игровой объект Fleet. Добавил к нему компонент скрипт Fleet.

Создание игрового объекта флота бота

Добавил на сцену пустой игровой объект botFleet. Добавил к нему компонент скрипт botFleet.

Создание игрового объекта для управления сценой

Добавил на сцену пустой игровой объект PlayScene. Добавил к нему компонент - скрипт PlayScene.s

Описание работы скрипта BehaviourScript

public class BehaviourScript : MonoBehaviour

{

public int population; //популяция планеты

public bool owned\_by\_user; //принадлежность игроку

public bool owned\_by\_bot; //принадлежность боту

public bool neutral; //нейтральная планета

public float counter; //счётчик времени

private Ship ship; //создаём объект типа Ship для получения доступа к игровому объекту Ship по его имени

private Fleet planetFleet; //создаём объект типа Fleet для получения доступа к игровому объекту Fleet по его имени

//Старт вызывается до первого обновления кадра

void Start()

{

}

//Обновление вызывается один раз за кадр

void Update()

{

if ((owned\_by\_user == true) || (owned\_by\_bot == true)) { //для планет игрока и бота

counter += Time.deltaTime;

if (counter >= 1) //если прошла одна или более секунд

{

population += 1; //увеличиваем население

counter = 0; //сбрасываем счётчик

}

}

}

private void OnMouseDown() //при нажатии кнопки мыши на планету

{

planetFleet = GameObject.Find("Fleet").GetComponent<Fleet>(); //получаем доступ к игровому объекту Fleet

if (planetFleet.shipsOnScene > 0) //если на сцене есть корабли

{

ship = GameObject.Find(planetFleet.shipName).GetComponent<Ship>(); //получаем доступ к игровому объекту Ship по его имени

}

if (owned\_by\_user == true) //если планета принадлежит игроку

{

if (planetFleet.allShipsHaveDestination) //если все корабли имеют пункт назначения или нет кораблей вообще

{

planetFleet.size = population / 2; //берём половину населения планеты для отправки на новый корабль

population -= planetFleet.size; //уменьшаем население в 2 раза

planetFleet.position = gameObject.GetComponent<Transform>().position; //запоминаем позицию планеты, с которой мы взяли население

}

else //если есть корабли без пункта назначения

{

ship.destination = gameObject.GetComponent<Transform>().position; //назначаем короблю пункт назначения

ship.destinationName = gameObject.name; //запоминаем название планеты, на которую летит корабль

ship.readyForTravel = true; //говорим, что корабль готов к отправлению

planetFleet.allShipsHaveDestination = true; //говорим, что корабли во флоте имеют пункт назначения

}

}

if (owned\_by\_bot == true || neutral == true) //если планета принадлежит боту или нейтральная

{

ship.destination = gameObject.GetComponent<Transform>().position; //запоминаем позицию планеты, на которую отправим корабли

ship.destinationName = gameObject.name; //запоминаем название планеты, на которую летит корабль

ship.readyForTravel = true; //корабли готовы к отправлению

planetFleet.allShipsHaveDestination = true; //все корабли имеют пункт назначения

}

}

}

Описание работы скрипта PlayScene

public class PlayScene : MonoBehaviour

{

public GameObject planet1; //создаём пустой игровой объект для добавления планеты 1 на сцену

public GameObject planet2; //"-"

public GameObject planet3;

public GameObject planet4;

public GameObject win; //создаём пустой игровой объект для отображения "победа"

public GameObject lose; //создаём пустой игровой объект для отображения "поражение"

public int playerPlanets; //счётчик планет игрока

public int botPlanets; //счётчик планет бота

public int playerShips; //счётчик кораблей игрока

public int botShips; //счётчик кораблей бота

private BehaviourScript newPlanet; //создаем объект для доступа к свойствам планеты

private Transform planetPosition; // создаем объект для назначения координат планеты

private string planetName; //название планеты

private string botPlanetName; //название планеты бота

private string playerPlanetName; //название планеты игрока

private Vector3[] randPosition; //массив случайных координат планет

private Vector3 samplePosition; //временное значение координат планеты

private bool positionAccepted; //принятие положения планеты

// Start is called before the first frame update

void Start()

{

System.Array.Resize(ref randPosition, 12); //изменение размера массива randPosition на 12 планет

positionAccepted = true;

playerPlanets = 0; //в начале игры у нас нет планет игрока на сцене

botPlanets = 0; //в начале игры у нас нет планет бота на сцене

playerShips = 0; //в начале игры у нас нет кораблей игрока на сцене

botShips = 0; //в начале игры у нас нет кораблей бота на сцене

float max = 0f, min = 0f; //переменные для вычисления Манхэттенского растояния от центра сцены,

//чтобы выбрать какая планета будет принадлежать игроку, а какая - боту

for (int i = 0; i < 12; i++)

{

positionAccepted = true;

samplePosition.Set(Random.Range(-7.5f, 7.5f), Random.Range(-3.8f, 3.8f), 0); //случайный выбор временной координаты

if (i > 0) //если у нас уже есть первые координаты для планеты

{

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

{

//если разница между временными и уже имеющимися коодинатами

//по обоим осям меньше 2.5, то мы не принимаем такие координаты

if ((Mathf.Abs(samplePosition.x - randPosition[j].x) < 2.5) &&

(Mathf.Abs(samplePosition.y - randPosition[j].y) < 2.5))

{

positionAccepted = false;

}

}

if (positionAccepted)

{

randPosition[i] = samplePosition;//положим временные координаты в массив

} else

{

i--; //возвращаемся к предыдущему шагу выбора координыт

}

} else randPosition[i] = samplePosition; //запоминаем координаты для самой первой планеты

}

for (int i = 1; i < 13; i++)

{

switch (Random.Range(1, 5)) //переключатель для случайного выбора размера планеты

{

case 1:

Instantiate(planet1, randPosition[i-1], transform.rotation); //вставляем самую маленькую планету

planetName = "Planet" + i.ToString(); // создаём имя планеты с номером

GameObject.Find("Planet1(Clone)").name = planetName; // меняем имя вставленной планеты

break;

case 2:

Instantiate(planet2, randPosition[i-1], transform.rotation); //"-"

planetName = "Planet" + i.ToString();

GameObject.Find("Planet2(Clone)").name = planetName;

break;

case 3:

Instantiate(planet3, randPosition[i-1], transform.rotation);

planetName = "Planet" + i.ToString();

GameObject.Find("Planet3(Clone)").name = planetName;

break;

case 4:

Instantiate(planet4, randPosition[i-1], transform.rotation);

planetName = "Planet" + i.ToString();

GameObject.Find("Planet4(Clone)").name = planetName;

break;

default:

Instantiate(planet1, randPosition[i-1], transform.rotation);

planetName = "Planet" + i.ToString();

GameObject.Find("Planet1(Clone)").name = planetName;

break;

}

newPlanet = GameObject.Find(planetName).GetComponent<BehaviourScript>(); //получаем доступ к св-вам вставленной планеты

newPlanet.neutral = true; //по умолчанию все планеты нейтральные

}

for (int i = 1; i < 13; i++)

{

planetName = "Planet" + i.ToString(); // создаём имя планеты с номером

planetPosition = GameObject.Find(planetName).GetComponent<Transform>(); //запоминаем позицию планеты с именем planetName

//если манхэтенское расстояние больше текущего расстояния справа от центра

if (planetPosition.position.x + Mathf.Abs(planetPosition.position.y) > max)

{

max = planetPosition.position.x + Mathf.Abs(planetPosition.position.y); //перезаписываем максимум

botPlanetName = planetName; //запоминаем имя планеты бота

}

//если отрицательное манхетенское расстояние меньше текущего расстояния слева от центра

if (planetPosition.position.x - Mathf.Abs(planetPosition.position.y) < min)

{

min = planetPosition.position.x - Mathf.Abs(planetPosition.position.y); //перезаписываем минимум

playerPlanetName = planetName; //запоминаем имя планеты игрока

}

}

//передаём планету боту

newPlanet = GameObject.Find(botPlanetName).GetComponent<BehaviourScript>();

newPlanet.neutral = false;

newPlanet.owned\_by\_bot = true;

botPlanets += 1;

newPlanet.GetComponent<SpriteRenderer>().color = Color.red;

//передаём планету игроку

newPlanet = GameObject.Find(playerPlanetName).GetComponent<BehaviourScript>();

newPlanet.neutral = false;

newPlanet.owned\_by\_user = true;

playerPlanets += 1;

newPlanet.GetComponent<SpriteRenderer>().color = Color.green;

}

// Update is called once per frame

void Update()

{

//условие поражения игрока

if (playerShips == 0 && playerPlanets ==0)

{

Instantiate(lose, transform.position, transform.rotation); //отображаем "поражение" на сцене

Time.timeScale = 0; //останавливаем течение времени в игре

Application.Quit(); //выход из игры

}

//условие поражения бота

if (botShips == 0 && botPlanets == 0)

{

Instantiate(win, transform.position, transform.rotation); //отображаем "победа" на сцене

Time.timeScale = 0; //останавливаем течение времени в игре

Application.Quit(); //выход из игры

}

}

}

Описание работы скрипта PopulationCounter

public class PopulationCounter : MonoBehaviour

{

// Start is called before the first frame update

void Start()

{

GetComponent<TextMesh>().text = GetComponentInParent<BehaviourScript>().population.ToString(); //получим кол-во популяции для отображения

}

// Update is called once per frame

void Update()

{

GetComponent<TextMesh>().text = GetComponentInParent<BehaviourScript>().population.ToString(); //получим кол-во популяции для отображения

}

}

Описание работы скрипта Ship

public class Ship : MonoBehaviour

{

public int size; //количество населения на корабле

public Vector3 destination; //координаты пункта назначения

public string destinationName; //название планеты, на которую отправим корабли

public bool readyForTravel; //готовность кораблей отправиться

private float axisSpeed; //коэффициент скорости по оси

private BehaviourScript destinationPlanet; //создаём объект для взаимодействия с планетой назначения

private PlayScene playScene; //создаём объект типа PlayScene для работы с основными параметрами игры

// Start is called before the first frame update

void Start()

{

readyForTravel = false; //при создании корабля он ещё не готов к отправке

axisSpeed = Mathf.Abs( (destination.x - transform.position.x) / (destination.y - gameObject.transform.position.y));

}

// Update is called once per frame

void Update()

{

playScene = GameObject.Find("PlayScene").GetComponent<PlayScene>();

if (readyForTravel) //если готовы к отправлению

{

if (destination.x > gameObject.GetComponent<Transform>().position.x) //если х пункта назначения больше, чем х положения корабля

{

transform.position = transform.position + new Vector3((float)0.01, 0, 0); //увеличиваем значение координаты х корабля

}

else if (destination.x < gameObject.GetComponent<Transform>().position.x) //если х пункта назначения меньше, чем х положения корабля

{

transform.position = transform.position - new Vector3((float)0.01, 0, 0); //уменьшаем значение координаты х корабля

}

if (destination.y > gameObject.GetComponent<Transform>().position.y)

{

transform.position = transform.position + new Vector3(0, (float)0.01, 0);

}

else if (destination.y < gameObject.GetComponent<Transform>().position.y)

{

transform.position = transform.position - new Vector3(0, (float)0.01, 0);

}

}

if ((Mathf.Abs(destination.x - transform.position.x) <= 0.1 )

&& (Mathf.Abs(destination.y - transform.position.y) <= 0.1)) // если расстояние между положением корабля и планетой меньше или равно 0.1

{

destinationPlanet = GameObject.Find(destinationName).GetComponent<BehaviourScript>();//получаем доступ к игровому объекту планеты по имени

if (destinationPlanet.owned\_by\_user == true) // если планета принадлежит игроку

{

destinationPlanet.population += size; //отправим население с корабля на планету

}

if (destinationPlanet.owned\_by\_bot == true || destinationPlanet.neutral == true) //если планета принадлежит боту или нейтральная

{

if (destinationPlanet.population >= size) // если население на планете больше населения на корабле

{

destinationPlanet.population -= size; // уменьшаем население на планете

}

else // иначе

{

destinationPlanet.population = size - destinationPlanet.population; // заселяем планету населением с корабля за вычетом населения планеты

if (destinationPlanet.owned\_by\_bot == true) // если планета принадлежит боту

{

destinationPlanet.owned\_by\_bot = false; //забираем планету у бота

playScene.botPlanets -= 1;

destinationPlanet.owned\_by\_user = true; //отдаём планету игроку

playScene.playerPlanets += 1;

destinationPlanet.GetComponent<SpriteRenderer>().color = Color.green; //меняем цвет планеты на зёленый

}

if (destinationPlanet.neutral == true) // если планета нейтральная

{

destinationPlanet.neutral = false; //перестаёт быть нейтральной

destinationPlanet.owned\_by\_user = true; //отдаём планету игроку

playScene.playerPlanets += 1;

destinationPlanet.GetComponent<SpriteRenderer>().color = Color.green; //меняем цвет планеты на зёленый

}

}

}

Destroy(gameObject); //уничтожаем корабль

GameObject.Find("Fleet").GetComponent<Fleet>().shipsOnScene -= 1; //уменьшаем счетчик кол-ва кораблей на 1

playScene.playerShips -= 1;

}

}

}

Описание работы скрипта Fleet

ublic class Fleet : MonoBehaviour

{

public int size; //количество населения для отправки на корабль

public Vector3 position; //позиция планеты, с которой мы взяли население

public int shipsOnScene; //количество кораблей на сцене

public string shipName; // имя корабля

public bool allShipsHaveDestination; //все корабли имеют пункт назначения

public GameObject ship; //создаём пустой игровой объект для добавления кораблей на сцену

private Ship newShip; //создаём объект типа Ship для работы с новым вставленным на сцену кораблём

private int shipNumber; //порядковый номер корабля

private PlayScene playScene; //создаём объект типа PlayScene для работы с основными параметрами игры

// Start is called before the first frame update

void Start()

{

shipNumber = 0; //начинаем нумерацию кораблей с 0

shipsOnScene = 0; //в начале игры у нас нет кораблей на сцене

allShipsHaveDestination = true; //по умолчанию все корабли имеют пункт назначения

}

// Update is called once per frame

void Update()

{

playScene = GameObject.Find("PlayScene").GetComponent<PlayScene>();

if (size > 0 ) //если количество населения для отправки на корабль > 0

{

shipNumber += 1; //увеличиваем порядковый номер корабля

Instantiate(ship, position, transform.rotation); //вставляем префаб корабля на сцену

shipsOnScene += 1; //увеличиваем счётчик количества кораблей на сцене на 1

playScene.playerShips += 1;

allShipsHaveDestination = false; //не все корабли имеют пункт назначения

shipName = "Ship" + shipNumber.ToString(); //создаём новое имя корабля = ship + порядковый номер

GameObject.Find("Ship(Clone)").name = shipName; //задаем уникальное имя новому кораблю

newShip = GameObject.Find(shipName).GetComponent<Ship>(); //получаем доступ к игровому объекту по имени

newShip.size = size; //отправили население на корабль

size = 0; //обнуляем количество населения для отправки на корабль

}

}

}

Описание работы скрипта botShip

public class botShip : MonoBehaviour

{

public int size; //количество населения на корабле бота

public Vector3 destination; //координаты пункта назначения

public string destinationName; //название планеты, на которую бот отправляет корабли

public bool readyForTravel; //готовность кораблей бота отправиться

private float axisSpeed; //коэффициент скорости по оси

private BehaviourScript destinationPlanet; //создаём объект для взаимодействия с планетой назначения

private PlayScene playScene; //создаём объект типа PlayScene для работы с основными параметрами игры

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

playScene = GameObject.Find("PlayScene").GetComponent<PlayScene>();

if (destination.x > gameObject.GetComponent<Transform>().position.x) //если х пункта назначения больше, чем х положения корабля

{

transform.position = transform.position + new Vector3((float)0.01, 0, 0); //увеличиваем значение координаты х корабля

}

else if (destination.x < gameObject.GetComponent<Transform>().position.x) //если х пункта назначения меньше, чем х положения корабля

{

transform.position = transform.position - new Vector3((float)0.01, 0, 0); //уменьшаем значение координаты х корабля

}

if (destination.y > gameObject.GetComponent<Transform>().position.y)

{

transform.position = transform.position + new Vector3(0, (float)0.01, 0);

}

else if (destination.y < gameObject.GetComponent<Transform>().position.y)

{

transform.position = transform.position - new Vector3(0, (float)0.01, 0);

}

// если расстояние между положением корабля и планетой меньше или равно 0.1

if ((Mathf.Abs(destination.x - transform.position.x) <= 0.1)

&& (Mathf.Abs(destination.y - transform.position.y) <= 0.1))

{

destinationPlanet = GameObject.Find(destinationName).GetComponent<BehaviourScript>(); //получаем доступ к игровому объекту планеты по имени

if (destinationPlanet.owned\_by\_bot == true) // если планета принадлежит боту

{

destinationPlanet.population += size; //отправим население с корабля на планету

}

if (destinationPlanet.owned\_by\_user == true || destinationPlanet.neutral == true) //если планета принадлежит игроку или нейтральная

{

if (destinationPlanet.population >= size) // если население на планете больше населения на корабле

{

destinationPlanet.population -= size; // уменьшаем население на планете

}

else

{

destinationPlanet.population = size - destinationPlanet.population; // заселяем планету населением с корабля за вычетом населения планеты

if (destinationPlanet.owned\_by\_user == true) // если планета принадлежит игроку

{

destinationPlanet.owned\_by\_user = false; //забираем планету у игрока

playScene.playerPlanets -= 1;

destinationPlanet.owned\_by\_bot = true; //отдаём планету боту

playScene.botPlanets += 1;

destinationPlanet.GetComponent<SpriteRenderer>().color = Color.red; //меняем цвет планеты на красный

}

if (destinationPlanet.neutral == true) // есчли планета нейтральная

{

destinationPlanet.neutral = false; //перестаёт быть нейтральной

destinationPlanet.owned\_by\_bot = true; // отдаем планету боту

playScene.botPlanets += 1;

destinationPlanet.GetComponent<SpriteRenderer>().color = Color.red; //меняем цвет планеты на красный

}

}

}

Destroy(gameObject); //уничтожаем корабль

playScene.botShips -= 1;

}

}

}

Описание работы скрипта BotFleet

public class BotFleet : MonoBehaviour

{

public int size; //количество населения бота для отправки на корабль

public Vector3 position; //позиция планеты, с которой мы взяли население бота

public int botShipsOnScene; //количество кораблей бота на сцене

public string botShipName; // имя корабля бота

public bool allShipsHaveDestination; //все корабли бота имеют пункт назначения

public int totalPlanetesOnScene; //все планеты на сцене

public GameObject botShip; //создаём пустой игровой объект для добавления кораблей бота на сцену

private botShip newBotShip; //создаём объект типа newBotShip для работы с новым вставленным на сцену кораблём

private int botShipNumber; //порядковый номер корабля бота

private BehaviourScript botPlanet; //создаем объект для доступа к игровому объекту

private bool readyToAttack; //готовность атаковать

public BehaviourScript planetToAttack; //создаем объект для доступа к игровому объекту для атаки

private PlayScene playScene;//создаём объект типа PlayScene для работы с основными параметрами игры

// Start is called before the first frame update

void Start()

{

botShipNumber = 0;//начинаем нумерацию кораблей с 0

}

// Update is called once per frame

void Update()

{

playScene = GameObject.Find("PlayScene").GetComponent<PlayScene>();

int totalBotPlanetPopulation = 0; //обнуляем счётчик популяции на всех планетах бота

for (int i = 1; i <= totalPlanetesOnScene; i++)

{

botPlanet = GameObject.Find("Planet" + i.ToString()).GetComponent<BehaviourScript>();

if (botPlanet.owned\_by\_bot == true)

totalBotPlanetPopulation += botPlanet.population; //прибавляем население планеты к счётчику, если это планета бота

}

for (int i = 1; i < totalPlanetesOnScene + 1; i++)

{

planetToAttack = GameObject.Find("Planet" + i.ToString()).GetComponent<BehaviourScript>();

//если планета не принадлежит боту и половина всеобщего количества населения на планетах бота больше чем

//население на выбранной планете

if (!planetToAttack.owned\_by\_bot && totalBotPlanetPopulation / 2 > planetToAttack.population + 1)

{

readyToAttack = true; // выставляем готовность атаковать

break;

}

}

if (readyToAttack) //если бот готов атаковать

{

for (int i = 1; i < totalPlanetesOnScene + 1; i++)

{

botPlanet = GameObject.Find("Planet" + i.ToString()).GetComponent<BehaviourScript>();

if (botPlanet.owned\_by\_bot == true) //если планета принадлежит боту

{

botShipNumber += 1; //увеличиваем номер корабля на 1

Instantiate(botShip, botPlanet.transform.position, transform.rotation); //вставляем корабль бота на сцену

botShipsOnScene += 1; //надо проверить, нужно ли?

playScene.botShips += 1;

botShipName = "botShip" + botShipNumber.ToString(); //создаем имя корабля бота

GameObject.Find("botShip(Clone)").name = botShipName; //меняем имя

newBotShip = GameObject.Find(botShipName).GetComponent<botShip>(); //получаем доступ к параметрам нового корабля

newBotShip.size = botPlanet.population / 2; // размер населения на корабле бота равно половине населения на планете бота

botPlanet.population -= newBotShip.size; //вычитаем размер населения на корабле из населения на планете

newBotShip.destination = planetToAttack.transform.position; //передаем пункт назначения

newBotShip.destinationName = planetToAttack.name; //передаем имя пункта назначения

}

}

readyToAttack = false; //отсутствие готовности атаковать

}

}

}

## Выбор игрового движка

Рассмотрим игровой движок Unity. У него много преимуществ:

* Доступность массы ресурсов
* Удобство развертывания в мобильной среде и поддержка более чем 20 ОС
* Основательная документация и обучающие видео
* Популярность (широкое сообщество) и множество ответов на вопросы
* Легкое освоение
* Быстрая компиляция
* Быстрое устранение ошибок и недоработок

Имеет недостатки:

* Закрытость исходного кода
* Занимает большую память в PC

Теперь рассмотрим преимущества Unreal Engine:

* Общая интеграция с С++
* Открытость исходного языка
* Лучшая графика
* Мультиплатформенность

Имеет больше недостатков в сравнении Unity:

* Огромная и сложная документация
* Сложность освоения для начинающихся и менее опытных разработчиков
* Медленная компиляция
* Слабое развертывание ПО и медленная мобильная разработка с Unreal.

Подведем итоги. Для нашего проекта лучше всего использовать игровой движок Unity, так как он подходит для начинающихся и легко освоится за короткое время, имеет множество обучающих литературу и видео. Также для него легче находить нужные ответы на вопросы в форумах.

## Описание технических решений

Во время выполнения данного проекта возникло несколько проблем.

1. Проблема в связи с трудным выбором игрового движка. На ознакомление с каждым из популярных игровых движков ушла масса времени. Благодаря списку преимуществ и недостатков между вышеупомянутыми игровыми движками выбор лежал за Unity.
2. Также возникшая проблема в процессе разработки заключается в том, что после размещения планеты накладывались друг на друга. Чтобы устранить эту проблему, нужно установить минимальную разницу между координатами планет. Размещение планет на сцене получается идеальным.
3. Также важно отметить, что планеты игрока и вражеского бота должны быть удаленны друг от друга, иначе неинтересно становится играть. Если они будут находиться недалеко друг от друга, то будет легко уничтожить друг друга за короткое время. Благодаря дисциплине теории системы и системного анализа я узнал, что можно решить эту проблему с помощью «Манхэттенского расстояния» или «расстояния городских кварталов». Согласно данной метрике, расстояние между двумя точками равно сумме модулей разностей их координат. Прежде чем выдать планеты игроку и вражескому боту после их случайного размещения, сначала анализируем, какие планеты находятся максимально удаленно слева и справа от центра сцены, запоминаем названия этих планет, затем передаем их игроку и боту по имени.
4. Также непростой задачей было создание и управление кораблями игрока. Для её решения пришлось давать каждому кораблю своё имя и вести подсчёт кораблей игрока на сцене.
5. Большинство обучающих литературы и видео на английском языке, это немного усложнило и замедлило процесс разработки.

##### Заключение

Несмотря на возникшие проблемы в процессе разработки игры, удалось довести игру до конца и реализовать готовый проект. Во время создания игры я ознакомился с игровым движком Unity и набрал навыки разработки двумерной игры. В дальнейшем планируется сделать более корректное перемещение кораблей, а также их ориентацию, отвязку изменений свойств объектов от частоты обновления кадров, разработать другие игры.

## Список используемых источников

1. Документация Unity. https://docs.unity3d.com/Manual/index.html
2. Форум. http://www.cyberforum.ru/unity/
3. Форум. https://habr.com/ru/hub/unity/
4. Обучающие курсы, туториалы и видео. https://learn.unity.com
5. Обучающие видео Unity в YouTube. https://www.youtube.com/results?search\_query=unity
6. Сравнение игровых движков.

https://habr.com/ru/company/piter/blog/269415/