**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СОГЛАСОВАНО  Научный сотрудник МЛ ИССА факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики», канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_ О.В. Максименкова  "\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы "Программная инженерия"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  "\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. | |
| |  |  | | --- | --- | | Подп. и дата |  | | Инв. № дубл. |  | | Взам. Инв. № |  | | Подп. и дата |  | | Инв. № подл. |  | | **МОБИЛЬНАЯ 2D КОСМИЧЕСКАЯ АРКАДА**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 ТЗ 01-1-ЛУ**  Исполнитель  Студент группы БПИ191  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.О. Казанцев  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. | | | | |
|  | |  | | |

УТВЕРЖДЕН

RU.17701729.04.01-01 ТЗ 01-1-ЛУ

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. №подл |  |

**МОБИЛЬНАЯ 2D КОСМИЧЕСКАЯ АРКАДА  
  
Текст программы  
  
RU.17701729.04.01-01 ТЗ 01-1  
  
Листов 60**

СОДЕРЖАНИЕ

[ТЕКСТ ПРОГРАММЫ 3](#_Toc40529631)

[Класс ApplicationSet 3](#_Toc40529632)

[Класс AudioManager 3](#_Toc40529633)

[Класс CameraScript 4](#_Toc40529634)

[Класс Constants 6](#_Toc40529635)

[Класс Events 8](#_Toc40529636)

[Класс GameDeath 9](#_Toc40529637)

[Класс GravitationalBody 9](#_Toc40529638)

[Класс LineRendererFx 36](#_Toc40529639)

[Класс MenuCamera 37](#_Toc40529640)

[Класс MenuManager 38](#_Toc40529641)

[Класс Movement 38](#_Toc40529642)

[Класс OrbitCatcher 40](#_Toc40529643)

[Класс OrbitManager 41](#_Toc40529644)

[Класс PrefabsStorage 43](#_Toc40529645)

[Класс ProgressBar 43](#_Toc40529646)

[Класс ScrollBackground 44](#_Toc40529647)

[Класс Spawner 44](#_Toc40529648)

[Класс TextTyper 56](#_Toc40529649)

[Класс TrashCollector 57](#_Toc40529650)

[Класс UIController 57](#_Toc40529651)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 59](#_Toc40529652)

# ТЕКСТ ПРОГРАММЫ

## [Класс](https://github.com/isp13/Accretion) ApplicationSet

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ApplicationSet : MonoBehaviour

{

/// <summary>

/// 0 - отключение vsync, else - включение

/// </summary>

private int VSyncCount {

set {

QualitySettings.vSyncCount = value;

}

}

/// <summary>

/// Желаемая частота обновления экрана. ios - максимум 60 fps

/// </summary>

private int TargetFrameRate

{

set

{

Application.targetFrameRate = value;

}

}

// Start is called before the first frame update

private void Awake()

{

// Turn off v-sync

VSyncCount = Constants.vSyncCount;

// устанавливаем максимальную частоту обновления экрана

TargetFrameRate = Constants.targetFrameRate;

}

}

## Класс AudioManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine;

public class AudioManager : MonoBehaviour

{

/// <summary>

/// Компонент, позволяющий работать с музыкой в Unity

/// </summary>

private AudioSource \_audioSource;

/// <summary>

/// Звуковая дорожка для проигрывания

/// </summary>

public AudioClip music;

/// <summary>

/// Говорит приложению не прерывать скрипт при открытии новой сцены

/// </summary>

private void Awake()

{

DontDestroyOnLoad(transform.gameObject);

\_audioSource = GetComponent<AudioSource>();

}

/// <summary>

/// Включает музыку

/// </summary>

public void PlayMusic()

{

if (\_audioSource.isPlaying) return;

\_audioSource.clip = music;

\_audioSource.Play(1);

}

public void StopMusic()

{

\_audioSource.Stop();

}

}

## Класс CameraScript

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraScript : MonoBehaviour

{

/// <summary>

/// Скорость слежения камеры за объектом

/// </summary>

public float FollowSpeed = 2f;

/// <summary>

/// Объект слежения

/// </summary>

public Transform Target;

/// <summary>

/// Желаемая координата Z камеры в трехмерном пространстве

/// </summary>

public int targetZPostition = -10;

/// <summary>

/// Начальный зум камеры

/// </summary>

public float Zoom1 = 1f;

/// <summary>

/// Желаемый зум камеры

/// </summary>

public float Zoom2 = 2f;

/// <summary>

/// Интервал плавного зума в секундах

/// </summary>

public float duration = 1.0f;

/// <summary>

/// 3 интервальная переменная для осуществления плавного зума

/// </summary>

///

private float elapsed = 0.0f;

/// <summary>

/// Разрешено ли изменять зум

/// </summary>

private bool transition = true;

/// <summary>

/// Устанавливает стандартную координату камеры при инициализации объекта

/// </summary>

private void Start()

{

this.transform.position = new Vector3(0, 0, 0);

}

/// <summary>

/// Перемещает вслед за объектом слежения.

/// </summary>

private void FixedUpdate()

{

Vector3 newPosition = Target.position;

newPosition.z = targetZPostition;

transform.position = Vector3.Slerp(transform.position, newPosition, FollowSpeed \* Time.deltaTime);

if (transition && this.tag == "MainCamera")

{

elapsed += Time.deltaTime / duration;

this.GetComponent<Camera>().orthographicSize = Mathf.Lerp(Zoom1, Zoom2, elapsed);

//this next line i'm not sure of, I'm not familiar with CameraMovement.ypos

// Camera.main.GetComponent<CameraMovement>().ypos = Mathf.Lerp(ypos1, ypos2, elapsed);

if (elapsed > 1.0f)

{

transition = false;

Zoom1 = Zoom2;

}

}

}

/// <summary>

/// Осуществляет плавный зум

/// </summary>

/// <param name="max">вторая координата для зума</param>

public void SmoothChangeOrthographicSize(float max)

{

Zoom2 = max;

transition = true;

elapsed = 0.0f;

}

}

## Класс Constants

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public static class Constants

{

/// GAME SETTINGS

public static int vSyncCount = 0;

public static int targetFrameRate = 300;

///

public static float GravityPower = 140; // default is 27 // ,было 70 // но тк изменилась гравитация, поднялв два раза

public static Dictionary<string, int> HierarchyDict = new Dictionary<string, int> {

{ "Asteroid" , 0}, { "DwarfPlanet", 1 }, { "Planet", 2 }, { "DwarfStar", 3 },

{ "Star", 4 }, { "GiantStar", 5 }, {"NeutronStar", 6 }, {"BlackHole", 7 } };

public static Dictionary<string, int> GrowCoefDict = new Dictionary<string, int> {

{ "Asteroid" , 1}, { "DwarfPlanet", 2 }, { "Planet", 3 }, { "DwarfStar", 4 },

{ "Star", 4 }, { "GiantStar", 5 }, {"NeutronStar", 15 }, {"BlackHole", 50 } };

public static string PlayersNextObject = "Dwarf Planet";

// Asteroids stats

public static float AsteroidStartingMass = 1f;

public static int AsteroidCriticalMass = 10;

public static int AsteroidScale = 1;

public static int AsteroidMainCameraDistance = 30;

// Dwarf planet stats

public static int DwarfPlanetsStartingMass = 10;

public static int DwarfPlanetCriticalMass = 25;

public static float DwarfPlanetScale = 10f;

public static float ColliderRadius\_DwarfPlanet = 0.54f;

public static int PlanetMainCameraDistance = 70;

// Planet stats

public static int PlanetsStartingMass = 20;

public static int PlanetCriticalMass = 50;

public static float PlanetScale = 13f;

public static float ColliderRadius\_Planet = 0.55f;

public static int DwarfStarStartingMass = 50;

public static int DwarfStarCriticalMass = 100;

public static float DwarfStarScale = 13f;

public static float ColliderRadius\_DwarfStar = 0.55f;

public static int StarStartingMass = 100;

public static int StarCriticalMass = 150;

public static float StarScale = 14f;

public static float ColliderRadius\_Star = 0.6f;

public static int GiantStarStartingMass = 150;

public static int GiantStarCriticalMass = 250;

public static float GiantStarScale = 16f;

public static float ColliderRadius\_GiantStar = 0.65f;

public static int NeutronStarStartingMass = 250;

public static int NeutronStarCriticalMass = 350;

public static float NeutronStarScale = 10f;

public static float ColliderRadius\_NeutronStar = 0.6f;

public static int BlackHoleStartingMass = 350;

public static int BlackHoleCriticalMass = 1000;

public static float BlackHoleScale = 10f;

public static float ColliderRadius\_BlackHole = 0.6f;

public static int MaxGravitationalDistance = 45;

public static int LowerSecondsGenPlanet = 2; // нижняя граница в секундах в промежутке генерирования планет

public static int UpperSecondsGenPlanet = 8; // верхняя граница в секундах в промежутке генерирования планет

public static int DistanceToGenerateObjects = 250;

public static float TrailDisapearTime = 1f;

public static float maxSpeed = 30;

public static float minSpeed = 10;

public static string ColorfulMaterialsFolder = "Assets/Resources/#OnePotatoKingdom\_FULL/Materials/Materials/";

public static string StarsMaterialsFolder = "Assets/Stars/";

public static bool PlayerIsMoving = false;

// не нарушая порядок, словари должны быть внизу

// СЛОВАРИ НЕ ЗАПОЛНЕНЫ ДО КОНЦА

public static Dictionary<string, int> HierarchyMinMass = new Dictionary<string, int> {

{ "Asteroid" , 0}, { "DwarfPlanet", AsteroidCriticalMass }, { "Planet", DwarfPlanetCriticalMass}, { "DwarfStar", PlanetCriticalMass},

{ "Star", DwarfStarCriticalMass }, { "GiantStar", StarCriticalMass }, {"NeutronStar", GiantStarCriticalMass }, {"BlackHole", NeutronStarCriticalMass } };

public static Dictionary<string, int> HierarchyMaxMass = new Dictionary<string, int> {

{ "Asteroid" , AsteroidCriticalMass}, { "DwarfPlanet", DwarfPlanetCriticalMass }, { "Planet", PlanetCriticalMass}, { "DwarfStar", DwarfStarCriticalMass},

{ "Star", StarCriticalMass }, { "GiantStar", GiantStarCriticalMass }, {"NeutronStar", NeutronStarCriticalMass }, {"BlackHole", BlackHoleCriticalMass } };

public static Dictionary<string, string[]> LegalToSpawn = new Dictionary<string, string[]> {

{ "Asteroid" , new string[] {"Asteroid", "DwarfPlanet" } },

{"DwarfPlanet" , new string[] {"Asteroid", "DwarfPlanet", "Planet" } },

{"Planet" , new string[] {"Asteroid", "DwarfPlanet", "Planet", "DwarfStar" } },

{"DwarfStar" , new string[] { "DwarfPlanet", "Planet", "DwarfStar", "Star" } },

{"Star" , new string[] { "DwarfPlanet", "Planet", "DwarfStar", "Star", "GiantStar" } },

{"GiantStar" , new string[] { "DwarfPlanet", "Planet", "DwarfStar", "Star", "GiantStar", "NeutronStar" } },

{"NeutronStar" , new string[] {"Planet", "DwarfStar", "Star", "GiantStar", "NeutronStar", "BlackHole" } },

{"BlackHole" , new string[] { "DwarfPlanet", "Planet", "DwarfStar", "Star", "GiantStar", "NeutronStar" , "BlackHole" } }

};

}

## Класс Events

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Events : MonoBehaviour

{

public delegate void EventHandler();

public static event EventHandler NotifyMovement;

private static System.DateTime lastTimeMovingEventFetched;

private static Random rnd = new Random();

// Start is called before the first frame update

void Start()

{

// adding new methods that can be invoked when player is moving

NotifyMovement += StartSpawning;

lastTimeMovingEventFetched = System.DateTime.Now;

}

public static void InvokeNotifyMovement()

{

int tmp = Random.Range(Constants.LowerSecondsGenPlanet, Constants.UpperSecondsGenPlanet);

Debug.Log(tmp);

NotifyMovement?.Invoke();

lastTimeMovingEventFetched = System.DateTime.Now;

}

public static void StartSpawning() {

Debug.Log("started spawning");

GameObject.Find("RandomObjectsGenerator").GetComponent<Spawner>().Spawn();

}

}

## Класс GameDeath

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class GameDeath : MonoBehaviour

{

public GameObject panel;

public void Show\_Panel ()

{

panel.SetActive(true);

}

public void Game\_start()

{

SceneManager.LoadScene("Assets/Scenes/SampleScene.unity");

}

public void MainMenu()

{

SceneManager.LoadScene("Assets/Scenes/Main Menu.unity");

}

}

## Класс GravitationalBody

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using UnityEditor;

using cakeslice;

[RequireComponent(typeof(Rigidbody2D))]

public class GravitationalBody : MonoBehaviour

{

private PrefabsStorage prefabs;

public string name = "";

public float maxDistance;

private float startingMass;

public float StartingMass // for transformations

{

get { return startingMass; }

set { startingMass = value; } // here need to check if its alot mass so object can transform to the next stage

}

private float imaginaryMass;

public float ImaginaryMass // for transformations

{

get { return imaginaryMass; }

set { imaginaryMass = value; } // here need to check if its alot mass so object can transform to the next stage

}

public Vector2 initialVelocity;

private Rigidbody2D rb;

//I use a static list of bodies so that we don't need to Find them every frame

static List<Rigidbody2D> attractableBodies = new List<Rigidbody2D>();

public List<GameObject> OrbitBodies = new List<GameObject>();

System.Random rnd = new System.Random();

public bool onOrbit = false;

public Transform target;

private System.DateTime lastTimeDownGrade;

private bool FirstAsteroidToBeHit = false;

private bool FirstTransformationIntoDwarfPlanet = false;

void Start()

{

lastTimeDownGrade = System.DateTime.Now;

//this.name = "Planet";

prefabs = GameObject.Find("PrefabStorage").GetComponent<PrefabsStorage>();

if (this.name == "Asteroid" || this.tag == "Asteroid")

{

this.StartingMass = Constants.AsteroidStartingMass;

this.GetComponent<Rigidbody2D>().AddForce(Constants.GravityPower \* new Vector3(rnd.Next(), rnd.Next(), rnd.Next()).normalized);

}

// подсветка включается только у планет и звезд. поэтому отключаем сейчас

if (this.tag == "Player" && this.name == "Asteroid")

this.GetComponent<Outline>().enabled = false;

this.maxDistance = Constants.MaxGravitationalDistance;

SetupRigidbody2D();

SetupColliders();

SetupMeshRenderer();

SetupTrailRenderer();

SetupMapPoint();

//Add this gravitational body to the list, so that all other gravitational bodies can be effected by it

attractableBodies.Add(rb);

}

void SetupMapPoint()

{

if (this.tag != "Player")

{

var point = GameObject.CreatePrimitive(PrimitiveType.Sphere);

Destroy(point.GetComponent<SphereCollider>());

//point.GetComponent<Renderer>().material.color = new Color(1, 1, 1, 1);

point.layer = LayerMask.NameToLayer("POINT");

var pointgm = Instantiate(point);

pointgm.transform.position = (this.transform.position);

pointgm.transform.localScale \*= 5;

//Sets "ChoiceButtonHolder" as the new parent of the s1Button.

pointgm.transform.SetParent(this.transform);

}

}

void SetupRigidbody2D()

{

if (this.tag != "Player" )

{

this.gameObject.AddComponent<Rigidbody2D>();

}

rb = this.gameObject.GetComponent<Rigidbody2D>();

rb.gravityScale = 0f;

rb.drag = 0f;

rb.angularDrag = 2f; // останавливает со временем вращение объекта

rb.mass = StartingMass;

rb.velocity = initialVelocity;

}

void SetupColliders()

{

this.gameObject.AddComponent<CircleCollider2D>();

var rb1 = this.gameObject.GetComponent<CircleCollider2D>();

this.gameObject.AddComponent<CircleCollider2D>();

var rb2 = this.gameObject.GetComponent<CircleCollider2D>();

rb2.isTrigger = true;

}

void SetupTrailRenderer() {

this.gameObject.AddComponent<TrailRenderer>();

this.GetComponent<TrailRenderer>().time = Constants.TrailDisapearTime;

this.GetComponent<TrailRenderer>().material.color = Random.ColorHSV(0f,1f,1f,1f,1f, 1f);

this.GetComponent<TrailRenderer>().enabled = false;

}

void SetupMeshRenderer() {

this.gameObject.AddComponent<MeshRenderer>();

}

void FixedUpdate()

{

if (!onOrbit) //если не на орбите - физика работает в нормальном режиме

{

foreach (Rigidbody2D otherBody in attractableBodies)

{

if (otherBody == null)

continue;

//We arn't going to add a gravitational pull to our own body

else if (otherBody == rb)

continue;

else

{

}

// first, it was without any gravity power. But movements were really slow

// добавил второе условие, чтобы притягивал только больший по рангу, не было взаимного притяжения

if (this.name == "Asteroid" && otherBody.gameObject.GetComponent<GravitationalBody>().name == "Asteroid")

{

}

else if ((this.name != "Asteroid") && Constants.HierarchyDict[this.name] >= Constants.HierarchyDict[otherBody.GetComponent<GravitationalBody>().name])

otherBody.AddForce(Constants.GravityPower \* DetermineGravitationalForce(otherBody));

}

}

else

{

Vector3 relativePos = (target.position + new Vector3(0, 1.5f, 0)) - transform.position;

Quaternion rotation = Quaternion.LookRotation(relativePos);

Quaternion current = transform.localRotation;

transform.localRotation = Quaternion.Slerp(current, rotation, Time.deltaTime);

transform.Translate(0, 0, 3 \* Time.deltaTime);

//rb.mass = 0; // обнуляем массу чтобы наш объект не тянуло в сторону орбитального

//attractableBodies.Remove(this.rb);

}

}

Vector2 DetermineGravitationalForce(Rigidbody2D otherBody)

{

Vector2 relativePosition = rb.position - otherBody.position;

float distance = Mathf.Clamp(relativePosition.magnitude, 0, maxDistance);

//the force of gravity will reduce by the distance squared

float gravityFactor = 1f - (Mathf.Sqrt(distance) / Mathf.Sqrt(maxDistance));

//creates a vector that will force the otherbody toward this body, using the gravity factor times the mass of this body as the magnitude

Vector2 gravitationalForce = relativePosition.normalized \* (gravityFactor \* rb.mass);

return gravitationalForce;

}

// если объект захвачен на орбиту, предоставить возможность его "съесть по нажатии на него"

void OnMouseDown()

{

Debug.Log("CLICK");

// this object was clicked - do something

if (onOrbit)

{

Debug.Log("DESROYING");

// увеличиваем массу центра системы из космических объектов кода съели

GameObject.Find("Player").GetComponent<GravitationalBody>().StartingMass += CalculateMassGrow(this.name);

Debug.Log("Destroying by click");

Destroy(this.gameObject);

}

else

{

this.StartingMass += OrbitBodies.Count;

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

{

Destroy(OrbitBodies[i].gameObject);

}

}

// если наш объект больше другого на две позиции - увеличение массы возможно даже врезавшись.

if (StartingMass / Constants.HierarchyMaxMass[name] >= 1f)

{

// transform to new object

Debug.Log("NEED TO TRANSFORM");

Upgrade\_Retransform\_Object();

if (this.tag == "Player" && (this.name == "Planet" || this.name.Contains("Star")))

this.GetComponent<Outline>().enabled = true;

else

{

this.GetComponent<Outline>().enabled = false;

}

}

}

void OnDestroy()

{

if (this.tag == "Player")

GameObject.Find("GameEnding").GetComponent<GameDeath>().Show\_Panel();

}

void OnCollisionEnter2D(Collision2D coll)

{

if (coll.gameObject.tag == "destroyer")

return;

if (this.name == "BlackHole")

{

Destroy(coll.gameObject);

this.StartingMass += CalculateMassGrow();

if (StartingMass / Constants.HierarchyMaxMass[name] >= 1f)

{

if (this.tag == "Player")

GameObject.Find("GameEnding").GetComponent<GameDeath>().Show\_Panel();

}

return;

}

// если наш объект больше другого на две позиции - увеличение массы возможно даже врезавшись.

if (Constants.HierarchyDict[name] > Constants.HierarchyDict[coll.gameObject.GetComponent<GravitationalBody>().name] + 1)

{

Debug.Log("Destroying object because a bigger one was near");

Destroy(coll.gameObject);

this.StartingMass += CalculateMassGrow();

if (StartingMass / Constants.HierarchyMaxMass[name] >= 1f)

{

// transform to new object

Debug.Log("NEED TO TRANSFORM");

Upgrade\_Retransform\_Object();

if (this.tag == "Player" && (this.name == "Planet" || this.name.Contains("Star")))

this.GetComponent<Outline>().enabled = true;

else

{

this.GetComponent<Outline>().enabled = false;

}

}

}

else if (Constants.HierarchyDict[name] > Constants.HierarchyDict[coll.gameObject.GetComponent<GravitationalBody>().name]) // если больше на одну позицию

{

Debug.Log("Destroying object because a bigger one was near");

//GameObject.Find("OnlyAsteroidsGenerator").GetComponent<Spawner>().SpawnAsteroidConstantPosition(this.transform.position + new Vector3((float)rnd.NextDouble(), (float)rnd.NextDouble(), (float)rnd.NextDouble()));

Destroy(coll.gameObject);

this.StartingMass -= CalculateMassGrow();

if (StartingMass < Constants.HierarchyMinMass[name]) // если объект потерял массу и стал весить меньше обычного

{

// transform to new object

Debug.Log("NEED TO TRANSFORM");

DownGrade\_Retransform\_Object();

if (this.tag == "Player" && (this.name == "Planet" || this.name.Contains("Star")))

this.GetComponent<Outline>().enabled = true;

else

{

this.GetComponent<Outline>().enabled = false;

}

// изменение мешей, массы и тп при трансформации

}

}

else if (this.name == "Asteroid" && Constants.HierarchyDict[name] == Constants.HierarchyDict[coll.gameObject.GetComponent<GravitationalBody>().name])

{

if (this.tag == "Player")

{

if (FirstAsteroidToBeHit == false)

{

FirstAsteroidToBeHit = true;

StartCoroutine(GameObject.Find("TextTyper").GetComponent<TextTyper>().TypeText("Congrats. Now gain 10 mass to transform into dwarf planet."));

}

this.startingMass += CalculateMassGrow();

Destroy(coll.gameObject);

if (StartingMass / Constants.HierarchyMaxMass[name] >= 1f)

{

// transform to new object

Debug.Log("NEED TO TRANSFORM");

Upgrade\_Retransform\_Object();

if (this.tag == "Player" && (this.name == "Planet" || this.name.Contains("Star")))

this.GetComponent<Outline>().enabled = true;

else

{

this.GetComponent<Outline>().enabled = false;

}

// изменение мешей, массы и тп при трансформации

}

else {

if (this.name == "Asteroid") // небольшое увеличение астероида после удара в другой астероид

this.transform.localScale \*= 1.1f;

}

}

}

else if (Constants.HierarchyDict[name] == Constants.HierarchyDict[coll.gameObject.GetComponent<GravitationalBody>().name])

{

//нужно переродить объекты в меньшие и заспавнить космический мусор

if (lastTimeDownGrade.AddSeconds(1) <= System.DateTime.Now && this.GetHashCode() > coll.gameObject.GetHashCode())

{

lastTimeDownGrade = System.DateTime.Now;

DownGrade\_Retransform\_Object();

coll.gameObject.GetComponent<GravitationalBody>().DownGrade\_Retransform\_Object();

this.GetComponent<Rigidbody2D>().AddForce((StartingMass \* 0.5f) \* Constants.GravityPower \* DetermineGravitationalForce(coll.gameObject.GetComponent<Rigidbody2D>()));

coll.gameObject.GetComponent<Rigidbody2D>().AddForce(-0.5f \* (StartingMass \* 0.4f) \* Constants.GravityPower \* DetermineGravitationalForce(coll.gameObject.GetComponent<Rigidbody2D>()));

}

}

}

public int CalculateMassGrow()

{

return Constants.GrowCoefDict[this.name];

}

public int CalculateMassGrow(string nm)

{

return Constants.GrowCoefDict[nm];

}

public string ShareObjectData(string option)

{

switch (option)

{

case "name":

return name.ToString();

case "mass":

return StartingMass.ToString();

case "coords":

return rb.position.x.ToString() + "," + rb.position.y.ToString();

case "vector":

return rb.velocity.x.ToString() + "," + rb.velocity.y.ToString();

}

return "error";

}

/// <summary>

/// get object movement vector

/// </summary>

/// <returns>Vector3 movement vector</returns>

public Vector3 GetMovementVector()

{

return rb.velocity;

}

public void Upgrade\_Retransform\_Object()

{

// transform to new object

Debug.Log("NEED TO TRANSFORM");

float saveMass = this.StartingMass;

// номер следующего объекта из словаря

int nextRank = Constants.HierarchyDict[name] + 1;

this.name = Constants.HierarchyDict.FirstOrDefault(x => x.Value == nextRank).Key;

if (this.tag != "Player")

this.tag = Constants.HierarchyDict.FirstOrDefault(x => x.Value == nextRank).Key;

// изменение мешей, массы и тп при трансформации

if (name == "Asteroid")

Retransform\_Asteroid();

else if (name == "DwarfPlanet")

Retransform\_DwarfPlanet();

else if (name == "Planet")

Retransform\_Planet();

else if (name == "DwarfStar")

Retransform\_DwarfStar();

else if (name == "Star")

Retransform\_Star();

else if (name == "GiantStar")

Retransform\_GiantStar();

else if (name == "NeutronStar")

Retransform\_NeutronStar();

else if (name == "BlackHole")

Retransform\_BlackHole();

else

Debug.Log("tag error");

this.StartingMass = (int)(saveMass \* 1.1f) + 1;

}

public void DownGrade\_Retransform\_Object()

{

float saveMass = Constants.HierarchyMinMass[this.name];

int nextRank = Constants.HierarchyDict[name] - 1;

this.name = Constants.HierarchyDict.FirstOrDefault(x => x.Value == nextRank).Key;

if (this.tag != "Player")

this.tag = Constants.HierarchyDict.FirstOrDefault(x => x.Value == nextRank).Key;

if (name == "Asteroid")

Retransform\_Asteroid();

else if (name == "DwarfPlanet")

Retransform\_DwarfPlanet();

else if (name == "Planet")

Retransform\_Planet();

else if (name == "DwarfStar")

Retransform\_DwarfStar();

else if (name == "Star")

Retransform\_Star();

else if (name == "GiantStar")

Retransform\_GiantStar();

else if (name == "NeutronStar")

Retransform\_NeutronStar();

else if (name == "BlackHole")

Retransform\_BlackHole();

else

Debug.Log("tag error");

this.StartingMass = (int)(saveMass \* 0.8f);

}

public void Retransform\_Asteroid()

{

Debug.Log("Retransform\_Asteroid");

// получаем рандомную планету из префабов

Object planetPrefab = prefabs.asteroids[rnd.Next(0, prefabs.asteroids.Length)] as GameObject;

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.AsteroidScale, Constants.AsteroidScale, Constants.AsteroidScale);

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

Destroy(randomOne);

if (this.tag == "Player")

{

Constants.PlayersNextObject = "Dwarf Planet";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.AsteroidMainCameraDistance);

}

}

/// <summary>

/// Функция для превращения модели объекта в модель карликовой планеты

/// </summary>

public void Retransform\_DwarfPlanet()

{

Debug.Log("Retransform\_DwarfPlanet");

// получаем рандомную планету из префабов

Object planetPrefab = prefabs.planets[rnd.Next(0, prefabs.planets.Length)] as GameObject;

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1,1,1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.DwarfPlanetScale, Constants.DwarfPlanetScale, Constants.DwarfPlanetScale);

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_DwarfPlanet;

}

}

Destroy(randomOne);

if (this.tag == "Player")

{

if (FirstTransformationIntoDwarfPlanet == false)

{

FirstTransformationIntoDwarfPlanet = true;

StartCoroutine(GameObject.Find("TextTyper").GetComponent<TextTyper>().TypeText("GJ. You are big, but not big enough to continue smashing everything ", "Now try to catch asteroid on orbit."));

}

Constants.PlayersNextObject = "Planet";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.PlanetMainCameraDistance);

}

}

public void Retransform\_Planet()

{

Debug.Log("Retransform\_Planet");

// получаем рандомную планету из префабов

Object planetPrefab = prefabs.planets[rnd.Next(0, prefabs.planets.Length)] as GameObject;

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.PlanetScale, Constants.PlanetScale, Constants.PlanetScale);

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_Planet;

}

}

Destroy(randomOne);

if (this.tag == "Player")

{

Constants.PlayersNextObject = "Dwarf Star";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.PlanetMainCameraDistance);

}

}

public void Retransform\_DwarfStar()

{

Debug.Log("Retransform\_DwarfStar");

// тк звезда, то увеличиваем подсвевиваемость префаба

var cam = GameObject.Find("Main Camera").GetComponent<cakeslice.OutlineEffect>().lineIntensity = 1.3f;

// получаем рандомную карликовую звезду из префабов

Object planetPrefab = prefabs.stars[1] as GameObject; // dark white

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.DwarfStarScale, Constants.DwarfStarScale, Constants.DwarfStarScale);

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_DwarfStar;

}

}

Destroy(randomOne);

if (this.tag == "Player")

{

Constants.PlayersNextObject = "Star";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.PlanetMainCameraDistance);

}

}

public void Retransform\_Star()

{

Debug.Log("Retransform\_Star");

// получаем рандомную звезду из префабов

Object planetPrefab = prefabs.stars[2] as GameObject; // yellow

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.StarScale, Constants.StarScale, Constants.StarScale);

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_Star;

}

}

Destroy(randomOne);

if (this.tag == "Player")

{

Constants.PlayersNextObject = "Giant Star";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.PlanetMainCameraDistance);

}

}

public void Retransform\_GiantStar()

{

Debug.Log("Retransform\_GiantStar");

// получаем рандомную звезду из префабов

Object planetPrefab = prefabs.stars[0] as GameObject; //red

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.GiantStarScale, Constants.GiantStarScale, Constants.GiantStarScale);

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_GiantStar;

}

}

Destroy(randomOne);

if (this.tag == "Player")

{

Constants.PlayersNextObject = "Neutron Star";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.PlanetMainCameraDistance);

}

}

public void Retransform\_NeutronStar()

{

Debug.Log("Retransform\_NeutronStar");

// получаем рандомную звезду из префабов

Object planetPrefab = prefabs.stars[4] as GameObject; // blue star

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

this.transform.localScale = new Vector3(Constants.NeutronStarScale, Constants.NeutronStarScale, Constants.NeutronStarScale);

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_NeutronStar;

}

}

Destroy(randomOne);

if (this.tag == "Player")

{

Constants.PlayersNextObject = "Black Hole";

GameObject.Find("Main Camera").GetComponent<CameraScript>().SmoothChangeOrthographicSize(Constants.PlanetMainCameraDistance);

}

}

public void Retransform\_BlackHole()

{

// получаем рандомную звезду из префабов

Object planetPrefab = prefabs.BlackHole[0] as GameObject;

GameObject randomOne = Instantiate(planetPrefab, new Vector3(1, 1, 1) \* 10000, this.transform.rotation) as GameObject;

this.gameObject.GetComponent<MeshFilter>().mesh = randomOne.GetComponent<MeshFilter>().mesh;

this.gameObject.GetComponent<MeshRenderer>().materials = randomOne.GetComponent<MeshRenderer>().materials;

this.StartingMass = Constants.HierarchyMinMass[name];

this.ImaginaryMass = Constants.HierarchyMaxMass[name];

this.transform.localScale = new Vector3(Constants.BlackHoleScale, Constants.BlackHoleScale, Constants.BlackHoleScale);

float scaledRadius = Mathf.Max(transform.localScale.x, transform.localScale.y);

this.GetComponent<CircleCollider2D>().radius = scaledRadius;

CircleCollider2D[] colliders = this.transform.GetComponents<CircleCollider2D>();

foreach (CircleCollider2D collider in colliders)

{

if (!collider.isTrigger)

{

// This is collider is not a trigger

collider.radius = Constants.ColliderRadius\_BlackHole;

}

}

Destroy(randomOne);

Constants.PlayersNextObject = "???";

}

}

## Класс LineRendererFx

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[RequireComponent(typeof(LineRenderer))]

public class LineRendererEx : MonoBehaviour

{

public int vertexCount = 40; // 4 vertices == square

public float lineWidth = 0.2f;

public float radius;

private LineRenderer lineRenderer;

private void Awake()

{

lineRenderer = GetComponent<LineRenderer>();

SetupCircle();

}

private void SetupCircle()

{

lineRenderer.widthMultiplier = lineWidth;

float deltaTheta = (2f \* Mathf.PI) / vertexCount;

float theta = 0f;

lineRenderer.positionCount = vertexCount;

for (int i = 0; i < lineRenderer.positionCount; i++)

{

Vector3 pos = new Vector3(radius \* Mathf.Cos(theta), radius \* Mathf.Sin(theta), 0f);

lineRenderer.SetPosition(i, pos);

theta += deltaTheta;

}

}

#if UNITY\_EDITOR

private void OnDrawGizmos()

{

float deltaTheta = (2f \* Mathf.PI) / vertexCount;

float theta = 0f;

Vector3 oldPos = Vector3.zero;

for (int i = 0; i < vertexCount + 1; i++)

{

Vector3 pos = new Vector3(radius \* Mathf.Cos(theta), radius \* Mathf.Sin(theta), 0f);

Gizmos.DrawLine(oldPos, transform.position + pos);

oldPos = transform.position + pos;

theta += deltaTheta;

}

}

#endif

}

## Класс MenuCamera

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MenuCamera : MonoBehaviour

{

public Vector3 Targetposition;

public bool camera\_move\_enabled = true;

void Update()

{

if (camera\_move\_enabled)

{

this.transform.position = Vector3.Lerp(transform.position, Targetposition, 0.1f \* Time.deltaTime);

}

}

}

## Класс MenuManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

public class MenuManager : MonoBehaviour

{

public Button startGameButton;

// Start is called before the first frame update

void Start()

{

startGameButton.onClick.AddListener(StartGame);

GameObject.Find("Music").GetComponent<AudioManager>().PlayMusic();

}

// Update is called once per frame

void Update()

{

}

void StartGame()

{

Debug.Log("CLICK");

SceneManager.LoadScene("Assets/Scenes/SampleScene.unity");

}

}

## Класс Movement

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Movement : MonoBehaviour

{

int speed = 10;

int speedLimit = 20;

public static bool isMoving = false;

private Rigidbody2D rb;

public FloatingJoystick variableJoystick;

// Start is called before the first frame update

void Start()

{

rb = this.gameObject.GetComponent<Rigidbody2D>();

}

// Next update in second

private int nextUpdate = 2;

// Update is called once per frame

void Update()

{

bool somethingWasPressed = false; // checking if movement was made this frame

if (Input.GetKey(KeyCode.LeftArrow))

{

rb.AddForce(Vector2.left \* speed);

//rb.AddForce(Vector3.up \* speed);

rb.velocity = Vector2.ClampMagnitude(rb.velocity, speedLimit);

somethingWasPressed = true;

}

if (Input.GetKey(KeyCode.UpArrow))

{

rb.AddForce(Vector2.up \* speed);

//rb.AddForce(Vector3.up \* speed);

rb.velocity = Vector2.ClampMagnitude(rb.velocity, speedLimit);

somethingWasPressed = true;

}

if (Input.GetKey(KeyCode.RightArrow))

{

rb.AddForce(Vector2.right \* speed);

//rb.AddForce(Vector3.up \* speed);

rb.velocity = Vector2.ClampMagnitude(rb.velocity, speedLimit);

somethingWasPressed = true;

}

if (Input.GetKey(KeyCode.DownArrow))

{

rb.AddForce(Vector2.down \* speed);

//rb.AddForce(Vector3.up \* speed);

rb.velocity = Vector2.ClampMagnitude(rb.velocity, speedLimit);

somethingWasPressed = true;

}

if (variableJoystick.Vertical != 0 || variableJoystick.Horizontal != 0)

{

Vector3 direction = Vector3.up \* variableJoystick.Vertical + Vector3.right \* variableJoystick.Horizontal;

rb.AddForce(direction \* speed);

rb.velocity = Vector2.ClampMagnitude(rb.velocity, speedLimit);

somethingWasPressed = true;

}

if (somethingWasPressed)

{

// If the next update is reached

if (Time.time >= nextUpdate)

{

int tmp = Random.Range(Constants.LowerSecondsGenPlanet, Constants.UpperSecondsGenPlanet);

// Change the next update (current second+1)

nextUpdate = Mathf.FloorToInt(Time.time) + tmp;

// Call your fonction

Events.InvokeNotifyMovement();

}

}

}

}

## Класс OrbitCatcher

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class OrbitCatcher : MonoBehaviour

{

bool isPlayerColliding = false;

// How long the player needs to stay at location

public float timerCountDown = 5.0f;

GravitationalBody player;

void FixedUpdate()

{

// Collision timer

if (isPlayerColliding == true)

{

timerCountDown -= Time.deltaTime;

if (timerCountDown < 0)

{

timerCountDown = 0;

}

}

}

private bool FirstOrbitObject = false;

private void Start()

{

player = GameObject.Find("Player").GetComponent<GravitationalBody>();

}

// Start the collision timer when player enters

void OnTriggerEnter2D(Collider2D other)

{

if (other.gameObject.tag != "Player")

{

Debug.Log("object Entered");

isPlayerColliding = true;

}

}

// Check if the player is still at location, if they are spawn our secret item

void OnTriggerStay2D(Collider2D other)

{

if (other.gameObject.tag != "Player" && isPlayerColliding == true)

{

if (timerCountDown <= 0)

{

other.GetComponent<GravitationalBody>().target = this.transform;

other.GetComponent<GravitationalBody>().onOrbit = true;

player.OrbitBodies.Add(other.gameObject);

other.GetComponent<TrailRenderer>().enabled = false;

timerCountDown = 5;

if (FirstOrbitObject == false && player.GetComponent<GravitationalBody>().name != "Asteroid")

{

FirstOrbitObject = true;

StartCoroutine(GameObject.Find("TextTyper").GetComponent<TextTyper>().TypeText("Now tap on yourself. You will 'eat' it & gain mass", "if you are big enough-smash into objects, otherwise catch them on orbit"));

}

}

}

}

// If the player is not colliding reset our timer

void OnTriggerExit2D(Collider2D other)

{

if (other.gameObject.tag != "Player")

{

timerCountDown = 5;

Debug.Log("object Exited");

isPlayerColliding = false;

player.OrbitBodies.Remove(other.gameObject);

other.GetComponent<GravitationalBody>().onOrbit = false;

}

}

}

## Класс OrbitManager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class OrbitManager : MonoBehaviour

{

//private int FalseEntries = 0;

// Start is called before the first frame update

LineRenderer OrbitRenderer;

GravitationalBody player;

void Start()

{

OrbitRenderer = GameObject.Find("OrbitRenderer").GetComponent<LineRenderer>();

player = GameObject.Find("Player").GetComponent<GravitationalBody>();

OrbitRenderer.enabled = false;

}

// Update is called once per frame

void Update()

{

}

//When the Primitive collides with the walls, it will reverse direction

private void OnTriggerEnter2D(Collider2D other)

{

if (player.name != "Asteroid")

{

//Debug.Log("ONTRIGGER ENTER");

StartCoroutine("FlashesBeforeSomething");

other.gameObject.GetComponent<TrailRenderer>().enabled = true;

OrbitRenderer.enabled = true;

}

}

//When the Primitive exits the collision, it will change Color

private void OnTriggerExit2D(Collider2D other) {

Debug.Log("ONTRIGGER EXIT");

//StartCoroutine("FlashesBeforeSomething");

other.gameObject.GetComponent<TrailRenderer>().enabled = false;

OrbitRenderer.enabled = false;

}

IEnumerator FlashesBeforeSomething()

{

int counter = 0;

while (counter !=8) // this just equates to "repeat forever"

{

counter += 1;

yield return new WaitForSeconds(0.3f); // "pauses" for 2 seconds.. note, the actual game doesn't pause..

//Debug.Log("IEnumerator");

OrbitRenderer.enabled = counter % 2 != 0;

}

}

}

## Класс PrefabsStorage

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PrefabsStorage : MonoBehaviour

{

[SerializeField] public GameObject[] asteroids;

[SerializeField] public Material[] asteroidsMaterials;

[SerializeField] public GameObject[] planets;

[SerializeField] public GameObject[] stars;

[SerializeField] public GameObject[] BlackHole;

}

## Класс ProgressBar

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ProgressBar : MonoBehaviour

{

// Start is called before the first frame update

public float barDisplay; //current progress

public Vector2 pos = new Vector2(20, 40);

public Vector2 size = new Vector2(60, 20);

public Texture2D emptyTex;

public Texture2D fullTex;

private GameObject player;

private GravitationalBody playersgravity;

void OnGUI()

{

//draw the background:

GUI.BeginGroup(new Rect(pos.x, pos.y, size.x, size.y));

GUI.Box(new Rect(0, 0, size.x, size.y), emptyTex);

//draw the filled-in part:

GUI.BeginGroup(new Rect(0, 0, size.x \* barDisplay, size.y));

GUI.Box(new Rect(0, 0, size.x, size.y), fullTex);

GUI.EndGroup();

GUI.EndGroup();

}

void Start()

{

player = GameObject.Find("Player");

playersgravity = player.GetComponent<GravitationalBody>();

}

void FixedUpdate()

{

//for this example, the bar display is linked to the current time,

//however you would set this value based on your desired display

//eg, the loading progress, the player's health, or whatever.

barDisplay = playersgravity.StartingMass / Constants.HierarchyMaxMass[playersgravity.name];

// barDisplay = MyControlScript.staticHealth;

}

}

## Класс ScrollBackground

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ScollBackground : MonoBehaviour

{

public float speed = 0.5f;

Material m\_Material;

GameObject pl;

// Start is called before the first frame update

void Start()

{

m\_Material = GetComponent<Renderer>().material;

pl = GameObject.Find("Player");

}

// Update is called once per frame

void FixedUpdate()

{

var vector\_velocity = pl.GetComponent<Rigidbody2D>().velocity;

Vector2 offset = new Vector2(vector\_velocity.x, vector\_velocity.y) \* speed \* -1;

m\_Material.mainTextureOffset += offset \* Time.deltaTime;

}

}

## Класс Spawner

using cakeslice;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using UnityEditor;

using UnityEngine;

public class Spawner : MonoBehaviour

{

public string genType = "Asteroid";

private int randomHash;

private System.Random rnd;

private System.DateTime lastTimeGenerated;

private PrefabsStorage prefabs;

private GameObject player;

private int asteroidsCount = 0;

// Start is called before the first frame update

void Start()

{

randomHash = this.GetInstanceID();

rnd = new System.Random(randomHash);

lastTimeGenerated = System.DateTime.Now;

player = GameObject.Find("Player");

prefabs = GameObject.Find("PrefabStorage").GetComponent<PrefabsStorage>();

}

// Update is called once per frame

void FixedUpdate()

{

if (genType == "AsteroidONLY" && asteroidsCount <= 12 )

{

asteroidsCount += 1;

Vector3 vectorToSpawn = player.GetComponent<GravitationalBody>().GetMovementVector().normalized + (rnd.Next(0, 2) == 1 ? -1 : 1) \* new Vector3(rnd.Next(), rnd.Next(), 0).normalized / 2;

SpawnAsteroid(vectorToSpawn);

//SpawnDwarfStar(vectorToSpawn);

//SpawnGiantStar(vectorToSpawn);

//SpawnNeutronStar(vectorToSpawn);

//SpawnBlackHole(vectorToSpawn);

//SpawnStar(vectorToSpawn);

}

}

/\*

\* If probabilityPower is above 1, lower values will be more common than higher values. If it's between 0 to 1, higher values will be more common than lower values. If it's 1, the results will be in a general randomness.

\*/

private int GetRandomNumber(int max, int min, double probabilityPower = 2)

{

var randomizer = new System.Random();

var randomDouble = randomizer.NextDouble();

var result = System.Math.Floor(min + (max + 1 - min) \* (System.Math.Pow(randomDouble, probabilityPower)));

return (int)result;

}

/// <summary>

/// берем вектор-направление игрока и спавним в том направлении

/// </summary>

public void Spawn()

{

Vector3 vectorToSpawn = player.GetComponent<GravitationalBody>().GetMovementVector().normalized + (rnd.Next(0,2) == 1 ? -1: 1) \* new Vector3(rnd.Next(), rnd.Next(), 0).normalized / 2;

string[] objectsToSpawn = (Constants.LegalToSpawn[player.GetComponent<GravitationalBody>().name]);

int index = GetRandomNumber(objectsToSpawn.Length - 1, 0, 2);

//string objectToSpawn = objectsToSpawn[rnd.Next(0, objectsToSpawn.Length)];

string objectToSpawn = objectsToSpawn[index];

if (System.Math.Pow(vectorToSpawn.x - player.transform.position.x, 2) + System.Math.Pow(vectorToSpawn.y - player.transform.position.y, 2) >= System.Math.Pow(Constants.DistanceToGenerateObjects, 2))

{

switch (objectToSpawn)

{

case "Asteroid":

Debug.Log("Spawning Asteroid");

SpawnAsteroid(vectorToSpawn);

break;

case "DwarfPlanet":

Debug.Log("Spawning DwarfPlanet");

SpawnDwarfPlanet(vectorToSpawn);

break;

case "Planet":

Debug.Log("Spawning Planet");

SpawnPlanet(vectorToSpawn);

break;

case "DwarfStar":

Debug.Log("Spawning DwarfStar");

SpawnDwarfStar(vectorToSpawn);

break;

case "Star":

Debug.Log("Spawning Star");

SpawnStar(vectorToSpawn);

break;

case "GiantStar":

Debug.Log("Spawning GiantStar");

SpawnGiantStar(vectorToSpawn);

break;

case "NeutronStar":

Debug.Log("Spawning NeutronStar");

SpawnNeutronStar(vectorToSpawn);

break;

case "BlackHole":

Debug.Log("Spawning BlackHole");

SpawnBlackHole(vectorToSpawn);

break;

default:

Debug.Log("WRONG CASE SWITCH");

break;

}

}

}

void SpawnPlanets(Vector3 vc)

{

// получаем рандомную планету из префабов

Object planetPrefab = prefabs.planets[rnd.Next(0, prefabs.planets.Length)] as GameObject;

// позиция игрока

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

// инициализируем планету в игре

var newplanet = Instantiate(planetPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "Planet";

// добавляем гравитацию ей

var gvBody = newplanet.AddComponent<GravitationalBody>();

gvBody.name = "Planet";

// меняем ее размер, чтобы он соответствовал ее типу

newplanet.transform.localScale \*= Constants.PlanetScale;

var rotator = newplanet.AddComponent<Rotatator>();

rotator.Randomize\_rotation();

}

void SpawnAsteroid(Vector3 vc)

{

Object asteroidPrefab = prefabs.asteroids[rnd.Next(0, prefabs.asteroids.Length)] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(asteroidPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "Asteroid";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "Asteroid";

newplanet.transform.localScale = new Vector3(Constants.AsteroidScale, Constants.AsteroidScale, Constants.AsteroidScale); // УБРАЛ УМНОЖИТЬ РАВНО

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.AsteroidStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.AsteroidCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

Material[] materials = new Material[] { prefabs.asteroidsMaterials[rnd.Next(0, prefabs.asteroidsMaterials.Length)] };

newplanet.GetComponent<MeshRenderer>().materials = materials;

}

void SpawnDwarfPlanet(Vector3 vc)

{

Object planetPrefab = prefabs.planets[rnd.Next(0, prefabs.planets.Length)] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(planetPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "DwarfPlanet";

newplanet.AddComponent<GravitationalBody>();

newplanet.AddComponent<Rotatator>();

newplanet.GetComponent<GravitationalBody>().name = "DwarfPlanet";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.DwarfPlanetsStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.DwarfPlanetCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.PlanetScale, Constants.PlanetScale, Constants.PlanetScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

void SpawnPlanet(Vector3 vc)

{

Object planetPrefab = prefabs.planets[rnd.Next(0, prefabs.planets.Length)] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(planetPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "Planet";

newplanet.AddComponent<GravitationalBody>();

var g = newplanet.AddComponent<Outline>();

Debug.Log(g);

newplanet.GetComponent<GravitationalBody>().name = "Planet";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.PlanetsStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.PlanetCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.PlanetScale, Constants.PlanetScale, Constants.PlanetScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

void SpawnDwarfStar(Vector3 vc)

{

Object dwarfStarsPrefab = prefabs.stars[0] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(dwarfStarsPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "DwarfStar";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "DwarfStar";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.DwarfStarStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.DwarfStarCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.DwarfStarScale, Constants.DwarfStarScale, Constants.DwarfStarScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

void SpawnStar(Vector3 vc)

{

Object StarsPrefab = prefabs.stars[rnd.Next(1, 3)] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(StarsPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "Star";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "Star";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.StarStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.StarCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.StarScale, Constants.StarScale, Constants.StarScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

void SpawnGiantStar(Vector3 vc)

{

Object giantStarsPrefab = prefabs.stars[3] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(giantStarsPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "GiantStar";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "GiantStar";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.GiantStarStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.GiantStarCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.GiantStarScale, Constants.GiantStarScale, Constants.GiantStarScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

void SpawnNeutronStar(Vector3 vc)

{

Object neutronStarsPrefab = prefabs.stars[4] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(neutronStarsPrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "NeutronStar";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "NeutronStar";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.NeutronStarStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.NeutronStarCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.NeutronStarScale, Constants.NeutronStarScale, Constants.NeutronStarScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

void SpawnBlackHole(Vector3 vc)

{

Object blackholePrefab = prefabs.BlackHole[0] as GameObject;

var Pos = player.GetComponent<Transform>().position + vc \* Constants.DistanceToGenerateObjects;

var newplanet = Instantiate(blackholePrefab, Pos, this.transform.rotation) as GameObject;

newplanet.tag = "BlackHole";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "BlackHole";

newplanet.GetComponent<GravitationalBody>().StartingMass = Constants.BlackHoleStartingMass + 3;

newplanet.GetComponent<GravitationalBody>().ImaginaryMass = Constants.BlackHoleCriticalMass;

newplanet.GetComponent<Rigidbody2D>().mass = newplanet.GetComponent<GravitationalBody>().StartingMass;

newplanet.transform.localScale = new Vector3(Constants.BlackHoleScale, Constants.BlackHoleScale, Constants.BlackHoleScale); // УБРАЛ УМНОЖИТЬ РАВНО

}

public void SpawnAsteroidConstantPosition(Vector3 vc)

{

Object asteroidPrefab = prefabs.asteroids[rnd.Next(0, prefabs.asteroids.Length)] as GameObject;

var newplanet = Instantiate(asteroidPrefab, vc, this.transform.rotation) as GameObject;

newplanet.tag = "Asteroid";

newplanet.AddComponent<GravitationalBody>();

newplanet.GetComponent<GravitationalBody>().name = "Asteroid";

newplanet.transform.localScale \*= Constants.AsteroidScale;

Material[] materials = new Material[] { prefabs.asteroidsMaterials[rnd.Next(0, prefabs.asteroidsMaterials.Length)] };

newplanet.GetComponent<MeshRenderer>().materials = materials;

}

}

## Класс TextTyper

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class TextTyper : MonoBehaviour

{

// Start is called before the first frame update

[SerializeField]

public Text uiText;

public string firstText = "";

public string secondText = "";

IEnumerator Start()

{

for (int i = 0; i < firstText.Length; i++)

{

yield return new WaitForSeconds(0.08f);

uiText.text = firstText.Substring(0, i + 1);

}

yield return new WaitForSeconds(1.5f);

uiText.text = "";

yield return new WaitForSeconds(2f);

for (int i = 0; i < secondText.Length; i++)

{

yield return new WaitForSeconds(0.04f);

uiText.text = secondText.Substring(0, i + 1);

}

yield return new WaitForSeconds(2f);

uiText.text = "";

}

public IEnumerator TypeText(string text)

{

uiText.text = "";

for (int i = 0; i < text.Length; i++)

{

yield return new WaitForSeconds(0.04f);

uiText.text = text.Substring(0, i + 1);

}

yield return new WaitForSeconds(2f);

uiText.text = "";

}

public IEnumerator TypeText(string text, string text2)

{

uiText.text = "";

for (int i = 0; i < text.Length; i++)

{

yield return new WaitForSeconds(0.04f);

uiText.text = text.Substring(0, i + 1);

}

yield return new WaitForSeconds(2f);

uiText.text = "";

for (int i = 0; i < text2.Length; i++)

{

yield return new WaitForSeconds(0.04f);

uiText.text = text2.Substring(0, i + 1);

}

yield return new WaitForSeconds(2f);

uiText.text = "";

}

}

## Класс TrashCollector

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class TrashCollector : MonoBehaviour

{

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void OnCollisionEnter2D(Collision2D collision)

{

Destroy(collision.gameObject);

Debug.Log("deleted");

}

}

## Класс UIController

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using UnityEngine;

using UnityEngine.UI;

public class UIController : MonoBehaviour

{

public Text planetName;

public Text Mass;

public Text Vector;

public Text Coords;

public Text nextObject;

public GameObject player;

private GravitationalBody scr;

// Start is called before the first frame update

void Start()

{

scr = player.GetComponent<GravitationalBody>();

planetName.text = "Type: planet";

Mass.text = "Mass: -1";

Vector.text = "Vector2: <0,0,0>";

Coords.text = "Coordinates: <0,0,0>";

}

// Update is called once per frame

void FixedUpdate()

{

planetName.text = "" + scr.ShareObjectData("name");

Mass.text = "Mass: " + scr.ShareObjectData("mass");;

Vector.text = "Vector2: <" + scr.ShareObjectData("vector") + ">" ;

Coords.text = "Coordinates: <" + scr.ShareObjectData("coords") + ">";

nextObject.text = "Next: " + Constants.PlayersNextObject;

}

void FetchInfoLabels() {

}

}

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.301-79 Программа и методика испытаний. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.401-78 Текст программы. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.505-79 Руководство оператора. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.404-79 Пояснительная записка. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. https://github.com [Электронный ресурс] / Казанцев Никита. – Режим доступа: https://github.com/isp13/Accretion, свободный. (дата обращения: 13.05.20).

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |