МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Кафедра КБ-9 «Предметно-ориентированные информационные системы»

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2

по дисциплине: «Алгоритмические основы разработки симуляторов»

на тему: «”Кукурузник”- самолёт из мудульных деталей»

Выполнил: студент группы БСБО-08-23

Гончаров А.С.

Проверил: ассистент кафедры КБ-9

Малько Е. И.

Москва, 2025

Разработана трёхмерная интерактивная сцена полёта лёгкого самолёта (типа «кукурузник»), реализованная в игровом движке Unity. Управление осуществляется с клавиатуры и мыши. В проекте реализована физическая модель, приближённая к реальной аэродинамике, с учётом подъёмной силы, лобового сопротивления, влияния высоты, ветра, турбулентности и сваливания.

Основным компонентом физической модели является скрипт AircraftPhysics.cs (см. рис. 1-5), прикреплённый к объекту самолёта и работающий в связке с Rigidbody. В нём реализован расчёт подъёмной силы и лобового сопротивления по классическим формулам $F_L = ½·ρ·V²·S·C_L(α)$ и $F_D = ½·ρ·V²·S·C_D(|α|)$, где коэффициенты подъёмной силы и сопротивления задаются через редактируемые AnimationCurve в зависимости от угла атаки. Плотность воздуха уменьшается с высотой по экспоненциальному закону $ρ = ρ_0·\exp(-h/8000)$, тяга двигателя падает с высотой по аналогичной зависимости, что имитирует поведение поршневого двигателя без наддува.

```
#region Unity callbacks
0 references | Unity Message
private void Awake()
{
    rb = GetComponent<Rigidbody>();
    input = GetComponent<PlayerController>();
    fuelRemaining = fuelCapacity;

    if (liftCurve.keys.Length == 0)
        SetupDefaultCurves();

    UpdateMass();
}

0 references | Unity Message
private void FixedUpdate()
{
    CalculateCurrentState();
    ApplyLiftAndDrag();
    ApplyEngineThrust();
    ApplyWindAndTurbulence();
    HandleStallBehaviour();
    ConsumeFuelAndUpdateMass();
}
#endregion

#region Curves initialization
1 reference
private void SetupDefaultCurves()
{
    liftCurve = new AnimationCurve(
        new Keyframe(-20f, -0.5f), new Keyframe(-5f, 0f),
        new Keyframe(5f, 0.5f),    new Keyframe(15f, 1.2f),
        new Keyframe(20f, 0.8f),   new Keyframe(25f, 0.5f)
    );
```

Рисунок 1. Скрипт AircraftPhysics.cs, часть 1

```
        dragCurve = new AnimationCurve(
            new Keyframe(-20f, 0.8f), new Keyframe(0f, 0.05f),
            new Keyframe(15f, 0.1f),  new Keyframe(20f, 0.3f),
            new Keyframe(25f, 0.5f)
        );
    }
    #endregion

    #region Flight state
    1 reference
    private void CalculateCurrentState()
    {
        Vector3 relativeAir = rb.linearVelocity - windVelocity;
        currentAirspeed = relativeAir.magnitude;

        if (currentAirspeed > 0.1f)
        {
            Vector3 localVel = transform.InverseTransformDirection(relativeAir);
            currentAoA = Mathf.Atan2(-localVel.y, localVel.z) * Mathf.Rad2Deg;
        }
        else
            currentAoA = 0f;
    }
    #endregion

    #region Aerodynamics
    1 reference
    private void ApplyLiftAndDrag()
    {
        if (currentAirspeed < 0.1f) return;

        float density = rhoSeaLevel * Mathf.Exp(-transform.position.y / 8000f);
        float dynamicPressure = 0.5f * density * currentAirspeed * currentAirspeed;
        Vector3 airflow = rb.linearVelocity - windVelocity;
```

Рисунок 2. Скрипт AircraftPhysics.cs, часть 2

Угол атаки определяется автоматически по вектору относительной скорости воздуха. При превышении критического угла атаки (12°) происходит сваливание: подъёмная сила резко снижается в три раза, сопротивление возрастает вдвое, добавляются случайные возмущения и турбулентность.

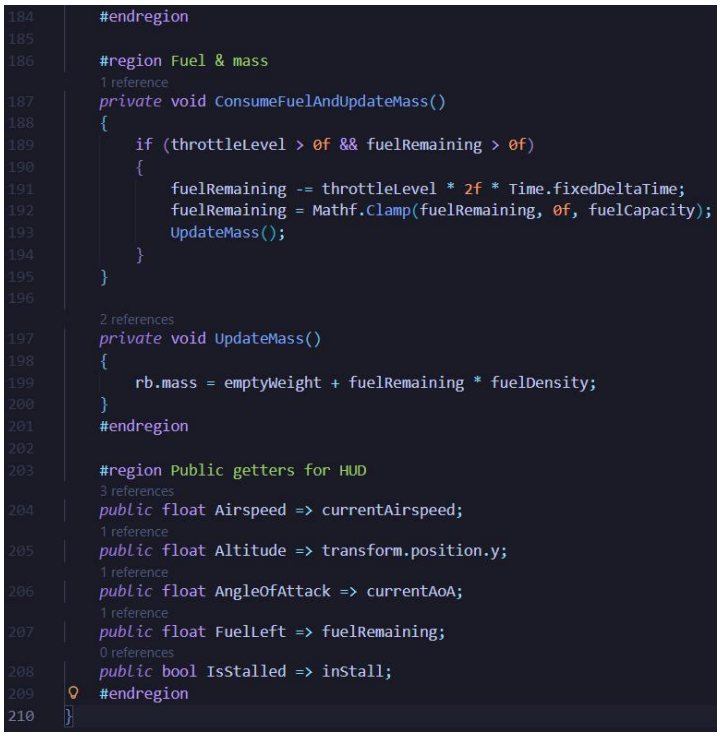
```
        float Cl = liftCurve.Evaluate(currentAoA);
        float Cd = dragCurve.Evaluate(Mathf.Abs(currentAoA));

        if (inStall)
        {
            Cl *= 0.3f;
            Cd *= 2f;
        }

        Vector3 liftDirection = Vector3.Cross(airflow.normalized, transform.right).normalized;
        rb.AddForce(liftDirection * dynamicPressure * wingArea * Cl);
        rb.AddForce(-airflow.normalized * dynamicPressure * wingArea * Cd);
    }
    #endregion

    #region Engine
    1 reference
    private void ApplyEngineThrust()
    {
        throttleLevel = input.isBoosting ? 1f : 0.5f;

        if (throttleLevel > 0f && fuelRemaining > 0f)
        {
            float altitudeFactor = Mathf.Exp(-transform.position.y / altitudeForHalfThrust);
            float thrust = throttleLevel * maxThrust * altitudeFactor;
            rb.AddForce(transform.forward * thrust);
        }
    }
    #endregion

    #region Environment effects
    1 reference
    private void ApplyWindAndTurbulence()
    {
        rb.AddForce(windVelocity * rb.mass * 0.1f);
```

Рисунок 3. Скрипт AircraftPhysics.cs, часть 3

```
        if (turbulencePower > 0f)
        {
            Vector3 turb = turbulencePower * currentAirspeed * new Vector3(
                Mathf.PerlinNoise(Time.time * turbulenceSpeed, 0f) - 0.5f,
                Mathf.PerlinNoise(0f, Time.time * turbulenceSpeed) - 0.5f,
                Mathf.PerlinNoise(Time.time * turbulenceSpeed, Time.time * turbulenceSpeed) - 0.5f
            );

            rb.AddForce(turb);
        }
    }
    #endregion

    #region Stall logic
    1 reference
    private void HandleStallBehaviour()
    {
        if (!inStall && Mathf.Abs(currentAoA) > stallAngle)
        {
            inStall = true;
            timeInStall = 0f;
        }

        if (inStall)
        {
            timeInStall += Time.fixedDeltaTime;

            if (timeInStall < 3f)
            {
                rb.AddForce(turbulencePower * 2f * Random.insideUnitSphere * rb.mass);
                rb.AddTorque(Random.insideUnitSphere * 0.1f);
            }

            if (Mathf.Abs(currentAoA) < stallAngle * 0.7f && timeInStall > 2f)
                inStall = false;
        }
```

Рисунок 4. Скрипт AircraftPhysics.cs, часть 4

Реализованы постоянный ветер, перлиновская турбулентность, расход топлива с изменением массы самолёта, что влияет на инерцию и поведение в воздухе. Все силы прикладываются через Rigidbody.AddForce в FixedUpdate. Скрипт предоставляет публичные свойства Airspeed, Altitude, AngleOfAttack, FuelLeft и IsStalled для использования другими системами.



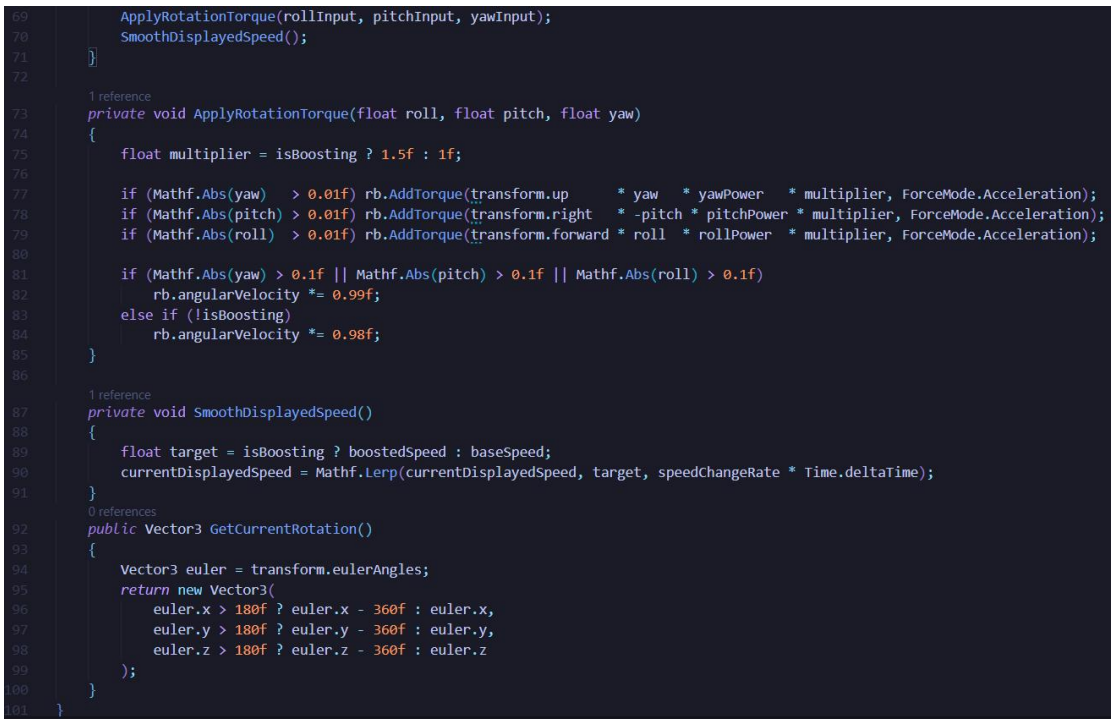
```
    #endregion

    #region Fuel & mass
    1 reference
    private void ConsumeFuelAndUpdateMass()
    {
        if (throttleLevel > 0f && fuelRemaining > 0f)
        {
            fuelRemaining -= throttleLevel * 2f * Time.fixedDeltaTime;
            fuelRemaining = Mathf.Clamp(fuelRemaining, 0f, fuelCapacity);
            UpdateMass();
        }
    }

    2 references
    private void UpdateMass()
    {
        rb.mass = emptyWeight + fuelRemaining * fuelDensity;
    }
    #endregion

    #region Public getters for HUD
    3 references
    public float Airspeed => currentAirspeed;
    1 reference
    public float Altitude => transform.position.y;
    1 reference
    public float AngleOfAttack => currentAoA;
    1 reference
    public float FuelLeft => fuelRemaining;
    0 references
    public bool IsStalled => inStall;
    #endregion
}
```

Рисунок 5. Скрипт AircraftPhysics.cs, часть 5

Управление самолётом реализовано в скрипте PlayerController.cs (см. рис. 6-7). Ось X мыши отвечает за крен, клавиши W/S и A/D — за тангаж и рыскание, Q/E — за дополнительный крен. Зажатый LeftShift переключает режим двигателя на полную тягу (throttle = 1.0), при отпущенной клавише поддерживается половинная тяга (0.5). Свойство isBoosting передаётся в AircraftPhysics и определяет текущую тягу. Добавлено демпфирование угловой скорости и метод GetCurrentRotation(), возвращающий углы Эйлера в диапазоне ±180° для корректного отображения на приборной панели.

```
35      0 references | Unity Message
        private void Awake()
36      {
37          rb = GetComponent<Rigidbody>();
38          Cursor.lockState = CursorLockMode.Locked;
39          currentDisplayedSpeed = baseSpeed;
40      }
41
        0 references | Unity Message
42      private void Update()
43      {
44          mouseHorizontal = Input.GetAxis("Mouse X") * mouseSens;
45          isBoosting = Input.GetKey(KeyCode.LeftShift);
46
47          if (Input.GetKeyDown(KeyCode.Escape))
48          {
49  #if UNITY_EDITOR
50              UnityEditor.EditorApplication.isPlaying = false;
51  #else
52              Application.Quit();
53  #endif
54          }
55      }
56
        0 references | Unity Message
57      private void FixedUpdate()
58      {
59          if (flightPhysics == null || flightPhysics.Airspeed < 2f)
60              rb.linearVelocity = transform.forward * currentDisplayedSpeed;
61
62          float rollInput  = -mouseHorizontal;
63          float pitchInput = Input.GetAxis("Vertical");
64          float yawInput   = Input.GetAxis("Horizontal");
65
66          if (Input.GetKey(KeyCode.Q)) rollInput += 1f;
67          if (Input.GetKey(KeyCode.E)) rollInput -= 1f;
```

Рисунок 6. Скрипт PlayerController.cs, часть 1



```
        ApplyRotationTorque(rollInput, pitchInput, yawInput);
        SmoothDisplayedSpeed();
    }

    1 reference
    private void ApplyRotationTorque(float roll, float pitch, float yaw)
    {
        float multiplier = isBoosting ? 1.5f : 1f;

        if (Mathf.Abs(yaw)   > 0.01f) rb.AddTorque(transform.up      * yaw   * yawPower   * multiplier, ForceMode.Acceleration);
        if (Mathf.Abs(pitch) > 0.01f) rb.AddTorque(transform.right   * -pitch * pitchPower * multiplier, ForceMode.Acceleration);
        if (Mathf.Abs(roll)  > 0.01f) rb.AddTorque(transform.forward * roll  * rollPower  * multiplier, ForceMode.Acceleration);

        if (Mathf.Abs(yaw) > 0.1f || Mathf.Abs(pitch) > 0.1f || Mathf.Abs(roll) > 0.1f)
            rb.angularVelocity *= 0.99f;
        else if (!isBoosting)
            rb.angularVelocity *= 0.98f;
    }

    1 reference
    private void SmoothDisplayedSpeed()
    {
        float target = isBoosting ? boostedSpeed : baseSpeed;
        currentDisplayedSpeed = Mathf.Lerp(currentDisplayedSpeed, target, speedChangeRate * Time.deltaTime);
    }
    0 references
    public Vector3 GetCurrentRotation()
    {
        Vector3 euler = transform.eulerAngles;
        return new Vector3(
            euler.x > 180f ? euler.x - 360f : euler.x,
            euler.y > 180f ? euler.y - 360f : euler.y,
            euler.z > 180f ? euler.z - 360f : euler.z
        );
    }
}
```

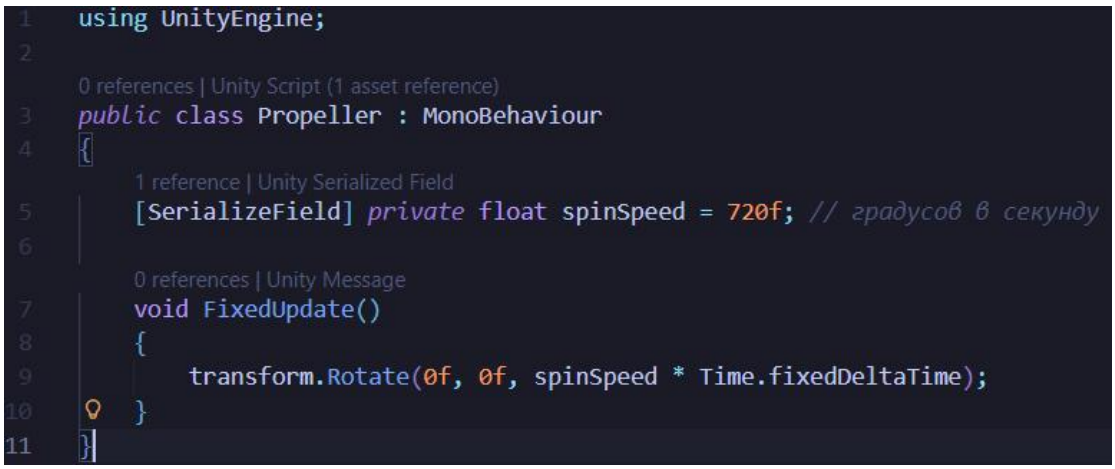
Рисунок 7. Скрипт PlayerController.cs, часть 2

Отображение лётных параметров выполнено в скрипте MainMenu.cs (см. рис. 8) исключительно через TextMeshProUGUI. В реальном времени выводятся: скорость в км/ч, остаток топлива в процентах, крен, тангаж и рыскание с форматом ±000°, угол атаки с точностью 0.1°, воздушная скорость в м/с и текущая высота. Данные считываются напрямую из AircraftPhysics и PlayerController, обновление происходит каждый кадр в методе Update. Слайдеры и графические индикаторы удалены для упрощения и повышения читаемости.

```
    private AircraftPhysics phys => physics ? physics : player?.GetComponent<AircraftPhysics>();
    2 references
    private Transform plane => physics ? physics.transform : player?.transform;

    0 references | Unity Message
    private void Update()
    {
        if (!phys || !plane) return;

        float kmh = phys.Airspeed * 3.6f;
        float fuelPct = phys.FuelLeft / 200f * 100f;

        if (speedText)     speedText.text     = $"Скорость: {(int)kmh} км/ч";
        if (fuelText)      fuelText.text      = $"Топливо: {(int)fuelPct}%";

        Vector3 e = plane.eulerAngles;
        float roll  = e.z > 180f ? e.z - 360f : e.z;
        float pitch = e.x > 180f ? e.x - 360f : e.x;
        float yaw   = e.y > 180f ? e.y - 360f : e.y;

        if (rollText)  rollText.text  = $"Крен: {roll:+000;-000}°";
        if (pitchText) pitchText.text = $"Тангаж: {pitch:+000;-000}°";
        if (yawText)   yawText.text   = $"Рыскание: {yaw:+000;-000}°";

        if (aoaText)      aoaText.text      = $"AoA: {phys.AngleOfAttack:+00.0;-00.0}°";
        if (airspeedText) airspeedText.text = $"Воздушная скорость: {(int)phys.Airspeed} м/с";
        if (altitudeText) altitudeText.text = $"Высота: {(int)phys.Altitude} м";
    }
}
```

Рисунок 8. Скрипт MainMenur.cs

Вращение воздушного винта реализовано в минималистичном скрипте Propeller.cs (см. рис. 9) путём постоянного увеличения локального угла поворота по оси Z в FixedUpdate.



```
using UnityEngine;

0 references | Unity Script (1 asset reference)
public class Propeller : MonoBehaviour
{
    1 reference | Unity Serialized Field
    [SerializeField] private float spinSpeed = 720f; // градусов в секунду

    0 references | Unity Message
    void FixedUpdate()
    {
        transform.Rotate(0f, 0f, spinSpeed * Time.fixedDeltaTime);
    }
}
```

Рисунок 9. Скрипт Propeller.cs

Для визуального отображения использовался terrain для локации, простые геометрические фигуры для модели самолёта (см. рис. 10).

Рисунок 10. Самолёт и terrain

Вывод

Реализованная физическая модель обеспечивает правдоподобное поведение самолёта на всех режимах полёта, включая набор высоты, сваливание, влияние ветра и турбулентности, что позволяет демонстрировать основные аэродинамические явления.

Ссылка на гитхаб: https://github.com/Angrock/Kukuruznik.git