Лабораторная работа №4:
Процедурная анимация в Unity

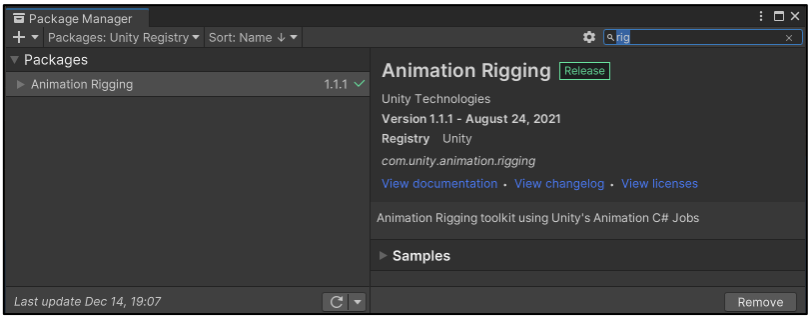
Цель:

Целью данной работы является получение навыков работы с процедурной анимацией в проектах Unity.

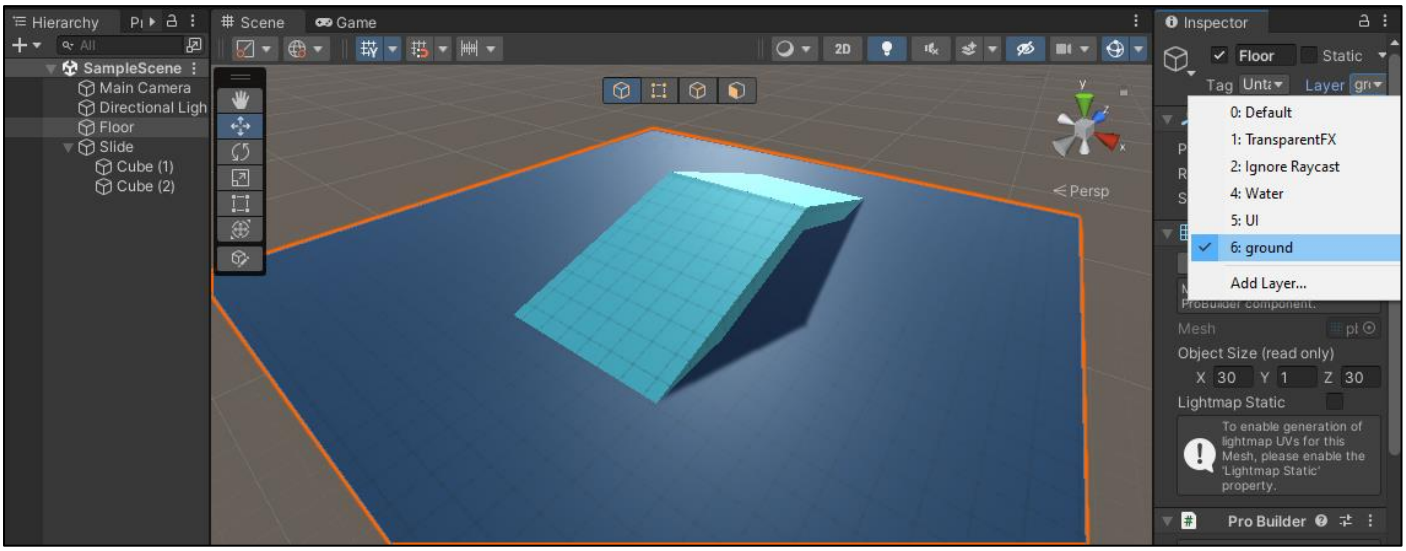
Справка:
Модель, используемая в лабораторной работе, доступна для скачивания в Unity Asset Store: https://assetstore.unity.com/packages/3d/characters/robots/spider-orange-181154

Подготовка сцены

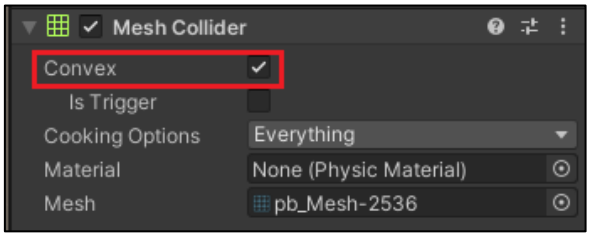
Для работы с процедурной анимацией необходимо установить пакет Animation Rigging:



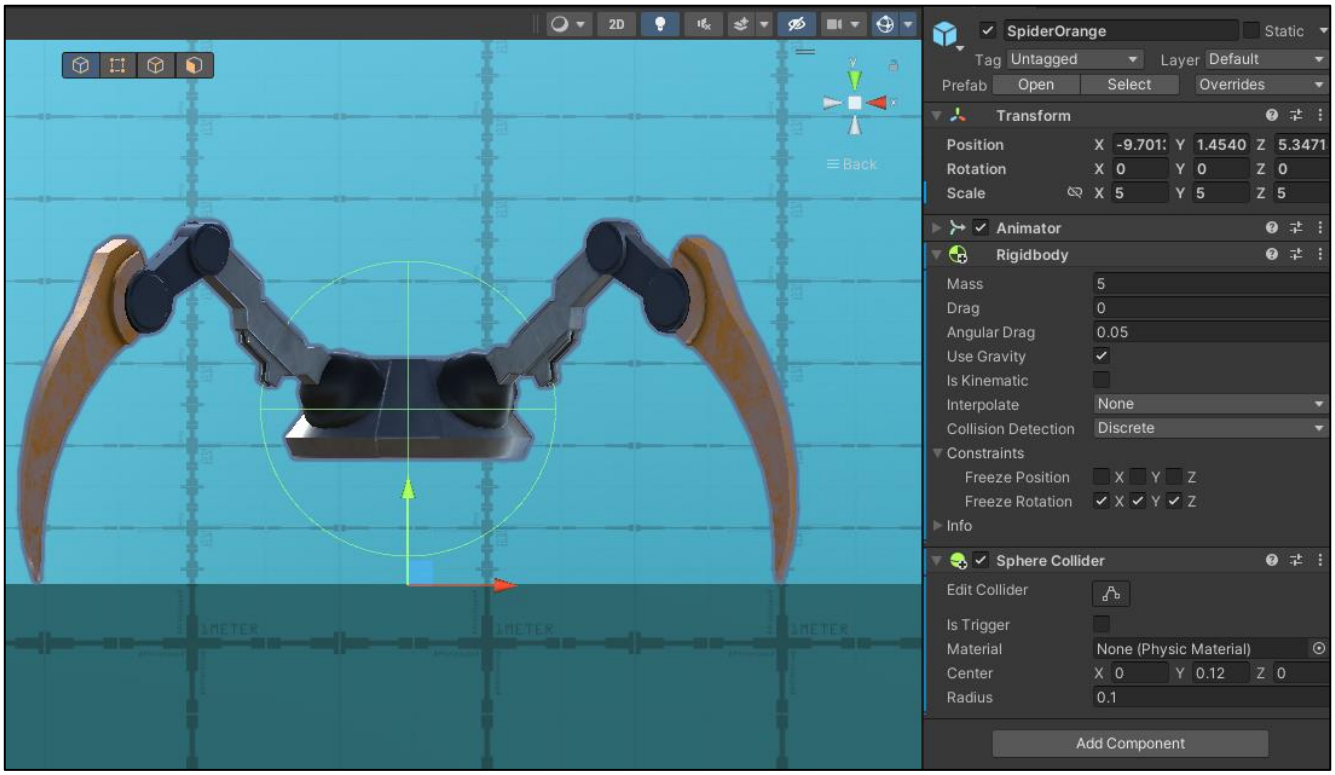
В сцене следует создать объекты “пол” и “горка”, объекты следует разместить на отдельном слое:



В случае, если в качестве ограничивающего объёма объектов используется Mesh Collider, не забудьте отметить их как Convex:



В завершении, добавьте в сцену желаемую модель, прикрепив к ней ограничивающий объём и физическое тело (если они отсутствуют):



Существуют два способа обработки перемещений “тела” подобных объектов, на основе физического тела и его ограничивающего объёма, и на основе определения позиции “тела” исходя из позиции его ног. В данном примере используется первый метод.

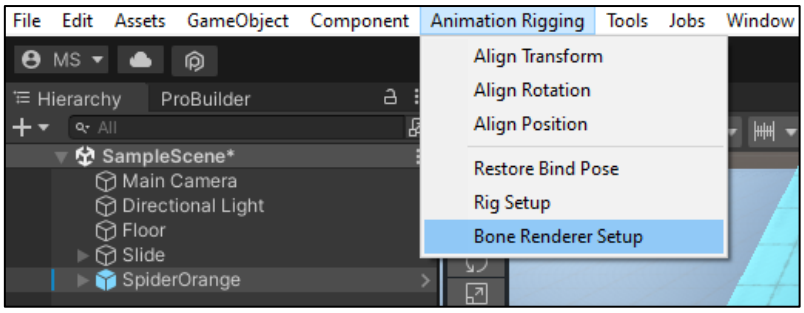
Скрипт управления перемещением

Прикрепите к модели скрипт управления следующего вида:

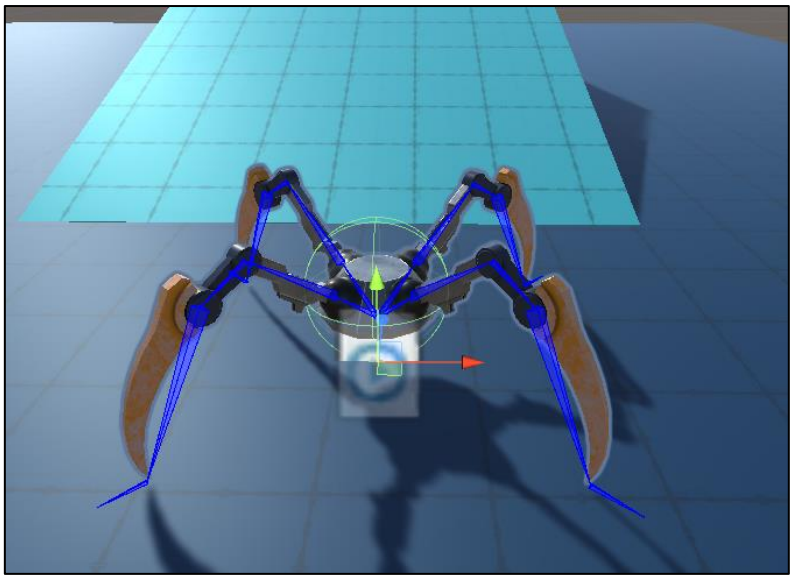
```
public class Controller : MonoBehaviour
{
    public float speed = 4;         // скорость перемещения
    public float jumpForse = 500;   // сила прыжка

    Rigidbody rb;

    Ссылок: 0
    void Start()...

    Ссылок: 0
    void LateUpdate()
    {
        float v = Input.GetAxisRaw("Vertical");                 // получение направления движения
        float h = Input.GetAxisRaw("Horizontal");               // по нажатым клавишам

        Vector3 dir = new Vector3(h, 0, v);
        dir = transform.TransformDirection(dir.normalized);     // применение поворота объекта к вектору направления движения
        dir *= speed;                                           // применение скорости к вектору направления движения
        dir.y = rb.velocity.y;                                  // восстановление смещения по оси Y
        rb.velocity = dir;

        if (Input.GetKeyDown(KeyCode.Space))                    // обработка нажатия на пробел
        {
            rb.AddForce(Vector3.up * jumpForse);                // придание объекту импульса по направлению вверх
        }

        if (Input.GetKey(KeyCode.Q))                            // обработка нажатия на кнопку Q
        {
            transform.Rotate(Vector3.up, -Mathf.PI/8, Space.World);// поворот объекта против часовой стрелки
        }

        if (Input.GetKey(KeyCode.E))                            // обработка нажатия на кнопку E
        {
            transform.Rotate(Vector3.up, Mathf.PI / 8, Space.World);// поворот объекта по часовой стрелке
        }
    }
}
```

Добавление ограничений

Выберите модель и добавьте к ней Bone Renderer Setup:

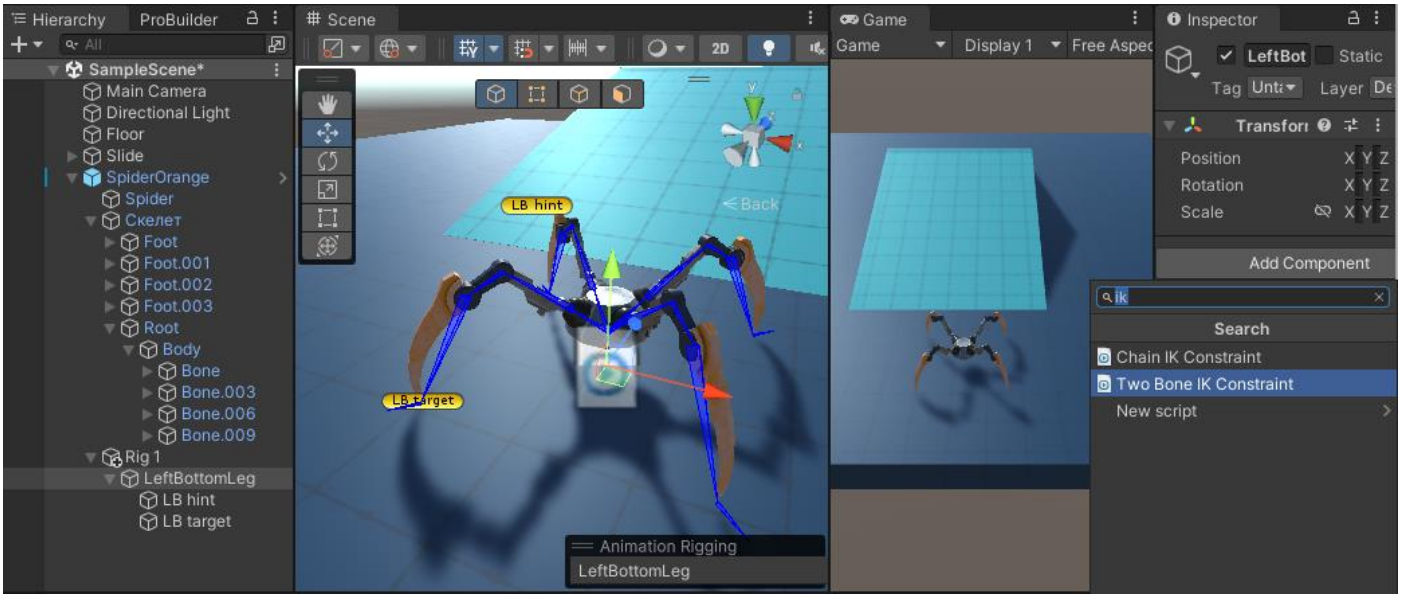


В результате, на модели отобразится её скелет:



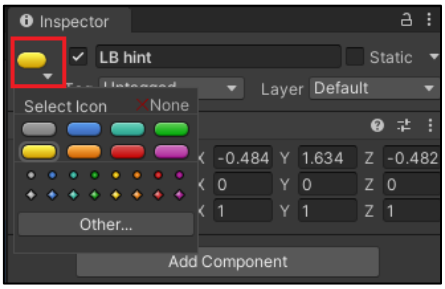
Этот шаг не является обязательным, но может быть полезен для понимания структуры модели.

Добавьте к модели Rig Setup, а затем в Rig1, создайте иерархию пустых объектов, как показано на изображении ниже:

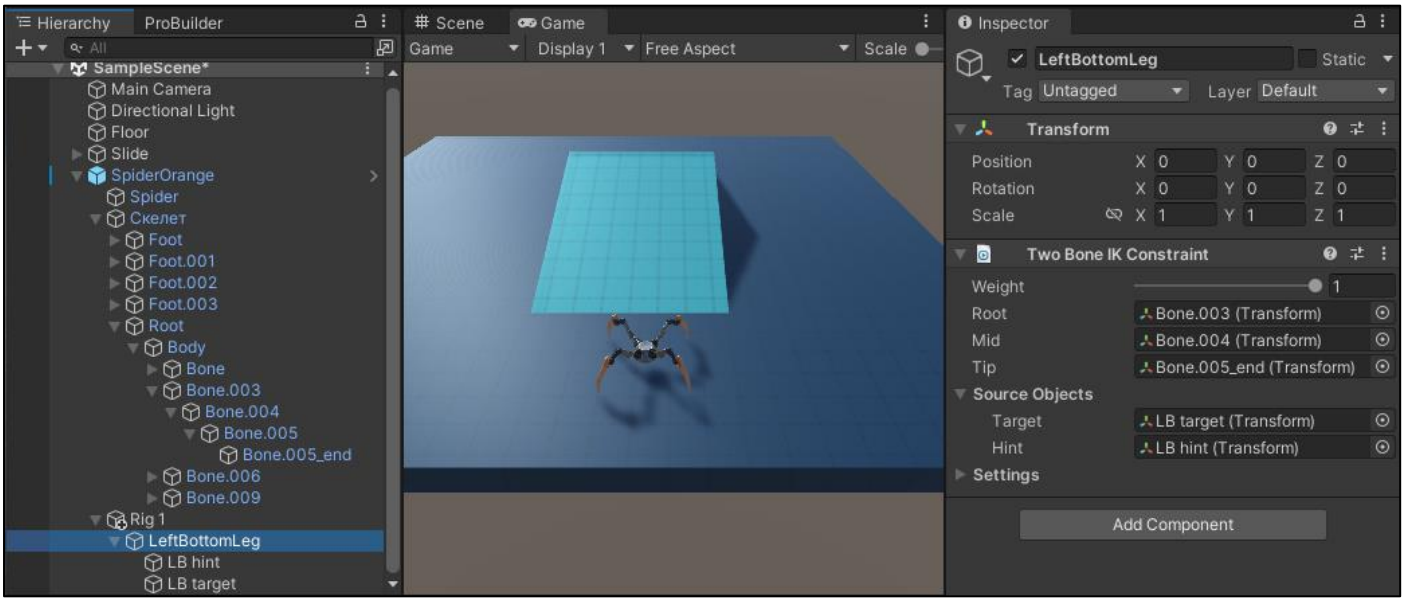


Разместите объект hint над центром левой задней ноги модели, а объект target под окончанием левой задней ноги, после чего, прикрепите к корневому объекту скрипт Two Bone IK Constraint.

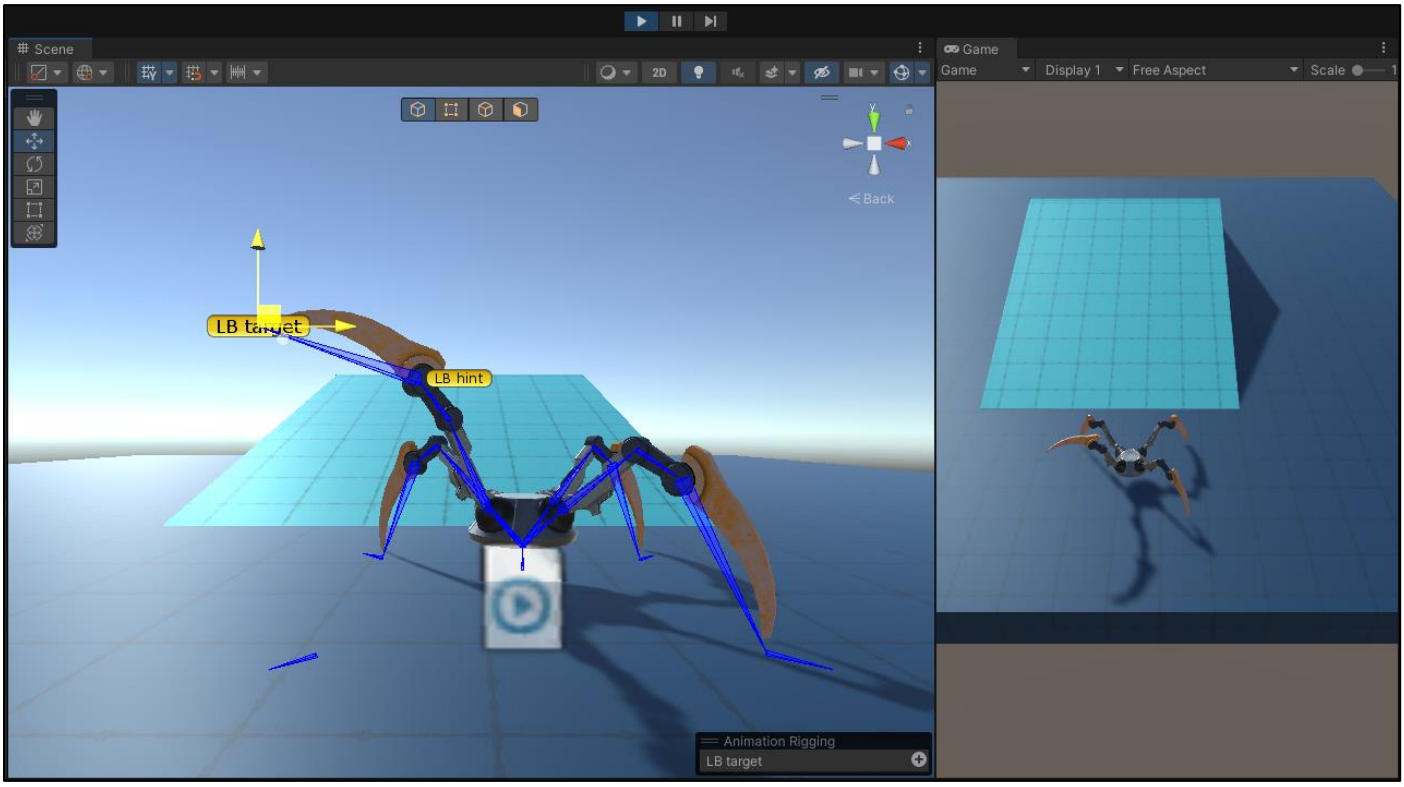
Сделать видимыми пустые объекты можно кликнув соответствующую иконку в инспекторе и выбрав цвет:



Заполните параметры скрипта следующим образом:

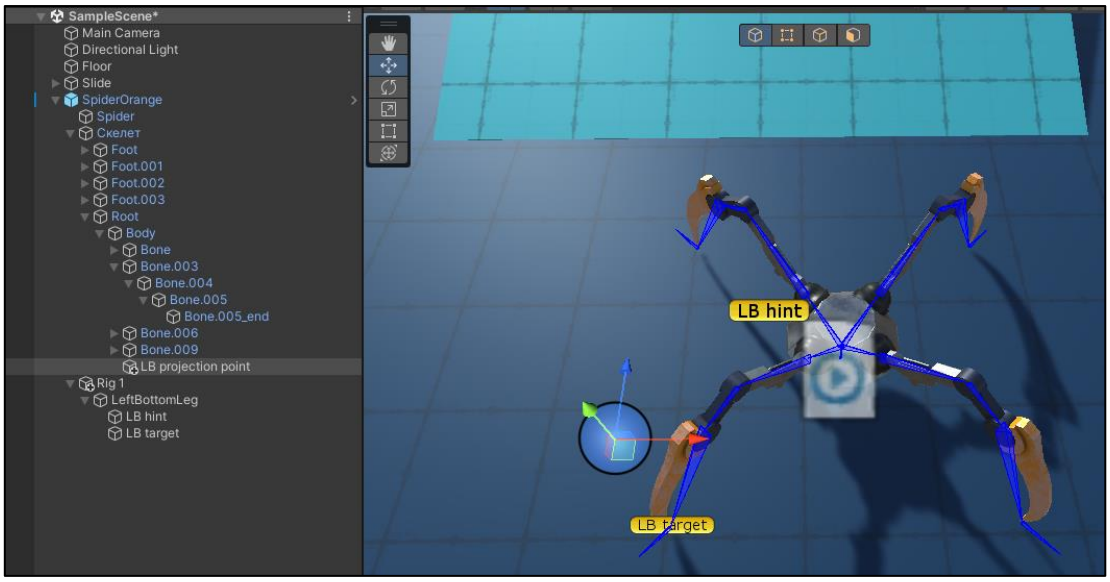


В случае, если всё было сделано правильно, при запуске проекта, перемещая target конечности, можно перемещать саму конечность, а перемещая hint, можно управлять её наклоном:



Автоматизация расчёта позиции target

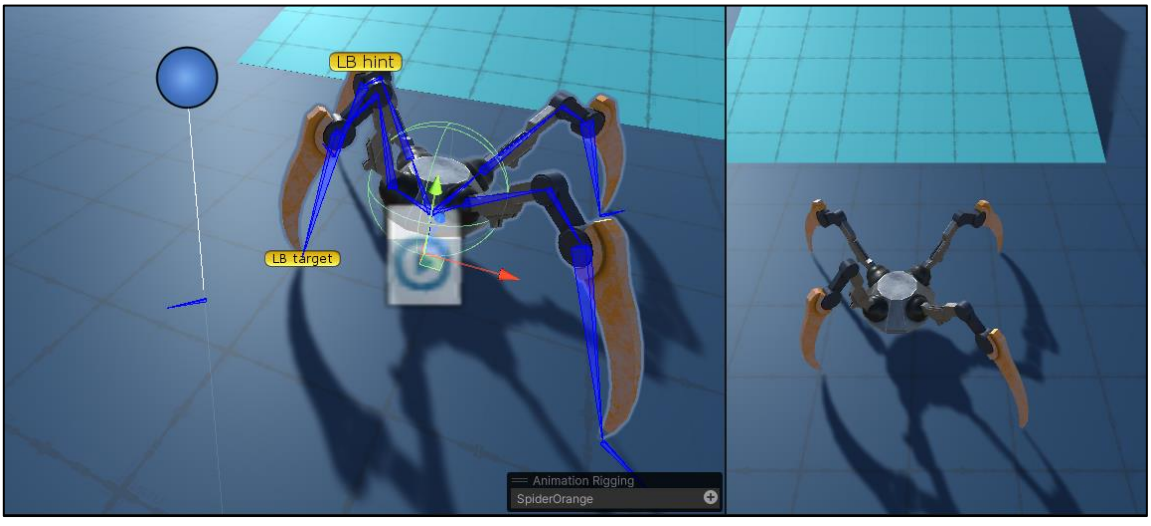
Добавьте к телу объекта точку (empty object), расположите её над позицией target:



Прикрепите к объекту target скрипт следующего вида:

```
public class LegScript : MonoBehaviour
{
    public Transform projector;         // ссылка на точку из которой будет осуществляться проекция
    public LayerMask ground;            // слой поверхностей, по которым допустимо перемещение

    Vector3 oldPosition;                // изначальная позиция конечности на поверхности
    RaycastHit hit;

    Ссылок: 0
    void Start()
    {
        Physics.Raycast(projector.position, Vector3.down, out hit, 100, ground);    // построение луча из точки проекции вертикально вниз
        oldPosition = hit.point;    // сохранение точки пересечения луча и поверхности
    }

    Ссылок: 0
    void LateUpdate()
    {
        transform.position = oldPosition;   // возврат позиции конечности в начальную точку перемещения
    }

    Ссылок: 0
    private void OnDrawGizmos()
    {
        Gizmos.DrawRay(projector.position, Vector3.down * 100);     // рисование луча в окне сцены
    }
}
```

Если всё было сделано правильно, то после запуска проекта, при перемещении модели, левая задняя нога будет стараться сохранить исходную позицию:

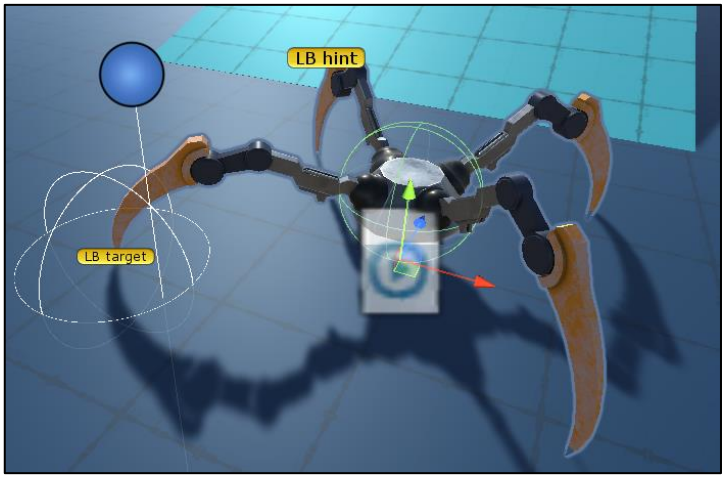


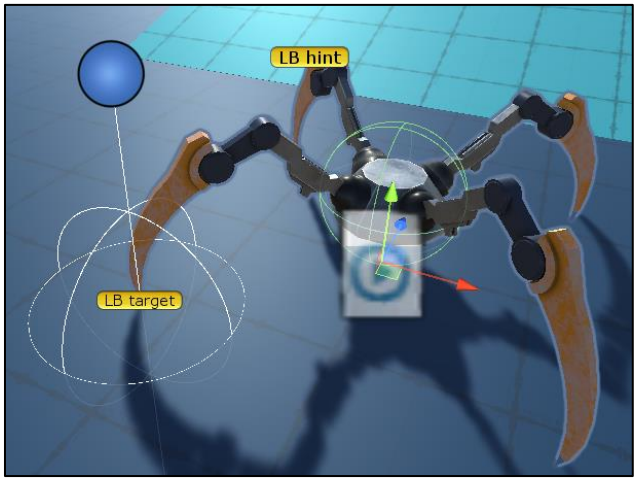
Реализация перемещения конечности

Для реализации перемещения конечности вслед за перемещением модели, необходимо добавить к скрипту конечности дистанцию перемещения модели, относительно начального положения, после превышения которой, будет совершаться шаг. Затем, каждый кадр, необходимо вычислять разницу между начальной и текущей проекцией конечности и когда эта разница превысит заданную дистанцию, изменять позицию конечности:

```
public class LegScript : MonoBehaviour
{
    public Transform projector;         // ссылка на точку из которой будет осуществляться проекция
    public LayerMask ground;            // слой поверхностей, по которым допустимо перемещение

    float stepDistance = 0.7f;          // дистанция шага

    Vector3 oldPosition;
    RaycastHit hit;

    Ссылок: 0
    void Start()...

    Ссылок: 0
    void LateUpdate()
    {
        transform.position = oldPosition;   // возврат позиции конечности в начальную точку перемещения

        Vector3 proj1 = Vector3.ProjectOnPlane(projector.position, Vector3.up);         // проекция точки построения луча на плоскость X|Z
        Vector3 proj2 = Vector3.ProjectOnPlane(oldPosition, Vector3.up);                // проекция текущей позиции конечности на плоскость X|Z

        if (Vector3.Distance(proj1, proj2) > stepDistance)                              // если расстояние между проекциями точек больше дистанции шага
        {
            Physics.Raycast(projector.position , Vector3.down, out hit, 100, ground);   // нахождение новой позиции для конечности
            oldPosition = hit.point;
        }
    }

    Ссылок: 0
    private void OnDrawGizmos()
    {
        Gizmos.DrawRay(projector.position, Vector3.down * 100);     // визуализация луча
        Gizmos.DrawWireSphere(transform.position, stepDistance);    // визуализация дистанции шага в виде сферы
    }
}
```

На изображениях ниже, проекция новой позиции конечности отмечена линией, а допустимая разница старой и новой позиций отображена сферой:





Для реализации более плавного движения, необходимо расширить список параметров шага:

```
    public Transform projector;         // ссылка на точку из которой будет осуществляться проекция
    public LayerMask ground;            // слой поверхностей, по которым допустимо перемещение

    float stepDistance = 0.7f;          // дистанция шага
    float stepSpeed = 8;                // скорость шага
    float stepHeight = 1;               // высота шага
    float overshoot = 0.8f;             // дополнительно смещение конечности по направлению движения

    Vector3 oldPosition, newPosition, currentPosition;

    RaycastHit hit;
    float lerp = 1; // коэффициент интерполяции
```

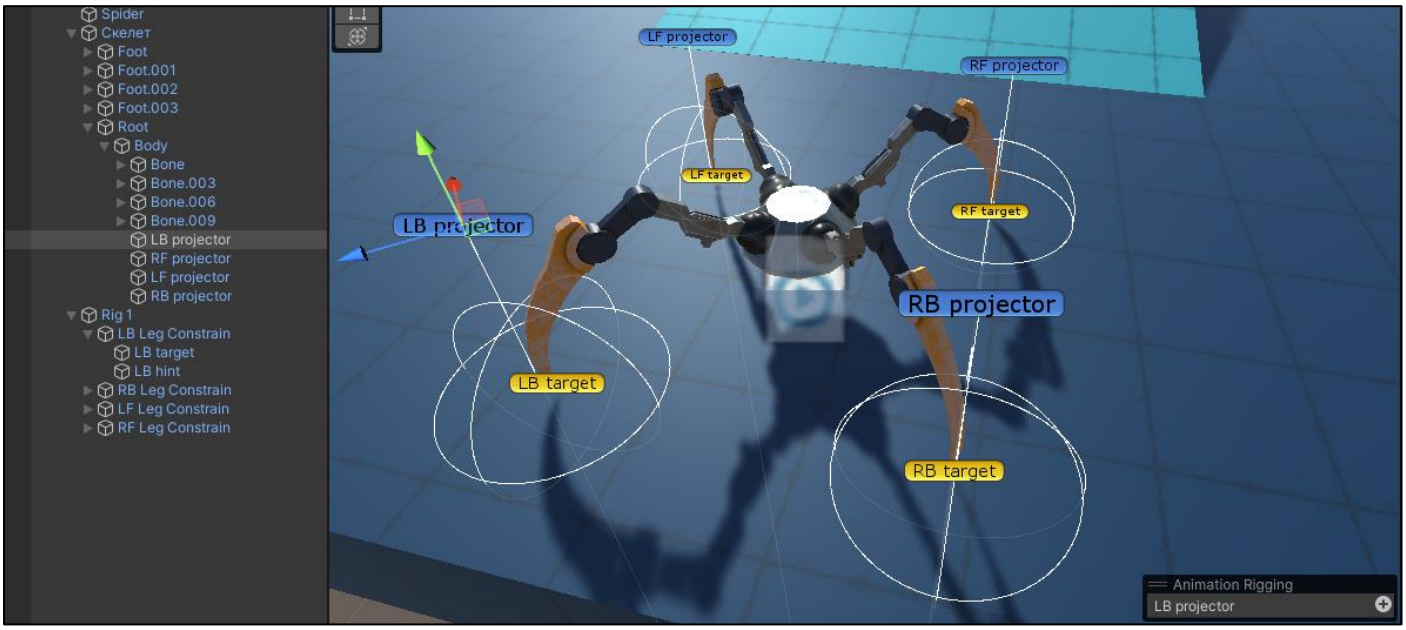
И модифицировать метод перемещения конечности:

```
void LateUpdate()
{
    transform.position = currentPosition;

    Vector3 proj1 = Vector3.ProjectOnPlane(projector.position, Vector3.up);         // проекция точки построения луча на плоскость X|Z
    Vector3 proj2 = Vector3.ProjectOnPlane(currentPosition, Vector3.up);            // проекция текущей позиции конечности на плоскость X|Z

    if (Vector3.Distance(proj1, proj2) > stepDistance && lerp >= 1)
    {
        Physics.Raycast(projector.position + (proj1 - proj2) * overshoot, Vector3.down, out hit, 100, ground);

        newPosition = hit.point;
        lerp = 0;
    }

    if (lerp < 1)
    {
        Vector3 tempPosition = Vector3.Lerp(oldPosition, newPosition, lerp);    // вычисление промежуточного значения между старой и новой позициями
        tempPosition.y += Mathf.Sin(lerp * Mathf.PI) * stepHeight;              // вычисление высоты конечности

        currentPosition = tempPosition;
        lerp += Time.deltaTime * stepSpeed;
    }
    else
    {
        oldPosition = newPosition;
    }
}
```

Если всё было сделано правильно, то при запуске проекта и перемещении модели, задняя левая конечность должна осуществлять шаг, при выходе линии проекции за пределы сферы.

Десинхронизация конечностей

Скопируйте созданные точки проекции и ограничения перемещения для оставшихся конечностей:

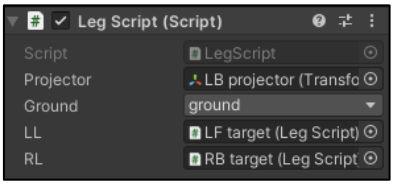


Не забудьте передать в скрипты соответствующие параметры, а объектам задать соответствующие позиции.

Добавьте в скрипт конечности ссылки на смежные конечности:

```
public Transform projector;         // ссылка на точку из которой будет осуществляться проекция
public LayerMask ground;            // слой поверхностей, по которым допустимо перемещение
public LegScript LL;                // ссылка на левую ногу
public LegScript RL;                // ссылка на правую ногу

float stepDistance = 0.7f;          // дистанция шага
float stepSpeed = 8;                // скорость шага
float stepHeight = 1;               // высота шага
float overshoot = 0.8f;             // дополнительно смещение конечности по направлению движения
```

Например, для задней левой конечности, смежными будут являться левая передняя и правая задняя:



Добавьте метод проверки, осуществляет ли конечность перемещение, и модифицируйте условие перемещения следующим образом:

```
Ссылок: 2
public bool isMooving() // если конечность перемещается, вернёт true
{
    return lerp < 1;
}

Ссылок: 0
void LateUpdate()
{
    transform.position = currentPosition;

    Vector3 proj1 = Vector3.ProjectOnPlane(projector.position, Vector3.up);         // проекция точки построения луча на плоскость X|Z
    Vector3 proj2 = Vector3.ProjectOnPlane(currentPosition, Vector3.up);            // проекция текущей позиции конечности на плоскость X|Z
    // если дистанция между старой и новой позициями больше чем дистанция шага, если конечность не перемещается
    // если смежные конечности тоже не перемещаются
    if (Vector3.Distance(proj1, proj2) > stepDistance && lerp >= 1 && LL.isMooving() == false && RL.isMooving() == false)
    {
        Physics.Raycast(projector.position + (proj1 - proj2) * overshoot, Vector3.down, out hit, 100, ground);

        newPosition = hit.point;
        lerp = 0;
    }

    if (lerp < 1)...
    else...

}
```

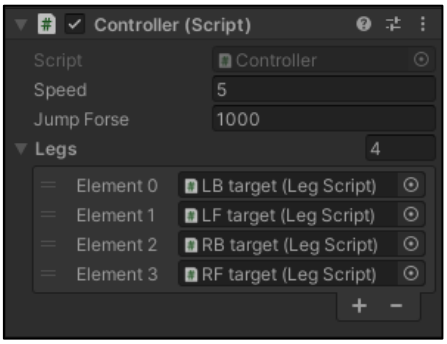
Если всё было сделано правильно, то теперь будут перемещаться не более двух конечностей за раз.

Выравнивание наклона модели

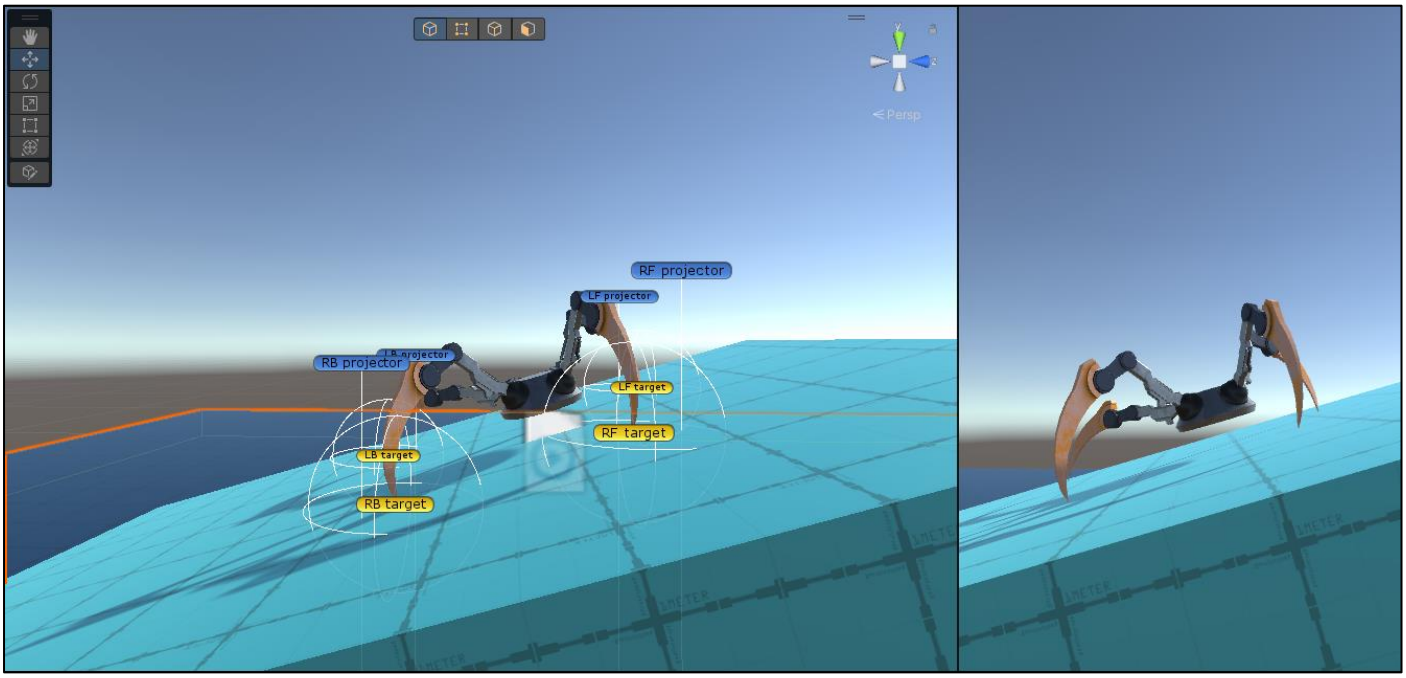
Чтобы корректно рассчитать наклон модели при её перемещении по наклонным поверхностям, необходимо перемножить вектора, простроенные из позиций противолежащих ног и использовать полученный результат в качестве нормали модели. Для этого модифицируйте скрипт управления моделью:

```
public float speed = 4;         // скорость перемещения
public float jumpForse = 500;   // сила прыжка

public LegScript[] legs;        // ссылки на ноги

Rigidbody rb;

Ссылок: 0
void Start()...

Ссылок: 0
void LateUpdate()
{
    float v = Input.GetAxisRaw("Vertical");                     // получение направления движения
    float h = Input.GetAxisRaw("Horizontal");                   // по нажатым клавишам

    Vector3 dir = new Vector3(h, 0, v);
    dir = transform.TransformDirection(dir.normalized);         // применение поворота объекта к вектору направления движения
    dir *= speed;                                               // применение скорости к вектору направления движения
    dir.y = rb.velocity.y;                                      // восстановление смещения по оси Y
    rb.velocity = dir;

    if (Input.GetKeyDown(KeyCode.Space))...

    if (Input.GetKey(KeyCode.Q))...

    if (Input.GetKey(KeyCode.E))...

    Vector3 v1 = legs[0].transform.position - legs[3].transform.position;
    Vector3 v2 = legs[1].transform.position - legs[2].transform.position;
    Vector3 normal = Vector3.Cross(v1, v2).normalized;                      // вычисление нормали к векторам, соединяющим протеволежащие ноги

    // поворот объекта к вычисленной нормали
    transform.rotation = Quaternion.Slerp(transform.rotation, (Quaternion.FromToRotation(transform.up, normal) * transform.rotation), Mathf.PI / 18);
}
```

В приведённой реализации, порядок перечисления конечностей должен соответствовать приведённому ниже:



Результат работы:



Задание:

Разработать и реализовать приложение, позволяющее управлять одной из моделей, прилагаемых к лабораторной работе (папка "Модели").

Модель должна иметь возможность перемещаться в плоскости X|Z и вращаться вокруг оси Y.

В процессе перемещения и вращения модели, должна рассчитываться и воспроизводиться процедурная анимация перемещения конечностей.

Запись трансляции с выполнением лабораторной работы можно найти по ссылке: https://eios.sibsutis.ru/mod/url/view.php?id=132013