УПРАВЛЕНИЕ ПЕРСОНАЖЕМ МЫШЬЮ И КЛАВИАТУРОЙ

MouseLook

```
using System.Collections;

public class MouseLook : MonoBehaviour {
    public enum RotationAxes {
        MouseXAndY = 0,
        MouseX = 1,
        MouseY = 2
    }
    public RotationAxes axes = RotationAxes.MouseXAndY;


    void Update() {
        if (axes == RotationAxes.MouseX) {
            // это поворот в горизонтальной плоскости
        }
        else if (axes == RotationAxes.MouseY) {
            // это поворот в вертикальной плоскости
        }
        else {
            // это комбинированный поворот
        }
    }
}
```

← Объявляем структуру данных enum, которая будет сопоставлять имена с параметрами.

← Объявляем общедоступную переменную, которая появится в редакторе Unity.

← Сюда поместим код для вращения только по горизонтали. 1

← Сюда поместим код для вращения только по вертикали. 2

← Сюда поместим код для комбинированного вращения. 3

Переключение управления

Панель Inspector отображает сопоставленные перечислениям общедоступные переменные в виде раскрывающихся меню

Вращение по горизонтали

Поворот по горизонтали, пока не связанный с движениями указателя

```
...
public RotationAxes axes = RotationAxes.MouseXAndY; ← Курсивом выделен код, который уже присутствует в сценарии; здесь он приведен для удобства.

public float sensitivityHor = 9.0f; ← Объявляем переменную для скорости вращения.

void Update() {
    if (axes == RotationAxes.MouseX) {
        transform.Rotate(0, sensitivityHor, 0); ← Сюда мы поместим команду Rotate, чтобы она запускалась в каждом кадре.
    }
...
```

Управление мышью по горизонтали

Команда Rotate, реагирующая на движения указателя мыши

```
...
transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
...
```

← Метод GetAxis() получает данные, вводимые с помощью мыши.

Поворот в вертикальной плоскости для сценария MouseLook

```
...
public float sensitivityHor = 9.0f;
public float sensitivityVert = 9.0f; ← Объявляем переменные, задающие поворот в вертикальной плоскости.

public float minimumVert = -45.0f;
public float maximumVert = 45.0f;

private float _rotationX = 0; ← Объявляем закрытую переменную для угла поворота по вертикали.

void Update() {
    if (axes == RotationAxes.MouseX) {
        transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
    }
    else if (axes == RotationAxes.MouseY) {
        _rotationX -= Input.GetAxis("Mouse Y") * sensitivityVert; ← Увеличиваем угол поворота по вертикали в соответствии с перемещениями указателя мыши.

        _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert); ← Фиксируем угол поворота по вертикали в диапазоне, заданном минимальным и максимальным значениями.

        float rotationY = transform.localEulerAngles.y; ← Сохраняем одинаковый угол поворота вокруг оси Y (т. е. вращение в горизонтальной плоскости отсутствует).

        transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0); ← Создаем новый вектор из сохраненных значений поворота.
    }
...
```

Одновременное перемещение взгляда по X и Y

Сценарий MouseLook, поворачивающий объект одновременно по горизонтали и по вертикали

```
...
else {
    _rotationX -= Input.GetAxis("Mouse Y") * sensitivityVert;
    _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert);

    float delta = Input.GetAxis("Mouse X") * sensitivityHor;
    float rotationY = transform.localEulerAngles.y + delta;

    transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
}
...
```

← Приращение угла поворота через значение delta.

← Значение delta – это величина изменения угла поворота.

ОТКЛЮЧАЕМ ВЛИЯНИЕ СРЕДЫ НА ИГРОКА

Хотя для данного проекта это не имеет особого значения, в большинстве современных игр от первого лица на все элементы сцены действует модель физической среды. Она заставляет объекты отскакивать и опрокидываться; такое поведение подходит для большинства объектов, но вращение нашего игрока должно контролироваться исключительно мышью, не попадая под влияние смоделированной физической среды.

По этой причине в сценариях, где присутствует ввод с помощью мыши, к компоненту Rigidbody игрока обычно добавляют свойство freezeRotation. Поместите в сценарий MouseLook вот такой метод Start():

```
...
void Start() {
    Rigidbody body = GetComponent<Rigidbody>();
    if (body != null) ← Проверяем, существует ли этот компонент.
    body.freezeRotation = true;
}
...
```

Rigidbody — это дополнительный компонент, которым может обладать объект. Моделируемая физическая среда влияет на этот компонент и управляет объектами, к которым он присоединен.

FPSInput
перемещаем объект

Код вращения из первого листинга с парой небольших изменений

```
using UnityEngine;
using System.Collections;
public class FPSInput : MonoBehaviour {
  public float speed = 6.0f; ← Необязательный элемент на случай, если вы захотите увеличить скорость.
  void Update() {
    transform.Translate(0, speed, 0); ← Меняем метод Rotate() на метод Translate()
  }
}
```

Реакция на нажатие клавиш

Изменение положения в ответ на нажатие клавиш

```
...
void Update() {
  float deltaX = Input.GetAxis("Horizontal") * speed;
  float deltaZ = Input.GetAxis("Vertical") * speed;
  transform.Translate(deltaX, 0, deltaZ);
}
```

← "Horizontal" и "Vertical" – это дополнительные имена для сопоставления с клавиатурой.

Обратите внимание, что значения перемещения меняются по координатам *X* и *Z*. Вы могли заметить в процессе экспериментов с методом `Translate()`, что изменение координаты *X* соответствует движению из стороны в сторону, в то время как изменение координаты *Z* означает движение вперед и назад.

Вставив в сценарий этот новый код перемещения, вы получите возможность двигать объект по сцене нажатием клавиш со стрелками или клавиш с буквами WASD. Это стандартная ситуация для большинства игр от первого лица. Сценарий, управляющий перемещениями, практически готов, осталось только внести в него некоторые изменения.

Движение с независимостью от кадровой частоты благодаря переменной deltaTime

```
...
void Update() {
  float deltaX = Input.GetAxis("Horizontal") * speed;
  float deltaZ = Input.GetAxis("Vertical") * speed;
  transform.Translate(deltaX * Time.deltaTime, 0, deltaZ * Time.deltaTime);
}
...
```

Перемещение компонента CharacterController вместо компонента Transform

```
...
private CharacterController _charController;
```
← Переменная для ссылки на компонент CharacterController.

```
void Start() {
  _charController = GetComponent<CharacterController>();
}
```
← Доступ к другим компонентам, присоединенным к этому же объекту.

```
void Update() {
  float deltaX = Input.GetAxis("Horizontal") * speed;
  float deltaZ = Input.GetAxis("Vertical") * speed;
  Vector3 movement = new Vector3(deltaX, 0, deltaZ);

  movement = Vector3.ClampMagnitude(movement, speed);
```
← Ограничим движение по диагонали той же скоростью, что и движение параллельно осям.

```
  movement *= Time.deltaTime;
  movement = transform.TransformDirection(movement);
```
← Преобразуем вектор движения от локальных к глобальным координатам.

```
  _charController.Move(movement);
}
...
```
← Заставим этот вектор перемещать компонент CharacterController.

Ходить, а не летать

Теперь, когда распознавание столкновений работает, в сценарий можно добавить силу тяжести, чтобы игрок стоял на полу. Объявим переменную `gravity` и воспользуемся ее значением для оси *Y*, как показано в следующем листинге.

Добавление силы тяжести в код движения

```
...
public float gravity = -9.8f;
...
void Update() {
  ...
  movement = Vector3.ClampMagnitude(movement, speed);
  movement.y = gravity; ← Используем значение переменной gravity вместо нуля.
  ...
```

СОВЕРШЕНСТВОВАНИЕ ГОТОВОГО СЦЕНАРИЯ

```
using UnityEngine;
using System.Collections;
[RequireComponent(typeof(CharacterController))]
[AddComponentMenu("Control Script/FPS Input")]
public class FPSInput : MonoBehaviour {
```

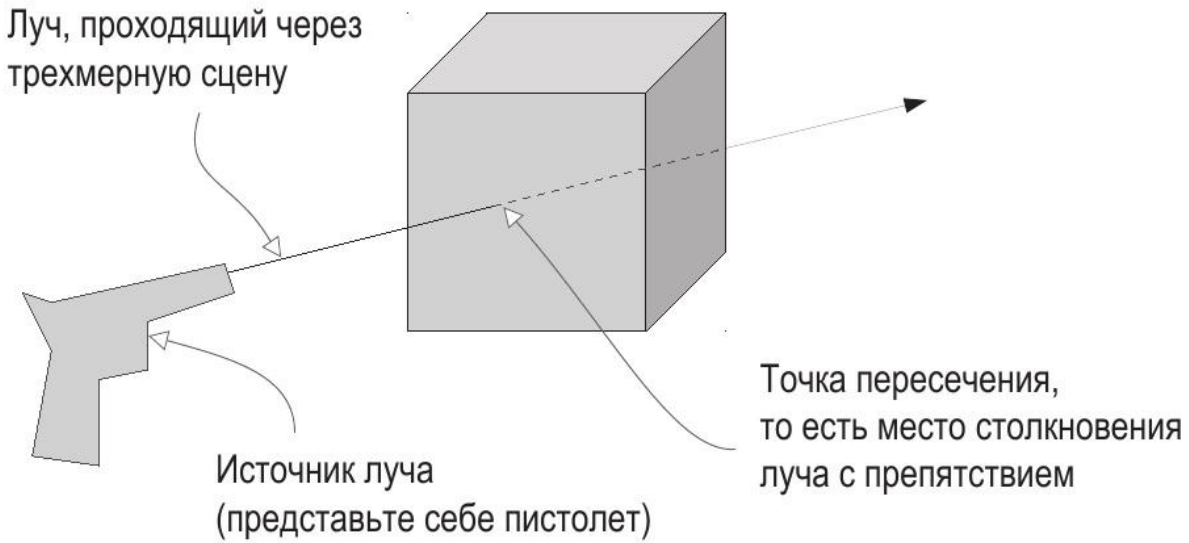
Готовый сценарий FPSInput

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(CharacterController))]
[AddComponentMenu("Control Script/FPS Input")]
public class FPSInput : MonoBehaviour {
  public float speed = 6.0f;
  public float gravity = -9.8f;

  private CharacterController _charController;

  void Start() {
    _charController = GetComponent<CharacterController>();
  }

  void Update() {
    float deltaX = Input.GetAxis("Horizontal") * speed;
    float deltaZ = Input.GetAxis("Vertical") * speed;
    Vector3 movement = new Vector3(deltaX, 0, deltaZ);
    movement = Vector3.ClampMagnitude(movement, speed);

    movement.y = gravity;

    movement *= Time.deltaTime;
    movement = transform.TransformDirection(movement);
    _charController.Move(movement);
  }
}
```

Добавляем в игру врагов и снаряды

- ✓ Как игрок и его враги получают возможность целиться и стрелять
- ✓ Попадания и реакция на них
- ✓ Заставляем врагов перемещаться
- ✓ Порождение новых объектов сцены

1. Написать код, позволяющий игроку стрелять.
2. Создать статичные цели, реагирующие на попадание.
3. Заставить цели перемещаться по сцене.
4. Вызвать автоматическое появление новых блуждающих целей.
5. Дать возможность целям/врагам кидать в игрока огненные шары.



Луч представляет собой воображаемую линию, и при бросании луча выясняется, с чем этот луч пересекается

Сценарий RayShooter, присоединяемый к камере

```
using UnityEngine;
using System.Collections;

public class RayShooter : MonoBehaviour {
  private Camera _camera;

  void Start() {
    _camera = GetComponent<Camera>(); ← Доступ к другим компонентам, присоединенным к этому же объекту.
  }

  void Update() {
    if (Input.GetMouseButtonDown(0)) { ← Реакция на нажатие кнопки мыши.
      Vector3 point = new Vector3( ← Середина экрана – это половина его ширины и высоты.
        _camera.pixelWidth/2, _camera.pixelHeight/2, 0);
      Ray ray = _camera.ScreenPointToRay(point); ← Создание в этой точке луча методом ScreenPointToRay().
      RaycastHit hit;
      if (Physics.Raycast(ray, out hit)) { ← Испущенный луч заполняет информацией переменную, на которую имеется ссылка.
        Debug.Log("Hit " + hit.point); ← Загружаем координаты точки, в которую попал луч.
      }
    }
  }
}
```

Сценарий RayShooter после добавления индикаторных сфер

```
using UnityEngine;
using System.Collections;

public class RayShooter : MonoBehaviour {
  private Camera _camera;

  void Start() {
    _camera = GetComponent<Camera>();
  }

  void Update() { ← Эта функция по большей части содержит знакомый нам код бросания луча из листинга 3.1.
    if (Input.GetMouseButtonDown(0)) {
      Vector3 point = new Vector3(
        _camera.pixelWidth/2, _camera.pixelHeight/2, 0);
      Ray ray = _camera.ScreenPointToRay(point);
      RaycastHit hit;
      if (Physics.Raycast(ray, out hit)) {
        StartCoroutine(SphereIndicator(hit.point)); ← Запуск сопрограммы в ответ на попадание.
      }
    }
  }

  private IEnumerator SphereIndicator(Vector3 pos) { ← Сопрограммы пользуются функциями IEnumerator.
    GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    sphere.transform.position = pos;

    yield return new WaitForSeconds(1); ← Ключевое слово yield указывает сопрограмме,
                                          когда следует остановиться.

    Destroy(sphere); ← Удаляем этот GameObject и очищаем память.
  }
}
```

Визуальный индикатор для точки прицеливания

```
...
void Start() {
  _camera = GetComponent<Camera>();

  Cursor.lockState = CursorLockMode.Locked;
  Cursor.visible = false;
}

void OnGUI() {
  int size = 12;
  float posX = _camera.pixelWidth/2 - size/4;
  float posY = _camera.pixelHeight/2 - size/2;
  GUI.Label(new Rect(posX, posY, size, size), "*");
}
...
```

Скрываем указатель мыши в центре экрана.

← Команда GUI.Label() отображает на экране символ.

Распознавание попаданий в цель

```
...
if (Physics.Raycast(ray, out hit)) {
  GameObject hitObject = hit.transform.gameObject; ← Получаем объект, в который попал луч.
  ReactiveTarget target = hitObject.GetComponent<ReactiveTarget>();
  if (target != null) { ← Проверяем наличие у этого объекта компонента ReactiveTarget.
    Debug.Log("Target hit");


    } else {
      StartCoroutine(SphereIndicator(hit.point));
    }
  }
  ...
```

Отправка сообщения целевому объекту

```
...
if (target != null) {
  target.ReactToHit(); ← Вызов метода для мишени вместо генерации отладочного сообщения.
} else {
  StartCoroutine(SphereIndicator(hit.point));
}
...
```

Сценарий ReactiveTarget, реализующий смерть врага при попадании

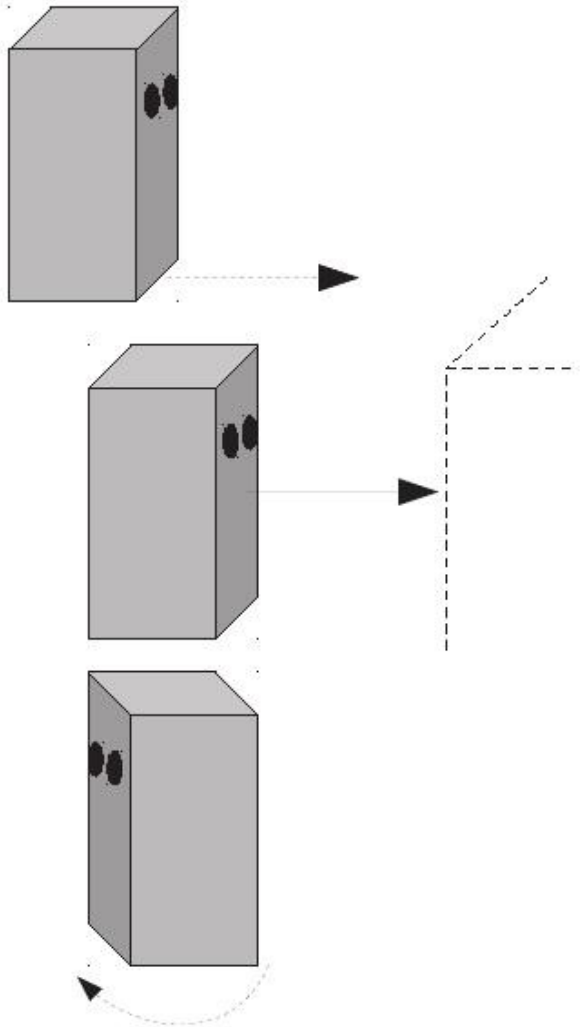
```
using UnityEngine;
using System.Collections;

public class ReactiveTarget : MonoBehaviour {

  public void ReactToHit() {  ← Метод, вызванный сценарием стрельбы.
    StartCoroutine(Die());
  }

  private IEnumerator Die() {  ← Опрокидываем врага, ждем 1,5 секунды и уничтожаем его.
    this.transform.Rotate(-75, 0, 0);

    yield return new WaitForSeconds(1.5f);

    Destroy(this.gameObject);  ← Объект может уничтожать сам себя точно так же, как любой другой объект.
  }
}
```

Шаг 1:
Небольшое смещение
вперед

Шаг 2:
Бросание луча
вперед для обнаружения
препятствий

Шаг 3:
Отход от препятствия

Шаг 4:
Визуализация кадра,
возвращение к шагу 1



Базовый искусственный интеллект: циклический процесс движения вперед с обходом препятствий

Окончательный вид сценария MouseLook

```
using UnityEngine;
using System.Collections;

public class MouseLook : MonoBehaviour {
  public enum RotationAxes {
    MouseXAndY = 0,
    MouseX = 1,
    MouseY = 2
  }
  public RotationAxes axes = RotationAxes.MouseXAndY;

  public float sensitivityHor = 9.0f;
  public float sensitivityVert = 9.0f;

  public float minimumVert = -45.0f;
  public float maximumVert = 45.0f;

  private float _rotationX = 0;

  void Start() {
    Rigidbody body = GetComponent<Rigidbody>();
    if (body != null)
    body.freezeRotation = true;
  }

  void Update() {
    if (axes == RotationAxes.MouseX) {
      transform.Rotate(0, Input.GetAxis("Mouse X") * sensitivityHor, 0);
    }
    else if (axes == RotationAxes.MouseY) {
      _rotationX -= Input.GetAxis("Mouse Y") * sensitivityVert;
      _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert);

      float rotationY = transform.localEulerAngles.y;

      transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
    }
    else {
      _rotationX -= Input.GetAxis("Mouse Y") * sensitivityVert;
      _rotationX = Mathf.Clamp(_rotationX, minimumVert, maximumVert);

      float delta = Input.GetAxis("Mouse X") * sensitivityHor;
      float rotationY = transform.localEulerAngles.y + delta;

      transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
    }
  }
}
```