🤖

NPC для городского окружения.

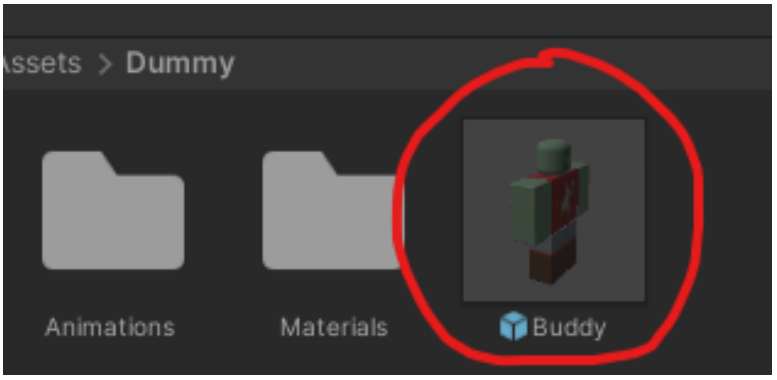
Подготовлено для хакатона “DECENTRATHON 2.0”. Кейс - GAMEDEV.

📜 Предисловие

- Движок: Выбор пал на движок Unity, т.к. он подходит для мобильных устройств и ПК.
- ИИ: Был написан на основе технологии “Raycast”.
- Архитектура: Для создания легкого в понимании и интеграции решения, был выбран паттерн “Final State Machine” или “Конечная Машина Состояний”.

🔥 GAME READY | Готовое к игре!

Скачайте Ассет из репозитория, или из релизов. Перетащите префаб объекта на сцену, укажите все необходимые слои и запускайте игру!

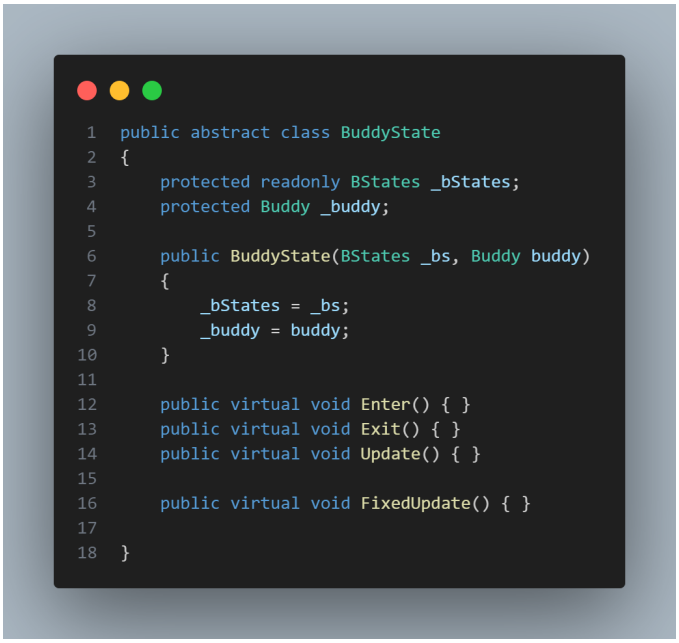


Префаб находиться в папке **“Dummy”.**

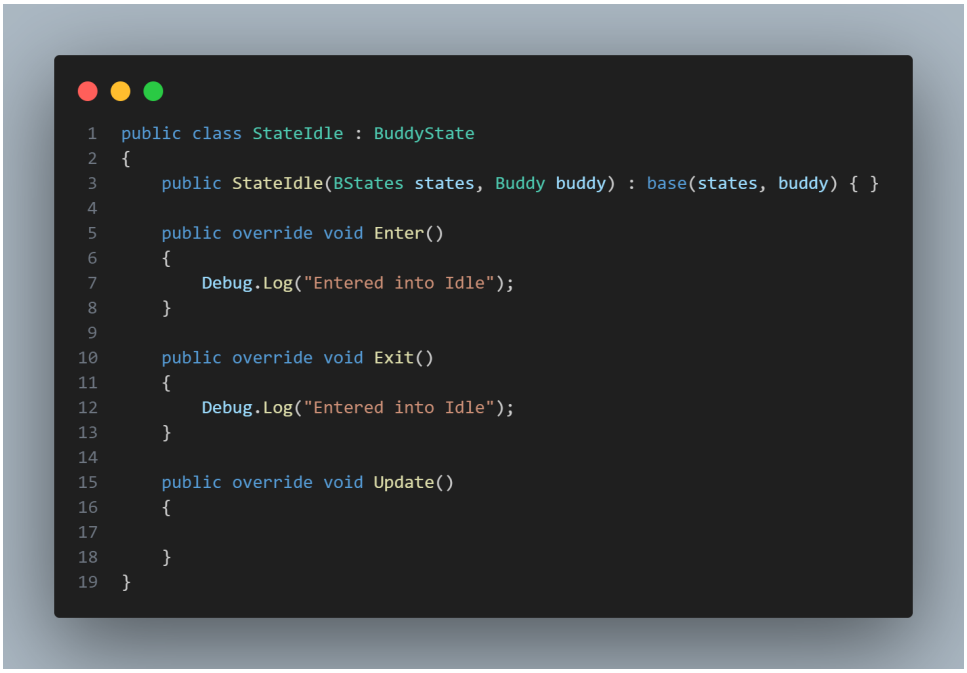
🦾 Основа NPC

Основа NPC была создана на основе паттерна “State Machine”, для удобной интеграции, расширения и пользования.

Основной скрипт машины состояний “BStates”. В нем находиться словарь с состояниями, а так же методы для работы с состояниями.



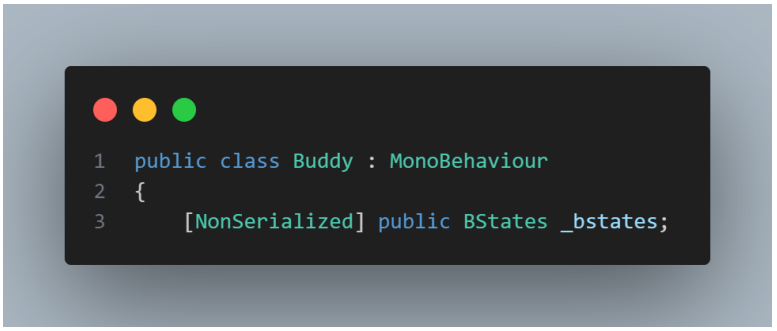
Абстрактный класс BuddyState, с конструктором для состояний, а так же виртуальными методами которые позволяют гибко работать с состояниями.



```
public class StateIdle : BuddyState
{
    public StateIdle(BStates states, Buddy buddy) : base(states, buddy) { }

    public override void Enter()
    {
        Debug.Log("Entered into Idle");
    }

    public override void Exit()
    {
        Debug.Log("Entered into Idle");
    }

    public override void Update()
    {

    }
}
```

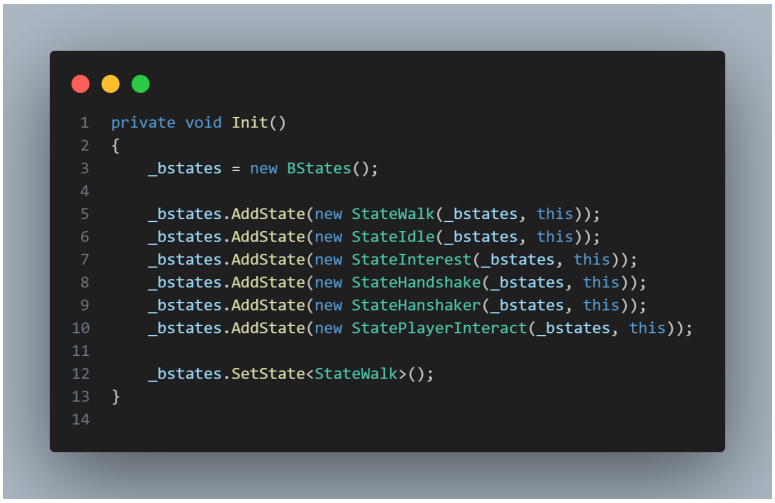
Рассмотрим пример с состоянием “Idle”, Реализован метод входа, и выхода, а так же метод Update для просчета функций каждый кадр.

✅ Реализация NPC



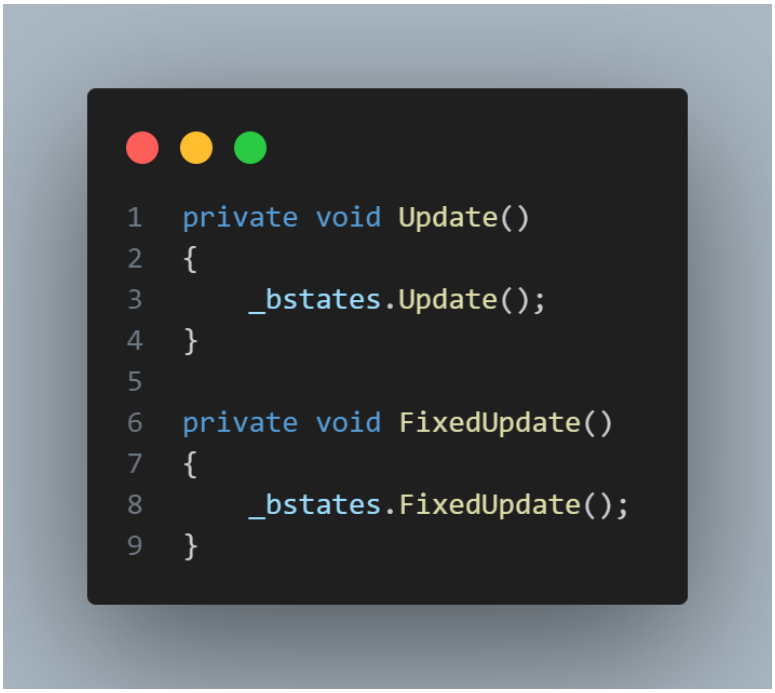
```
public class Buddy : MonoBehaviour
{
    [NonSerialized] public BStates _bstates;
```

Скрипт “Buddy” является нашим NPC. Наследуется от MonoBehaviour. Содержит ссылку на главный скрипт состояний для управления ими.



```
private void Init()
{
    _bstates = new BStates();

    _bstates.AddState(new StateWalk(_bstates, this));
    _bstates.AddState(new StateIdle(_bstates, this));
    _bstates.AddState(new StateInterest(_bstates, this));
    _bstates.AddState(new StateHandshake(_bstates, this));
    _bstates.AddState(new StateHanshaker(_bstates, this));
    _bstates.AddState(new StatePlayerInteract(_bstates, this));

    _bstates.SetState<StateWalk>();
}
```

Функция **Init** вызываемая только в **Start** для инициализации состояний и их добавление, так же содержит точку входа в начальное состояние → **StateWalk**.



```
private void Update()
{
    _bstates.Update();
}

private void FixedUpdate()
{
    _bstates.FixedUpdate();
}
```

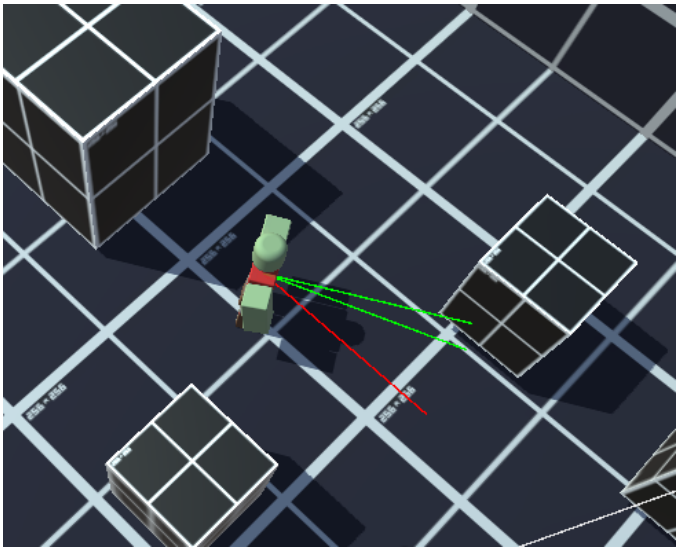
Вызов функций для обработки состояний, Для функции Update соответственно. **Обработка действий состояний происходит в реализованном методе самого состояния!**

👀 Зрение NPC

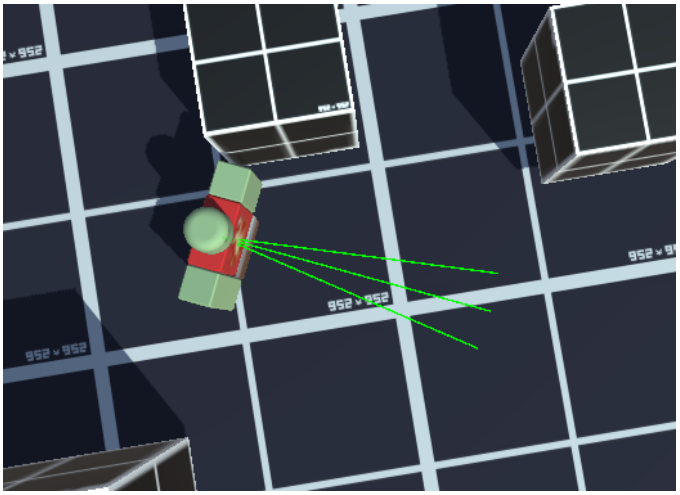
Зрение у NPC есть благодаря технологии Raycast. Сканируя перед собой объекты обходит их.

```
private void ScanAround()
    {
        RayCount = 0;
        for (int i = 0; i < 3; i++)
        {
            RayCount += 45 * Mathf.Deg2Rad / 3;

            Vector3 dir = transform.TransformDirection(new Vector3(Mathf.Sin(RayCount) - offsetRay.x,
                0, Mathf.Cos(RayCount) - offsetRay.z)) + transform.forward;

            _rayVision = new Ray(transform.position, dir);

            if (Physics.Raycast(_rayVision, out _hitVision, VisionDistance))
            {
                Vector3 newDirection = Vector3.Cross(_hitVision.normal, Vector3.up).normalized;

                Direction = Vector3.Lerp(Direction, newDirection, Time.deltaTime * 0.7f);

                Debug.DrawRay(_rayVision.origin, newDirection * VisionDistance, UnityEngine.Color.red);
            }
            else
            {

                Debug.DrawRay(_rayVision.origin, _rayVision.direction * VisionDistance, UnityEngine.Color.green);
            }
        }
    }
```

Функция **ScanAround** вызываемая в методе **Update** пускает 3 луча в направлении перед собой. Дальность лучей настраивается в Инспекторе.



Луч касающийся любого объекта перед собой меняет направление благодаря функции **Vector3.Cross** относительно нормали объекта которого касается. Разворот происходит с помощью смены **Direction** интерполяцией вектора.

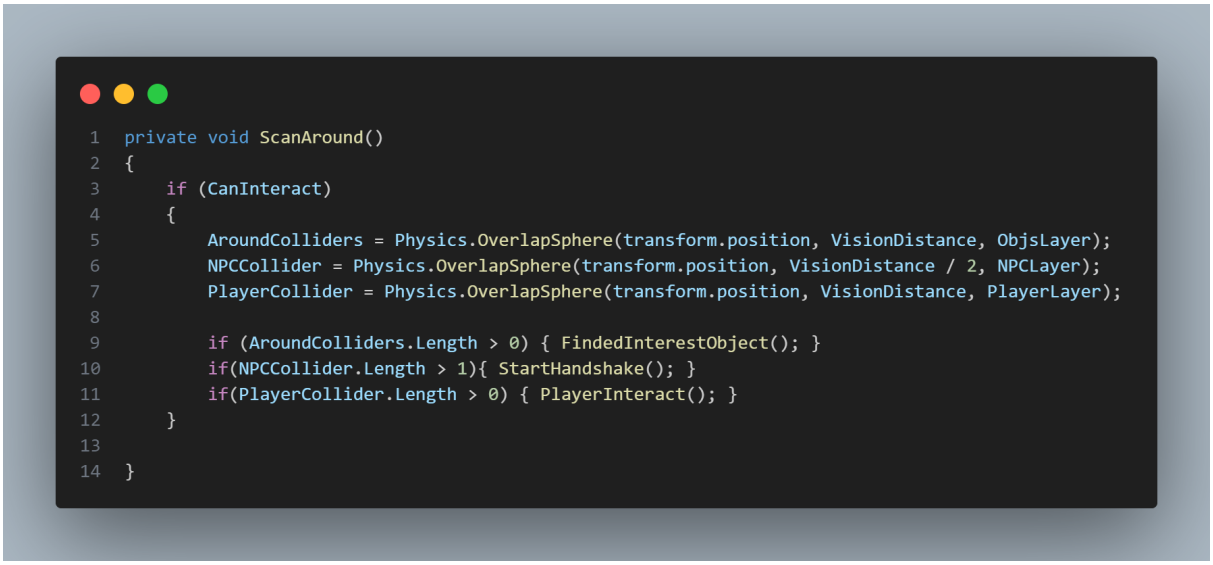


👁️ Дополнительное зрение вокруг

У NPC присутствует доп. зрение для обнаружения объектов и работы с ними.

Реализовано с помощью **OverlapSphere,** на настраиваемых **Слоях(Layers)** для экономии ресурсов.

Просчет объектов для взаимодействия с ними просчитывается когда значение **CanInteract == true**. При обнаружении объектов, **NPC** переходит в другое состояние.



```
private void ScanAround()
{
    if (CanInteract)
    {
        AroundColliders = Physics.OverlapSphere(transform.position, VisionDistance, ObjsLayer);
        NPCCollider = Physics.OverlapSphere(transform.position, VisionDistance / 2, NPCLayer);
        PlayerCollider = Physics.OverlapSphere(transform.position, VisionDistance, PlayerLayer);

        if (AroundColliders.Length > 0) { FindedInterestObject(); }
        if(NPCCollider.Length > 1){ StartHandshake(); }
        if(PlayerCollider.Length > 0) { PlayerInteract(); }
    }

}
```

🤝 Взаимодействие NPC

Для взаимодействия с окружением, игроком, и локацией существуют несколько состояний (Можно добавлять сколько угодно).

⚠ ВАЖНО | Перезарядка

У NPC существует перезарядка на любое взаимодействие. Время перезарядки настраиваемое.

📄 StateInterest.cs | Взаимодействие с окружением

▼ 🟦 Содержание

```
public class StateInterest : BuddyState
{
    private Collider Obj;
    private float time_stay = 0;

    public StateInterest(BStates states, Buddy buddy) : ba

    public override void Enter()
    {
```

```
        Debug.Log("Entered to exploring state");

        Obj = _buddy.AroundColliders[0];

        Vector3 direction = (Obj.transform.position - _bud
        direction.y = 0;
        _buddy.transform.rotation = Quaternion.LookRotatio

        _buddy._animator.Play("Interest");

        Debug.Log($"Interest object is {Obj.name}");
    }

    public override void Exit()
    {
        Debug.Log("Exited to exploring state");
    }
    public override void Update()
    {
        if (time_stay > 3.5) {
            time_stay = 0;
            _bStates.SetState<StateWalk>();
        }
        else
        {
            time_stay += Time.deltaTime;
        }
    }

}
```

Является состоянием при котором NPC находится в шоке от красоты предмета, с соответствующей анимацией.

```
    private void FindedInterestObject()
    {
        CanInteract = false;
        InteractReloading = 0;
```

```
        _bstates.SetState<StateInterest>();
    }
```

Вызывается при обнаружении объекта со специальным тэгом. (Тэг настраивается в инспекторе).

📄 StatePlayerInteract.cs | Взаимодействие с игроком

▼ 📁 Содержание

```
public class StatePlayerInteract : BuddyState
{
    private protected GameObject PlayerObj;
    private protected float time_interact;

    public StatePlayerInteract(BStates states, Buddy buddy
    public override void Enter()
    {
        Debug.Log("State Entered to InteractWithPlayer");

        time_interact = 0;

        PlayerObj = _buddy.PlayerCollider[0].gameObject;

        Vector3 direction = (PlayerObj.transform.position
        direction.y = 0;
        _buddy.transform.rotation = Quaternion.LookRotatio

        _buddy._animator.Play("PlayerInteract");
    }
    public override void Update()
    {
        if(time_interact < 1.7f)
        {
            time_interact += Time.deltaTime;
        }
```

```
        else
        {
            _bStates.SetState<StateWalk>();
        }
    }

}
```

Состояние при котором NPC обращается к игроку указывая на него (Ничего интереснее не придумал).

```
    public void PlayerInteract()
    {
        CanInteract = false;
        InteractReloading = 0;

        _bstates.SetState<StatePlayerInteract>();
    }
```

Вызывается при обнаружении игрока по слоям.

📄 StateHanshaker.cs | Взаимодействие между NPC

▼ 📁 Содержание

```
public class StateHanshaker : BuddyState
{
    private protected Buddy HandshakePair;
    private protected float time;

    public StateHanshaker(BStates states, Buddy buddy) : b

    public override void Enter()
    {
        foreach(Collider c in _buddy.NPCCollider)
        {
```

```
            if(c.name != _buddy.name)
            {
                HandshakePair = c.GetComponent<Buddy>();
                Debug.Log($"Founded Handshake Pair - {Hand
                break;
            }
        }

        if (HandshakePair == null) { _bStates.SetState<Sta
        else
        {
            HandshakePair.JoinHandShake();

            Vector3 direction = (HandshakePair.transform.p
            direction.y = 0;
            _buddy.transform.rotation = Quaternion.LookRot

            HandshakePair.transform.rotation = Quaternion.

            time = 0;

            _buddy._animator.Play("handshakeMain");
            HandshakePair._animator.Play("handshakeSecond"
        }
    }

    public override void Exit()
    {

    }

    public override void Update()
    {
        HandshakePair.transform.position = Vector3.Lerp(Ha
            _buddy.transform.position + _buddy.transform.f

        if(time < 2.1f)
        {
```

```
            time += Time.deltaTime;
        }
        else
        {
            _bStates.SetState<StateWalk>();
            HandshakePair._bstates.SetState<StateWalk>();
        }
    }

}
```

Состояние при котором тот кто перешел в состояние двигает к себе NPC которого нашел дополнительным зрением и проигрывает анимации рукопожатий у обоих NPC.

```
private void StartHandshake()
{
    CanInteract = false;
    InteractReloading = 0;

    _bstates.SetState<StateHanshaker>();
}
```

Функция с вызовом.

📄 StateHandshake.cs | Взаимодействие между NPC

▼ 📁 Содержание

```
public class StateHandshake : BuddyState
{
    public StateHandshake(BStates state, Buddy buddy) : ba

    public override void Enter()
    {
        Debug.Log("JoinedToHandshake");
```

```
    }

}
```

Состояние для второго участника рукопожатия, сделан для отключения управления.

```
    public void JoinHandShake()
    {
        CanInteract = false;
        InteractReloading = 0;

        _bstates.SetState<StateHandshake>();
    }
```

Вызов состояния.

🛠️ Создание состояний

Последовательность действий: Создаем скрипт → Наследуемся от BuddyState → Реализуем конструктор → Реализуем нужные методы.

Пример пустого состояния.

```
public class TestState : BuddyState
{
    public TestState(BStates states, Buddy buddy) : base(stat

    public override void Enter() { }
    public override void Exit() { }
    public override void Update() { }
    public override void FixedUpdate() { }

}
```

⚙️ Описание Инспектора

- Необходимые компоненты: Rigidbody, Capsule Collider, Animator.
- **Ground Layer** - используется для препятствий.
- **Objs Layer** - используется для нахождение специальных объектов.
- **NPC Layer** - используется для нахождения других NPC.
- **Player Layer** - используется для нахождения Игрока.
- **Vision distance** - расстояние зрения для обхода препятствий, так же используется для дополнительного зрения.

- **Offset Ray** - Отклонение для настройки взгляда NPC.
- **Walk speed** - скорость движение к направлению задаваемое лучами.
- **Rotate Turn Smooth Time** - скорость поворота.
- **Animator** - аниматор для управления анимациями.
- **Can Interacting** - Возможность взаимодействия в зависимости от перезарядки и времени перезарядки.
- **Interact Reload Time** - Время перезарядки.