



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".



Основа NPC

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

```
1 public class BStates
```

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

```
1 public abstract class BuddyState
2 {
3     protected readonly BStates _bStates;
4     protected Buddy _buddy;
5
6     public BuddyState(BStates _bs, Buddy buddy)
7     {
8         _bStates = _bs;
9         _buddy = buddy;
10    }
11
12    public virtual void Enter() { }
13    public virtual void Exit() { }
14    public virtual void Update() { }
15
16    public virtual void FixedUpdate() { }
17
18 }
```

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

```
1 public class StateIdle : BuddyState
2 {
3     public StateIdle(BStates states, Buddy buddy) : base(states, buddy) { }
4
5     public override void Enter()
6     {
7         Debug.Log("Entered into Idle");
8     }
9
10    public override void Exit()
11    {
12        Debug.Log("Entered into Idle");
13    }
14
15    public override void Update()
16    {
17    }
18 }
19 }
```

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

✓ Реализация NPC

```
1 public class Buddy : MonoBehaviour
2 {
3     [NonSerialized] public BStates _bstates;
```

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

```
1 private void Init()
2 {
3     _bstates = new BStates();
4
5     _bstates.AddState(new StateWalk(_bstates, this));
6     _bstates.AddState(new StateIdle(_bstates, this));
7     _bstates.AddState(new StateInterest(_bstates, this));
8     _bstates.AddState(new StateHandshake(_bstates, this));
9     _bstates.AddState(new StateHandshaker(_bstates, this));
10    _bstates.AddState(new StatePlayerInteract(_bstates, this));
11
12    _bstates.SetState<StateWalk>();
13 }
14
```

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



```
1 private void Update()
2 {
3     _bstates.Update();
4 }
5
6 private void FixedUpdate()
7 {
8     _bstates.FixedUpdate();
9 }
```

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

👁️ Зрение NPC

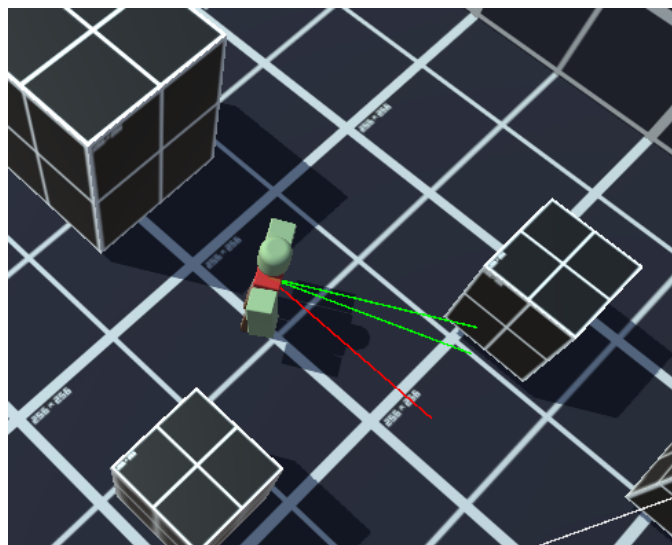
Зрение у NPC есть благодаря технологии Raycast. Сканируя перед собой объекты обходит их.

```

1 private void ScanAround()
2 {
3     RayCount = 0;
4     for (int i = 0; i < 3; i++)
5     {
6         RayCount += 45 * Mathf.Deg2Rad / 3;
7
8         Vector3 dir = transform.TransformDirection(new Vector3(Mathf.Sin(RayCount) - offsetRay.x,
9             0, Mathf.Cos(RayCount) - offsetRay.z)) + transform.forward;
10
11         _rayVision = new Ray(transform.position, dir);
12
13         if (Physics.Raycast(_rayVision, out _hitVision, VisionDistance))
14         {
15             Vector3 newDirection = Vector3.Cross(_hitVision.normal, Vector3.up).normalized;
16
17             Direction = Vector3.Lerp(Direction, newDirection, Time.deltaTime * 0.7f);
18
19             Debug.DrawRay(_rayVision.origin, newDirection * VisionDistance, UnityEngine.Color.red);
20         }
21         else
22         {
23
24             Debug.DrawRay(_rayVision.origin, _rayVision.direction * VisionDistance, UnityEngine.Color.green);
25         }
26     }
27 }

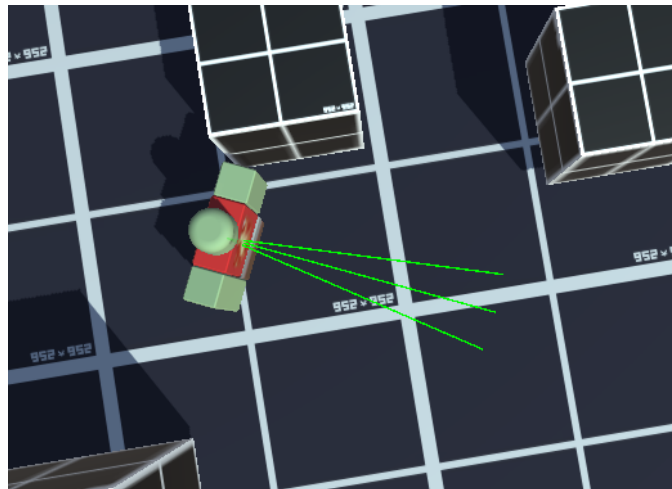
```

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.

```

1 private void ScanAround()
2 {
3     if (CanInteract)
4     {
5         AroundColliders = Physics.OverlapSphere(transform.position, VisionDistance, ObjsLayer);
6         NPCCollider = Physics.OverlapSphere(transform.position, VisionDistance / 2, NPCLayer);
7         PlayerCollider = Physics.OverlapSphere(transform.position, VisionDistance, PlayerLayer);
8
9         if (AroundColliders.Length > 0) { FindedInterestObject(); }
10        if(NPCCollider.Length > 1){ StartHandshake(); }
11        if(PlayerCollider.Length > 0) { PlayerInteract(); }
12    }
13
14 }

```

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.LookRotation(direction)  
  
        _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 - {HandshakePair.name}");
            break;
        }
    }

    if (HandshakePair == null) { _bStates.SetState<State>(State.Waiting); }
    else
    {
        HandshakePair.JoinHandShake();

        Vector3 direction = (HandshakePair.transform.position - transform.position).normalized;
        direction.y = 0;
        _buddy.transform.rotation = Quaternion.LookRotation(direction);

        HandshakePair.transform.rotation = Quaternion.LookRotation(direction);

        time = 0;

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

public override void Exit()
{
}

public override void Update()
{
    HandshakePair.transform.position = Vector3.Lerp(HandshakePair.transform.position,
        _buddy.transform.position + _buddy.transform.forward * 1.5f, Time.deltaTime * 5);

    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(states, buddy)
    {
    }

    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 нахождения 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.