**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**Дисциплина:** Шаблоны проектирования

**Тема:** Принятие решений AI персонажем

**Выполнил: студент группы** *231-338*

*Богослов Илья Максимович*

(Фамилия И.О.)

**Дата, подпись** \_\_\_*22.05.2025*\_\_\_  ***\_\_\_*** ***\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Проверил: \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_***

(Фамилия И.О., степень, звание) **(Оценка)**

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ***\_\_\_\_\_\_\_\_\_\_\_***

(Дата) (Подпись)

**Замечания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва**

**2025**

**Цель**: Реализуйте простую систему принятия решений для NPC (персонажа, управляемого компьютером) с использованием шаблона Strategy.

**Описание**: В играх NPC часто требуется динамически реагировать на различные ситуации, меняя свое поведение в зависимости от контекста. Шаблон Strategy позволяет инкапсулировать различные алгоритмы или стратегии поведения, что делает систему AI более гибкой и легко модифицируемой.

**Шаги**:

1. **Определение стратегий**:
   * Определите несколько различных стратегий поведения для вашего NPC. Например, "блуждание", "преследование игрока", "уклонение от опасности" и так далее.
2. **Реализация шаблона Strategy**:
   * Создайте абстрактный класс или интерфейс для стратегии, который определяет общий метод действия.
   * Для каждой стратегии создайте конкретный класс, реализующий ваш интерфейс или абстрактный класс стратегии. Каждый класс должен содержать логику, специфичную для этой стратегии поведения.
3. **Интеграция с NPC**:
   * Интегрируйте вашу систему стратегий с NPC, позволяя ему выбирать и менять стратегии в зависимости от игровой ситуации.
4. **Тестирование**:
   * Запустите вашу игру и наблюдайте, как NPC меняет свое поведение в различных ситуациях, основываясь на выбранной стратегии.

**Ход работы:**

Для NPC были взяты 3 стратегии поведения: блуждание, преследование игрока и избегание опасности.

Каждая стратегия включает в себя действие, происходящие за один тик нахождения в этой стратегии, действие при выборе этой стратегии и действие при отказе от этой стратегии.

В соответствии с этим был создан интерфейс для стратегий:

public interface INpcStrategy  
{  
 void Tick(NpcContext ctx);  
   
 void Enter(NpcContext ctx) { }  
   
 void Exit(NpcContext ctx) { }  
}

Контекст, доступный стратегиям, т.е. те объекты, что NPC берёт в расчёт при выборе стратегии:

using UnityEngine;  
using UnityEngine.AI;  
  
public sealed class NpcContext  
{  
 public readonly Transform Self;  
 public readonly Transform Player;  
 public readonly NavMeshAgent Agent;  
  
 public NpcContext(Transform self, Transform player, NavMeshAgent agent)  
 {  
 Self = self;  
 Player = player;  
 Agent = agent;  
 }  
}

И конкретные реализации стратегий.

Блуждание:

using UnityEngine;  
using UnityEngine.AI;  
  
public sealed class WanderStrategy : INpcStrategy  
{  
 private readonly float \_radius;  
 private readonly float \_idleTime;  
 private float \_timer;  
 private Vector3 \_target;  
  
 public WanderStrategy(float radius = 10f, float idleTime = 2f)  
 {  
 \_radius = radius;  
 \_idleTime = idleTime;  
 }  
  
 public void Enter(NpcContext ctx)  
 {  
 PickNewPoint(ctx);  
 }  
  
 public void Tick(NpcContext ctx)  
 {  
 \_timer += Time.deltaTime;  
 if (\_timer >= \_idleTime && !ctx.Agent.pathPending && ctx.Agent.remainingDistance < 0.3f)  
 PickNewPoint(ctx);  
 }  
  
 private void PickNewPoint(NpcContext ctx)  
 {  
 \_timer = 0f;  
 var random = Random.insideUnitSphere \* \_radius + ctx.Self.position;  
 if (!NavMesh.SamplePosition(random, out var hit, \_radius, NavMesh.**AllAreas**))  
 return;  
 \_target = hit.position;  
 ctx.Agent.SetDestination(\_target);  
 }  
}

Преследование:

public sealed class ChasePlayerStrategy : INpcStrategy  
{  
 private readonly float \_stopDistance;  
  
 public ChasePlayerStrategy(float stopDistance = 1.5f)  
 {  
 \_stopDistance = stopDistance;  
 }  
  
 public void Tick(NpcContext ctx)  
 {  
 if (!ctx.Player) return;  
  
 ctx.Agent.SetDestination(ctx.Player.position);  
  
 if (ctx.Agent.remainingDistance <= \_stopDistance && !ctx.Agent.pathPending)  
 ctx.Agent.isStopped = true;  
 else  
 ctx.Agent.isStopped = false;  
 }  
}

Избегание опасности:

using UnityEngine;  
  
public sealed class AvoidDangerStrategy : INpcStrategy  
{  
 private readonly Transform \_danger;  
 private readonly float \_fleeDistance;  
  
 public AvoidDangerStrategy(Transform danger, float fleeDistance = 6f)  
 {  
 \_danger = danger;  
 \_fleeDistance = fleeDistance;  
 }  
  
 public void Tick(NpcContext ctx)  
 {  
 var dir = (ctx.Self.position - \_danger.position).normalized;  
 var point = ctx.Self.position + dir \* \_fleeDistance;  
  
 if (  
 UnityEngine.AI.NavMesh.SamplePosition(  
 point,  
 out var hit,  
 \_fleeDistance,  
 UnityEngine.AI.NavMesh.**AllAreas** )  
 )  
 ctx.Agent.SetDestination(hit.position);  
 }  
}

Мозг NPC, в котором мы определили логику выбора им стратегии поведения и задали, что мы определяем, как игрока, и что, как опасность:

using UnityEngine;  
using UnityEngine.AI;  
  
[RequireComponent(typeof(NavMeshAgent))]  
public sealed class NpcBrain : MonoBehaviour  
{  
 [Header("References")]  
 [SerializeField]  
 private Transform player;  
  
 [SerializeField]  
 private Transform danger;  
  
 [Header("Distances")]  
 public float chaseRadius = 12f;  
 public float dangerRadius = 6f;  
  
 private NavMeshAgent \_agent;  
 private NpcContext \_ctx;  
  
 private INpcStrategy \_current;  
 private readonly WanderStrategy \_wander = new();  
 private ChasePlayerStrategy \_chase;  
 private AvoidDangerStrategy \_avoid;  
  
 private void Awake()  
 {  
 \_agent = GetComponent<NavMeshAgent>();  
 \_ctx = new NpcContext(transform, player, \_agent);  
  
 \_chase = new ChasePlayerStrategy();  
 if (danger)  
 \_avoid = new AvoidDangerStrategy(danger);  
 }  
  
 private void OnEnable() => SwitchTo(\_wander);  
  
 private void Update()  
 {  
 PickStrategy();  
 \_current.Tick(\_ctx);  
 }  
  
 private void PickStrategy()  
 {  
 if (danger && Vector3.Distance(transform.position, danger.position) < dangerRadius)  
 {  
 if (\_current != \_avoid)  
 SwitchTo(\_avoid);  
 return;  
 }  
  
 if (player && Vector3.Distance(transform.position, player.position) < chaseRadius)  
 {  
 if (\_current != \_chase)  
 SwitchTo(\_chase);  
 return;  
 }  
  
 if (\_current != \_wander)  
 SwitchTo(\_wander);  
 }  
  
 private void SwitchTo(INpcStrategy next)  
 {  
 \_current?.Exit(\_ctx);  
 \_current = next;  
 \_current?.Enter(\_ctx);  
 }  
}

На первом скриншоте видно, как NPC (синий) незаинтересованно блуждает и его направление движения хаотично:

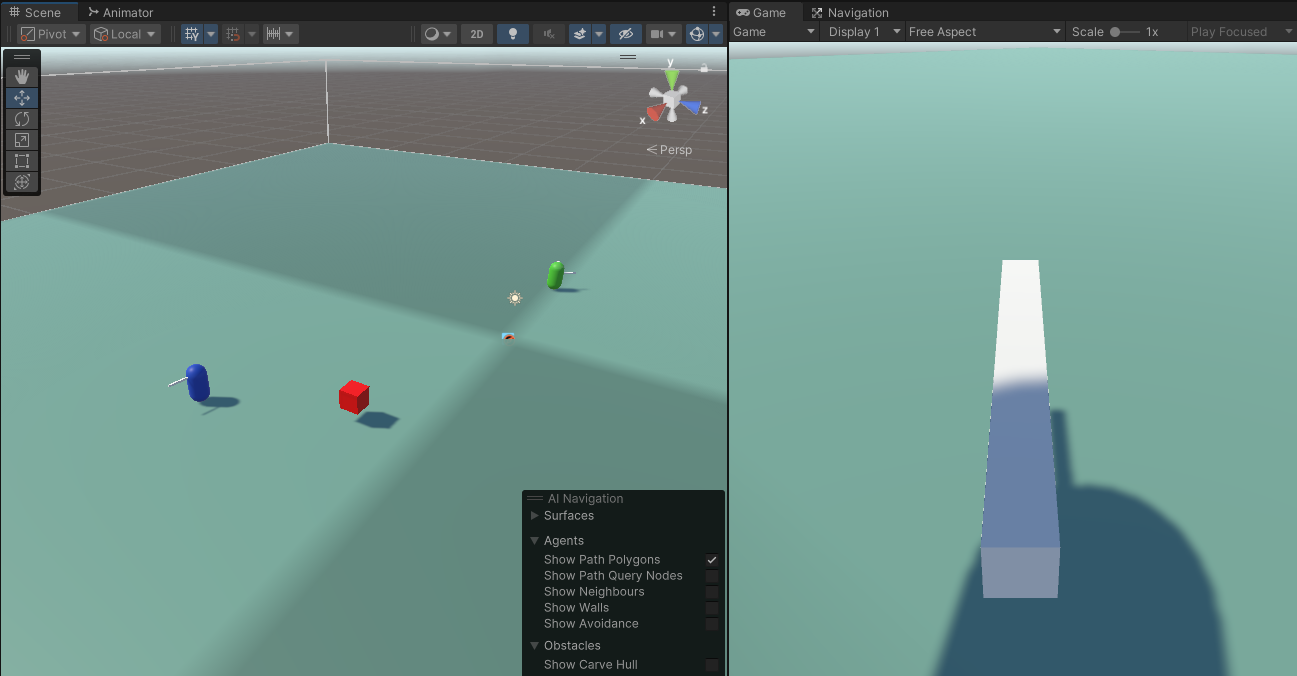


Рисунок 1 – Блуждание

При приближении к NPC на указанное в параметрах расстояние, в моём случае 12 юнитов, NPC начинает следовать по направлению к игроку:

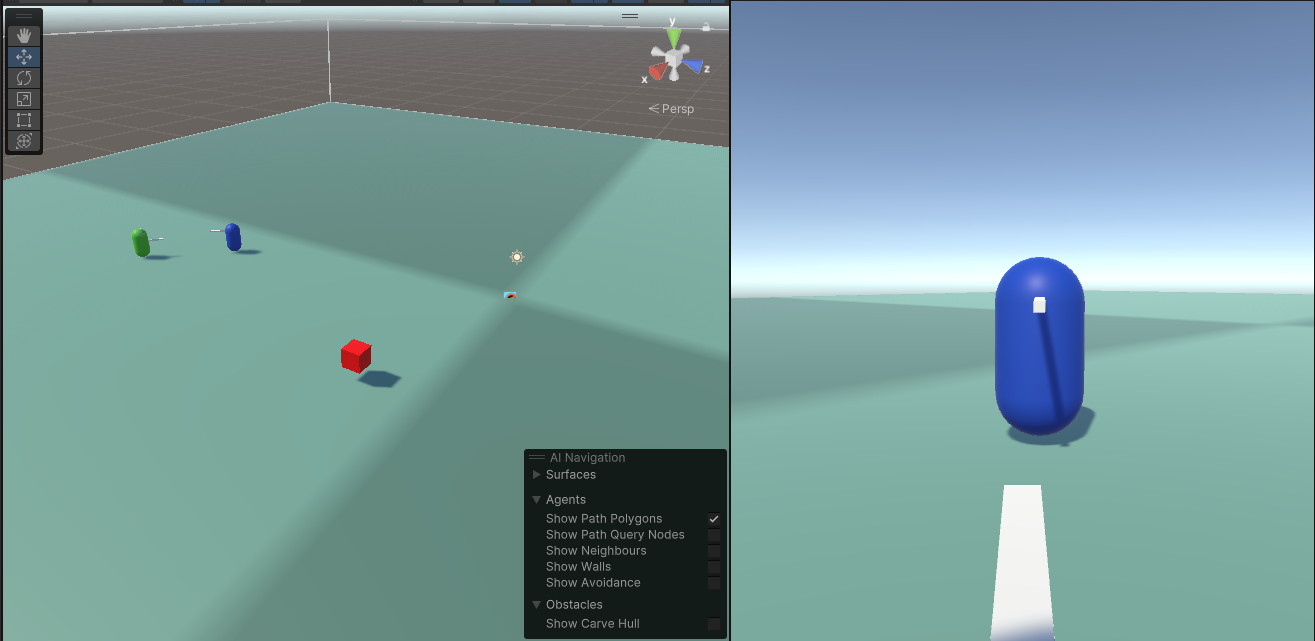


Рисунок 2 - Преследование

При возникновении рядом с NPC опасности, он пытается поменять направление своего движения и избежать захождения в его зону действия - 6 юнитов вокруг:

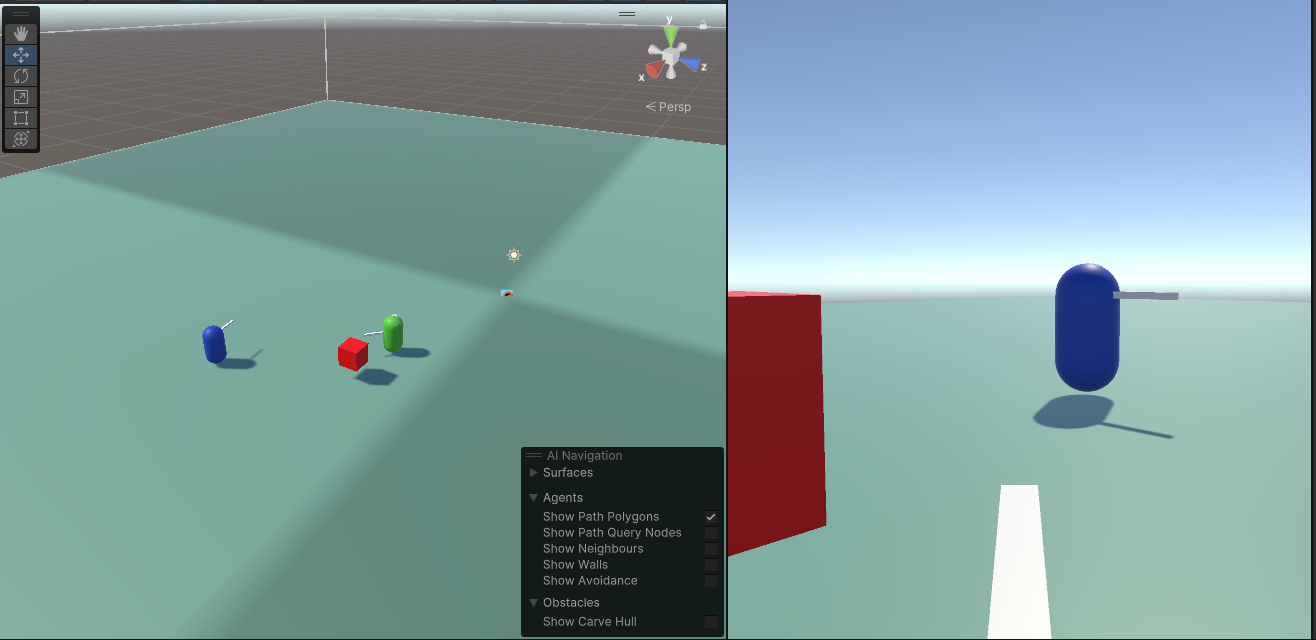


Рисунок 3 – Избегание опасности