

**федеральное государственное автономное образовательное
учреждение высшего образования**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ)
(Факультет информационных технологий)**

**(Институт Принтмедиа и информационных технологий)
Кафедра Информатики и информационных технологий**

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 2

Дисциплина: Шаблоны проектирования

Тема: Принятие решений AI персонажем

Выполнил(а): студент(ка) группы 221-3711

Денисов В.А.
(Фамилия И.О.)

Дата, подпись _____
(Дата) (Подпись)

Проверил: _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____
(Дата) (Подпись)

Замечания: _____

Москва 2024

Лабораторная работа №1

Принятие решений AI персонажем

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:

Определение стратегий:

- Определите несколько различных стратегий поведения для вашего NPC. Например, "блуждание", "преследование игрока", "уклонение от опасности" и так далее.

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.

Интеграция с NPC:

- Интегрируйте вашу систему стратегий с NPC, позволяя ему выбирать и менять стратегии в зависимости от игровой ситуации.

Тестирование:

- Запустите вашу игру и наблюдайте, как NPC меняет свое поведение в различных ситуациях, основываясь на выбранной стратегии.

Скрипты:

1. Интерфейс IMovable:

- **Свойства:** Объявляет методы для управления движением объектов (**Move**, **EnableMove**, **DisableMove**).
- **Основное свойство:** Задает контракт для объектов, которые могут двигаться и управлять своим движением.

```
interface IMovable
{
    public void Move();
    public void EnableMove();
    public void DisableMove();
}
```

2. Скрипт PlayerMovement:

- **Свойства:** Управляет движением игрока на основе пользовательского ввода, используя **Rigidbody2D**.
- **Основное свойство:** Реализует интерфейс **IMovable** для движения игрока и предоставляет методы для включения и отключения движения.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour, IMovable
{
    [SerializeField] private float _moveSpeed;
    private bool _canMove;
    private Rigidbody2D _rigidbody;

    private void Awake()
    {
        _rigidbody = GetComponent<Rigidbody2D>();
        _canMove = true;
    }

    private void FixedUpdate()
    {
        Move();
    }

    public void Move()
    {

```

```

        if (!_canMove) return;

        float moveX = Input.GetAxis("Horizontal") * _moveSpeed *
Time.fixedDeltaTime;
        float moveY = Input.GetAxis("Vertical") * _moveSpeed *
Time.fixedDeltaTime;
        _rigidbody.velocity = new Vector2(moveX, moveY);
    }

    public void EnableMove() => _canMove = true;
    public void DisableMove() => _canMove = false;
}

```

3. Абстрактный класс NPCStrategy:

- **Свойства:** Задаёт базовый класс для различных стратегий поведения NPC, с обязательным методом **Act**.
- **Основное свойство:** Обеспечивает структуру для различных стратегий поведения NPC, требуя реализации метода **Act** в подклассах.

```

using UnityEngine;
public abstract class NPCStrategy
{
    protected GameObject nPC;
    public NPCStrategy(GameObject gameObject)
    {
        nPC = gameObject;
    }
    public abstract void Act();
}

```

4. Скрипт NPCChasing:

- **Свойства:** Реализует стратегию преследования игрока для NPC, управляет движением NPC к позиции игрока.
- **Основное свойство:** Наследует **NPCStrategy** и реализует **IMovable** для управления движением NPC при преследовании игрока.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NPCChasing : NPCStrategy, IMovable
{
    private Transform player;
    private bool canMove = true;
    public NPCChasing(GameObject target, Transform player) : base(target)
    {

```

```

        this.player = player;
    }

    public override void Act()
    {
        Move();
    }
    public void Move()
    {
        if (canMove == false) return;
        npc.transform.position = Vector2.MoveTowards(npc.transform.position,
player.transform.position, 2f * Time.deltaTime);
    }
    public void EnableMove() => canMove = true;

    public void DisableMove() => canMove = false;
}

```

5. Скрипт NPCController:

- **Свойства:** Управляет поведением NPC в зависимости от расстояния до игрока, переключая стратегии между патрулированием, реагированием и преследованием.
- **Основное свойство:** Смена стратегий поведения NPC в реальном времени на основе положения игрока.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NPCController : MonoBehaviour
{
    public float playerDetectionRadius = 10f;
    public float chaseRadius = 3f;
    public float jumpRadius = 7f;

    public GameObject npc;
    public Transform player;
    private NPCStrategy behavior;

    public Transform[] patrolPoints;
    public float moveSpeed;
    public int patrolDestination;
    private NPCPatroling patroling;
    private NPCReacting reacting;
    private NPCChasing chasing;
}

```

```

private void Start()
{
    patrolling = new NPCPatrolling(npc, patrolPoints, moveSpeed,
patrolDestination);
    reacting = new NPCReacting(npc);
    chasing = new NPCChasing(npc, player);

    behavior = patrolling;
}

private void Update()
{
    float distanceToPlayer = Vector3.Distance(transform.position, player.position);

    if (distanceToPlayer <= chaseRadius)
    {
        behavior = chasing;
    }
    else if (distanceToPlayer <= jumpRadius)
    {
        behavior = reacting;
    }
    else
    {
        behavior = patrolling;
    }

    behavior.Act();
}

public void SetTarget(Transform target)
{
    player = target;
}
}

```

6. Скрипт NPCPatrolling:

- **Свойства:** Реализует стратегию патрулирования для NPC, перемещая его между заданными точками патрулирования.
- **Основное свойство:** Наследует **NPCStrategy** и управляет перемещением NPC по патрульным точкам.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

public class NPCPatrolling : NPCStrategy
{
    public Transform[] patrolPoints;
    public float moveSpeed;
    public int patrolDestination;

    public NPCPatrolling(GameObject target, Transform[] patrolPoints, float
moveSpeed, int patrolDestination) : base(target)
    {
        this.patrolPoints = patrolPoints;
        this.moveSpeed = moveSpeed;
        this.patrolDestination = patrolDestination;
    }

    public override void Act()
    {
        nPC.transform.position = Vector2.MoveTowards(nPC.transform.position,
patrolPoints[patrolDestination].position, moveSpeed);
        if (Vector2.Distance(nPC.transform.position,
patrolPoints[patrolDestination].position) < 0.2f)
        {
            patrolDestination = (patrolDestination + 1) % patrolPoints.Length;
        }
    }
}

```

7. Скрипт NPCReacting:

- **Свойства:** Реализует простую реакцию NPC на определенные события (например, печать сообщения в консоль).
- **Основное свойство:** Наследует **NPCStrategy** и выполняет базовую реакцию NPC.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NPCReacting : NPCStrategy
{
    public NPCReacting(GameObject target) : base(target) { }

    public override void Act()
    {
        Debug.Log("Who is it?");
    }
}

```

Ход работы

В данной лабораторной работе реализовано три вида поведения NPC: патрулирование, реакция на появление игрока и преследование игрока. Выбор стратегии поведения зависит от расстояния до игрока. Благодаря использованию абстрактного класса `NPCStrategy`, мы можем вызывать метод `Act` для выполнения поведения NPC, не указывая конкретную реализацию метода.

Когда игрок находится на расстоянии более 7 игровых единиц, NPC переходит в режим патрулирования, перемещаясь между заданными точками. Если игрок приближается на расстояние от 7 до 3 игровых единиц, NPC останавливается и реагирует на присутствие игрока (в нашем случае это выражается в выводе сообщения в консоль). Когда игрок находится ближе 3 игровых единиц, NPC начинает преследование, стараясь догнать игрока.

Использование абстрактного класса позволяет легко переключаться между различными стратегиями поведения NPC в зависимости от ситуации, что делает систему более гибкой и расширяемой.

Ссылка на проект в Github: