**Лабораторна робота №3**

**Тема:** Вивчення поведінкових шаблонів проектування.

**Мета:** Набути практичних навичок в використанні поведінкових шаблонів використання.

***Завдання:***

Вивчити та реалізувати шаблони проектування згідно варіанту. Оформити звіт, у якому описати предметну область та проблему, яку вирішуватимете за допомогою шаблона проектування, здійснити опис шаблона проектування, діаграма класів шаблону, реалізація та приклад використання програми(результати роботи).

***Варіант 6***

* Ланцюг відповідальностей.
* Шаблонний метод.
* Медіатор.

***Хід роботи***

***Ланцюг відповідальностей – шаблон проектування який є ланцюгом обробників, по якому «проходиться» інформація. Якщо один обробник не може обробити повідомлення, він передає його іншому обробнику.***

***Код:***

namespace BPatern\_Chain\_Of\_Responses

{

class Program

{

static void Main(string[] args)

{

Banana\_Havatel havatel1 = new Banana\_Havatel();

Orange\_Havatel havatel2 = new Orange\_Havatel();

Coockie\_Havatel havatel3 = new Coockie\_Havatel();

havatel1.next\_havatel = havatel2;

havatel2.next\_havatel = havatel3;

havatel1.toEat("Шини від машини");

havatel1.toEat("Печенька");

Console.Read();

}

}

abstract public class Havatel

{

public string havka;

public Havatel next\_havatel;

public abstract void toEat(string the\_havka);

}

public class Banana\_Havatel : Havatel

{

public override void toEat(string the\_havka)

{

havka = the\_havka;

if (havka == "Банан") { Console.WriteLine("Хаватель Бананів: Хаваю "+havka+"... "); }

else

{

if (next\_havatel != null)

{

Console.WriteLine("Хаватель Бананів: Я "+havka+" не хаваю, так шо даю следующему. ");

next\_havatel.toEat(havka);

} else { Console.WriteLine("Хаватель Бананів: Хавателів більше нема, так шо ніхто більше не буде хавати. "); }

}

}

}

public class Orange\_Havatel : Havatel

{

public override void toEat(string the\_havka)

{

havka = the\_havka;

if (havka == "Апельсин") { Console.WriteLine("Хаватель Апельсинів: Хаваю " + havka + "... "); }

else

{

if (next\_havatel != null)

{

Console.WriteLine("Хаватель Апельсинів: Я " + havka + " не хаваю, так шо даю следующему. ");

next\_havatel.toEat(havka);

}

else { Console.WriteLine("Хаватель Апельсинів: Хавателів більше нема, так шо ніхто більше не буде хавати. "); }

}

}

}

public class Coockie\_Havatel : Havatel

{

public override void toEat(string the\_havka)

{

havka = the\_havka;

if (havka == "Печенька") { Console.WriteLine("Хаватель Печеньок: Хаваю " + havka + "... "); }

else

{

if (next\_havatel != null)

{

Console.WriteLine("Хаватель Печеньок: Я " + havka + " не хаваю, так шо даю следующему. ");

next\_havatel.toEat(havka);

}

else { Console.WriteLine("Хаватель Печеньок: Хавателів більше нема, так шо ніхто більше не буде хавати. "); }}}}}

***Шаблонний метод – шаблон проектування, який застосовується для того щоб вносити мінімальні правки в код программи не копіюючи його повністью. Використовується якщо є декілька дуже схожих алгоритмів. Основу заносять у базовий клас, відмінності реалізують через похідні класи.***

***Код:***

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

abstract public class Patern : MonoBehaviour

{

public GameObject item;

public bool isBuilding;

public Vector3 current\_Coord;

public void InstUp()

{

current\_Coord.y++;

Destroy(Instantiate(item, current\_Coord, item.transform.rotation), 10);

}

public void InstLeft()

{

current\_Coord.x--;

Destroy(Instantiate(item, current\_Coord, item.transform.rotation),10);

}

public void InstRight()

{

current\_Coord.x++;

Destroy(Instantiate(item, current\_Coord, item.transform.rotation), 10);

}

public abstract IEnumerator differ();

public IEnumerator Building()

{

isBuilding = true;

InstUp();

yield return new WaitForSeconds(0.6f);

StartCoroutine(differ());

yield return new WaitForSeconds(0.6f);

}

// Use this for initialization

void Start () {

} // Update is called once per frame

void Update ()}

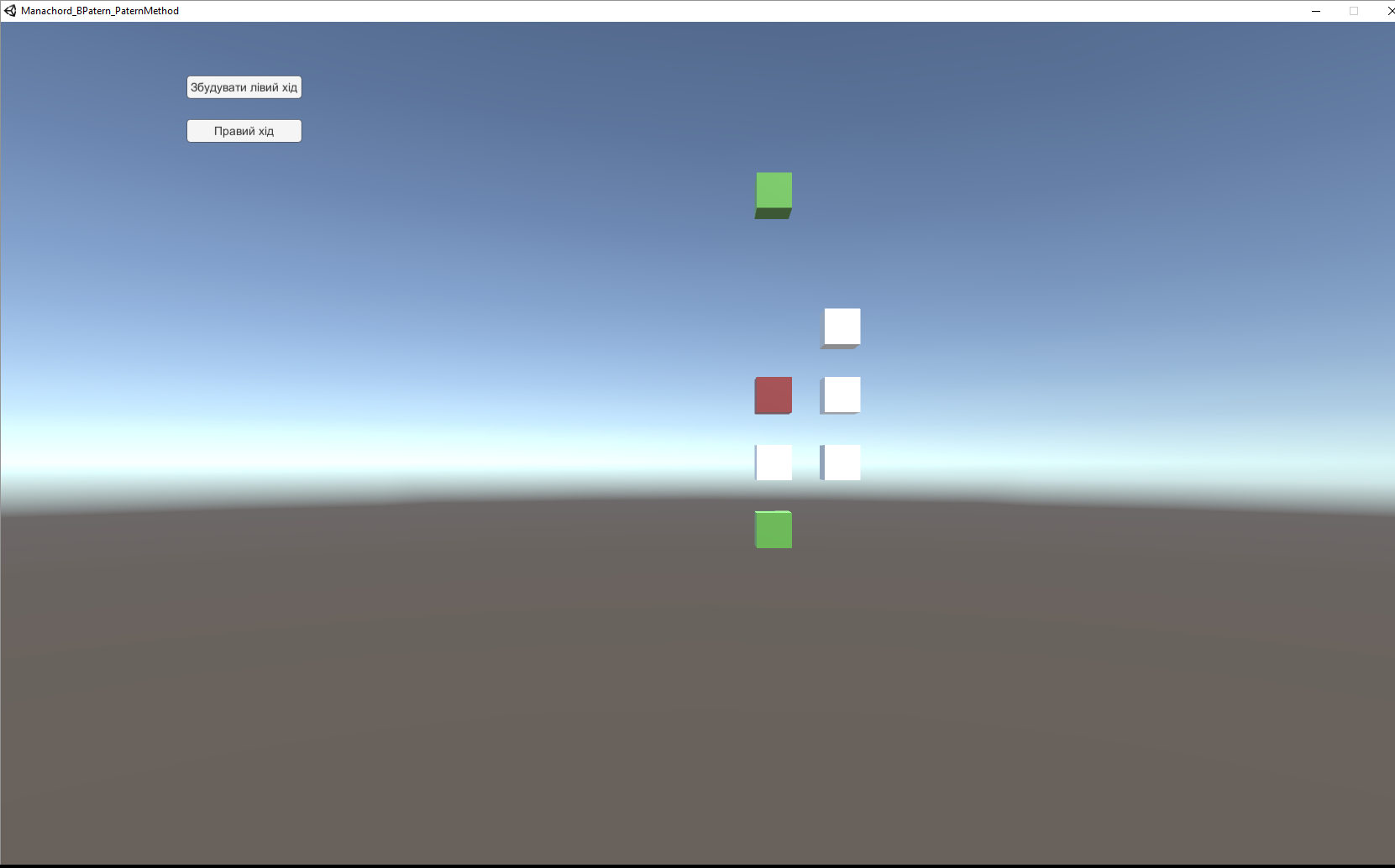


Рисунок 1. Використання одного з похідних алгоритмів.

***Медіатор – шаблон проектування який забезпечує зв’язок між багатьма класами, при чому позбавляє класів необхідності явно посилатись один на іншого. Код:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Mediator\_BPatern

{

class Program

{

static void Main(string[] args)

{

GameManager MainGameManager = new GameManager();

Player Vasya = new Player("Vasya", MainGameManager);

Player Lykan = new Player("Lykan", MainGameManager);

Player Gorch = new Player("Gorch", MainGameManager);

MainGameManager.players.Add(Vasya);

MainGameManager.players.Add(Lykan);

MainGameManager.players.Add(Gorch);

MainGameManager.CheckMiniMap();

Vasya.Move();

MainGameManager.CheckMiniMap();

Console.ReadKey();

}

public class GameManager

{

public List<Player> players = new List<Player>();

public void CheckMiniMap()

{

foreach (var pl in players) { pl.CheckMiniMap(); }

}

}

public class Player

{

GameManager gm;

int x, y, z;

string name;

public Player(string Name, GameManager GM) { name = Name; gm = GM; }

public void Move()

{

x++;y++;z++;

}

public void CheckMiniMap()

{

foreach (var pl in gm.players)

{

Console.WriteLine("Гравець "+name+" побачив що у гравця "+pl.name+ " такі координати.");

Console.WriteLine("X: " +pl.x);

Console.WriteLine("X: " + pl.y);

Console.WriteLine("X: " + pl.z);

}

}

}

}

}

Висновки: На цій лабораторній роботі, я навчився використовувати такі поведінкові шаблони проектування як : Ланцюг відповідальностей, Медіатор та Посредник. А також я закріпив навички програмування на мові C#.