KAUNO TECHNOLOGIJOS UNIVERSITETAS

INformatikos fakultetas

T120B516 Objektinis programų projektavimas

Projekto ataskaita

Studentų komanda:

Mantas Zambacevičius IFF-4/1,

Ernestas Venckus IFF-4/3,

Žilvinas Abromavičius IFF-4/3

Dėstytojas:

lekt. Barisas Dominykas

KAUNAS, 2017

Turinys

[1 Kuriamo žaidimo aprašymas 3](#_Toc496547958)

[2 Naudojami šablonai 3](#_Toc496547959)

[2.1 Singleton 3](#_Toc496547960)

[2.1.1 Klasių diagrama 3](#_Toc496547961)

[2.1.2 Esminis kodas 3](#_Toc496547962)

[2.1.3 Naudojimo pagrindimas 4](#_Toc496547963)

[2.2 Factory 4](#_Toc496547964)

[2.2.1 Klasių diagrama 4](#_Toc496547965)

[2.2.2 Esminis kodas 4](#_Toc496547966)

[2.2.3 Naudojimo pagrindimas 5](#_Toc496547967)

[2.3 Abstract factory 5](#_Toc496547968)

[2.3.1 Klasių diagrama 5](#_Toc496547969)

[2.3.2 Esminis kodas 6](#_Toc496547970)

[2.3.3 Naudojimo pagrindimas 7](#_Toc496547971)

[2.4 Observer 8](#_Toc496547972)

[2.4.1 Klasių diagrama 8](#_Toc496547973)

[2.4.2 Esminis kodas 8](#_Toc496547974)

[2.4.3 Naudojimo pagrindimas 9](#_Toc496547975)

[2.5 Adapter 10](#_Toc496547976)

[2.5.1 Klasių diagrama 10](#_Toc496547977)

[2.5.2 Esminis kodas 10](#_Toc496547978)

[2.5.3 Naudojimo pagrindimas 11](#_Toc496547979)

[2.6 Prototype 11](#_Toc496547980)

[2.6.1 Klasių diagrama 11](#_Toc496547981)

[2.6.2 Esminis kodas 12](#_Toc496547982)

[2.6.3 Naudojimo pagrindimas 14](#_Toc496547983)

[2.7 Command 14](#_Toc496547984)

[2.7.1 Klasių diagrama 14](#_Toc496547985)

[2.7.2 Esminis kodas 15](#_Toc496547986)

[2.7.3 Naudojimo pagrindimas 17](#_Toc496547987)

[2.8 Bridge 17](#_Toc496547988)

[2.8.1 Klasių diagrama 17](#_Toc496547989)

[2.8.2 Esminis kodas 17](#_Toc496547990)

[2.8.3 Naudojimo pagrindimas 18](#_Toc496547991)

[3 Išvados 18](#_Toc496547992)

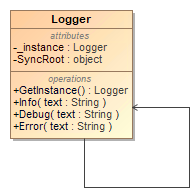
# Kuriamo žaidimo aprašymas

Projekto tikslas – sukurti nedidelį žaidimą, taikant projektavimo šablonus. Kuriamas žaidimas yra 2D šaudyklė. Projektas išskaidytas į 4 laboratorinius darbus. Pirmajam laboratoriniam darbui realizavome Singleton, Factory, Abstract Factory bei Observer šablonus. Antrajam darbui – Adapter, Prototype, Command ir Bridge.

# Naudojami šablonai

## Singleton

### Klasių diagrama



### Esminis kodas

using System;

namespace Shooter.Classes

{

public class Logger

{

private static volatile Logger \_instance;

private static readonly object SyncRoot = new object();

private Logger() { }

public static Logger Instance

{

get

{

if (\_instance == null)

{

lock (SyncRoot)

{

if(\_instance == null)

\_instance = new Logger();

}

}

return \_instance;

}

}

public void Info(string text) {…}

public void Debug(string text) {…}

public void Error(string text) {…}

private void WriteFormattedLog(LogLevel level, string text) {…}

private enum LogLevel {…}

}

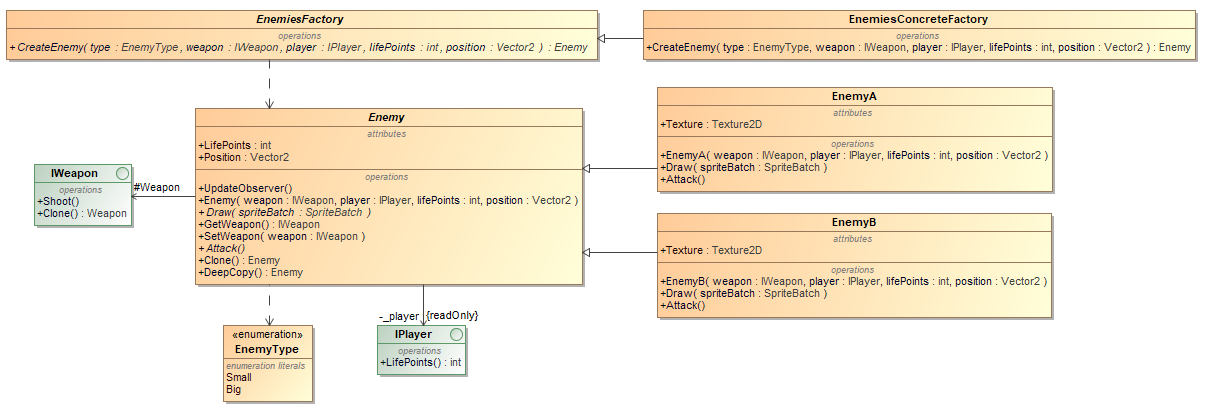
}

### Naudojimo pagrindimas

Logger klasės paskirtis išvesti informacijos, derinimo ir klaidų pranešimus į konsolę. Kadangi visoje programoje reikalingas tik vienas šios klasės objektas, šiai klasėj pritaikomas „Singleton“ dizaino šablonas.

## Factory

### Klasių diagrama



### Esminis kodas

using Microsoft.Xna.Framework;

using Shooter.Classes;

using Shooter.Enums;

using Shooter.Interfaces;

namespace Shooter.PatternClasses

{

public abstract class EnemiesFactory

{

public abstract Enemy CreateEnemy(EnemyType type, IWeapon weapon, IPlayer player, int lifePoints, Vector2 position, Texture2D texture);

}

}

using Microsoft.Xna.Framework;

using Shooter.Classes;

using Shooter.Enums;

using Shooter.Interfaces;

namespace Shooter.PatternClasses

{

public class EnemiesConcreteFactory : EnemiesFactory

{

public override Enemy CreateEnemy(EnemyType type, IWeapon weapon, IPlayer player, int lifePoints, Vector2 position, Texture2D texture)

{

switch (type)

{

case EnemyType.Small:

return new EnemyA(weapon, player, lifePoints, position, texture);

case EnemyType.Big:

return new EnemyB(weapon, player, lifePoints, position, texture);

default:

return null;

}

}

}

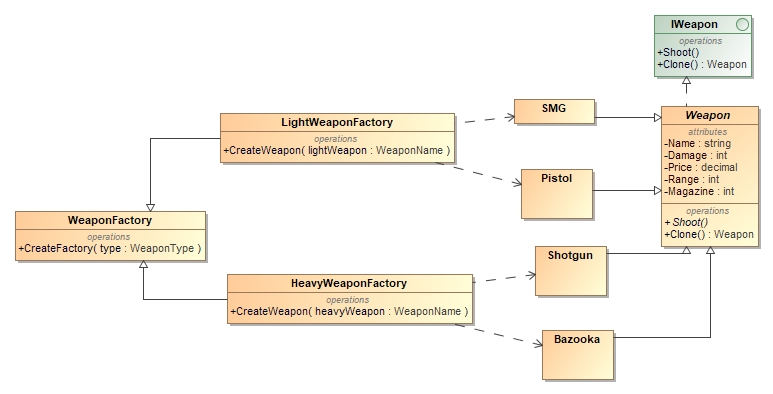
}

### Naudojimo pagrindimas

EnemiesFactory klasės paskirtis yra kurti nurodyto tipo Enemy objektus inkapsuliuojant patį sukūrimą, taip leidžiant vartotojui nesigilinti kokios klasės objektas buvo sukurtas.

## Abstract factory

### Klasių diagrama



### Esminis kodas

using Shooter.Classes;

using Shooter.Enums;

namespace Shooter.PatternClasses

{

public class WeaponFactory

{

public static WeaponFactory CreateFactory(WeaponType type)

{

switch (type)

{

case WeaponType.Heavy:

return new HeavyWeaponFactory();

case WeaponType.Light:

return new LightWeaponFactory();

default:

return null;

}

}

public virtual Weapon CreateWeapon(WeaponName weapon)

{

return null;

}

}

}

using Shooter.Classes;

using Shooter.Enums;

namespace Shooter.PatternClasses

{

public class LightWeaponFactory : WeaponFactory

{

public override Weapon CreateWeapon(WeaponName lightWeapon)

{

switch (lightWeapon)

{

case WeaponName.Pistol:

return new Pistol();

case WeaponName.SMG:

return new SMG();

default:

return null;

}

}

}

}

using Shooter.Classes;

using Shooter.Enums;

namespace Shooter.PatternClasses

{

public class HeavyWeaponFactory : WeaponFactory

{

public override Weapon CreateWeapon(WeaponName heavyWeapon)

{

switch (heavyWeapon)

{

case WeaponName.Bazooka:

return new Bazooka();

case WeaponName.Shotgun:

return new Shotgun();

default:

return null;

}

}

}

}

using System;

namespace Shooter.Classes

{

public class SMG : Weapon

{

public SMG() : base("SMG")

{

}

public override void Shoot()

{

throw new NotImplementedException();

}

}

}

namespace Shooter.Classes

{

public class Shotgun : Weapon

{

public Shotgun() : base("Shotgun")

{

}

public override void Shoot()

{

throw new System.NotImplementedException();

}

}

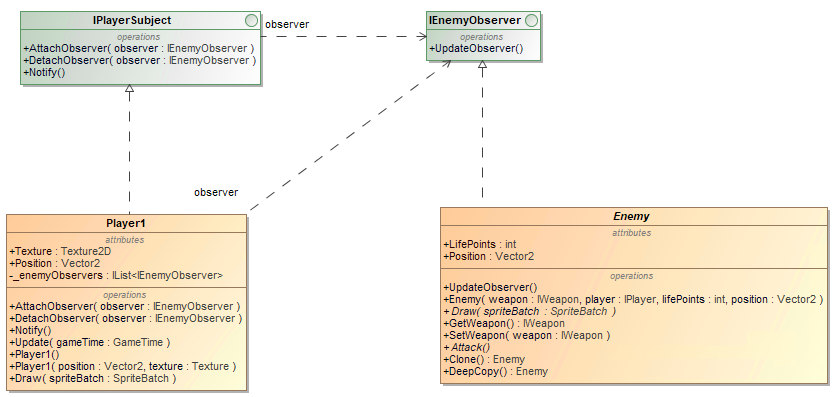
}

### Naudojimo pagrindimas

WeaponFactory klasės paskirtis kurti ginklus žaidimo žemėlapyje, nenurodant konkrečių klasių.

## Observer

### Klasių diagrama



### Esminis kodas

namespace Shooter.Interfaces

{

interface IPlayerSubject

{

void AttachObserver(IEnemyObserver enemy);

void DetachObserver(IEnemyObserver enemy);

void Notify();

}

}

namespace Shooter.Interfaces

{

public interface IEnemyObserver

{

void UpdateObserver();

}

}

namespace Shooter.Classes

{

public abstract class Enemy : IEnemy, IEnemyObserver, IMapObject, EnemyPrototype

{

public Vector2 Position { get; set; }

public int LifePoints { get; set; }

private readonly IPlayer \_player;

protected IWeapon Weapon;

public abstract void Draw(SpriteBatch spriteBatch);

protected Enemy(IWeapon weapon, IPlayer player, int lifePoints, Vector2 position)

{

Weapon = weapon;

\_player = player;

LifePoints = lifePoints;

Position = position;

}

public IWeapon GetWeapon() {…}

public void SetWeapon(IWeapon weapon) {…}

public abstract void Attack();

public virtual void UpdateObserver()

{

Console.WriteLine($"Enemy notified of life points {\_player.LifePoints}");

}

public Enemy Clone() {…}

public Enemy DeepCopy() {…}

}

}

namespace Shooter.Classes

{

public class Player1 : IPlayer, IPlayerSubject, IMapObject

{

private readonly IList<IEnemyObserver> \_enemyObservers;

public Texture2D Texture { get; set; }

public Vector2 Position { get; set; }

public void Draw(SpriteBatch spriteBatch) {…}

public void Update(GameTime gameTime) {…}

public int LifePoints { get; set; }

public Player1() {…}

public Player1(Vector2 position, Texture2D texture) {…}

public void AttachObserver(IEnemyObserver observer)

{

\_enemyObservers.Add(observer);

}

public void DetachObserver(IEnemyObserver observer)

{

\_enemyObservers.Remove(observer);

}

public void Notify()

{

foreach (var enemyObserver in \_enemyObservers)

{

enemyObserver.UpdateObserver();

}

}

public void Move(KeyboardState keyboardState, KeyboardState previousState) {…}

}

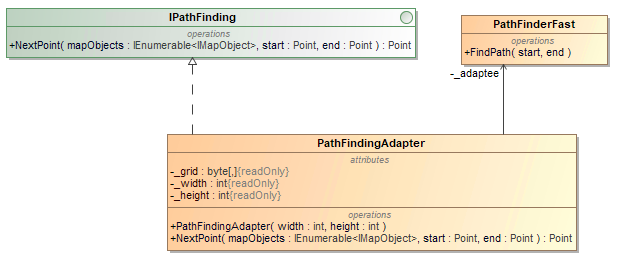
}

### Naudojimo pagrindimas

„Observer“ dizaino šablono naudojimo paskirtis: kai egzistuoja „vienas-su-daug“ ryšis tarp objektų, ir norima vieno objekto būsenos pasikeitimus perduoti daugeliui kitų objektų. Projekto atveju, šis šablonas įgyvendintas tarp Player1 ir Enemy klasių: kai pasikeičia Player1 būsena (pvz. pasikeičia žaidėjo pozicija žaidimo lauke), tai yra informuojama visiems Enemy klasės objektams.

## Adapter

### Klasių diagrama



### Esminis kodas

namespace Shooter.Interfaces

{

public interface IPathFinding

{

Point NextPoint(IEnumerable<IMapObject> mapObjects, Point start, Point end);

}

}

namespace Shooter.PatternClasses

{

class PathFindingAdapter : IPathFinding

{

private readonly PathFinderFast \_adaptee;

private readonly byte[,] \_grid;

private readonly int \_width;

private readonly int \_height;

public PathFindingAdapter(int width, int height)

{

\_width = width;

\_height = height;

\_grid = new byte[width,height];

\_adaptee = new PathFinderFast(\_grid);

}

public Point NextPoint(IEnumerable<IMapObject> mapObjects, Point start, Point end)

{

for (var i = 0; i < \_width; i++)

{

for (var j = 0; j < \_height; j++)

{

\_grid[i, j] = PathFinderHelper.EMPTY\_TILE;

}

}

foreach (var mapObject in mapObjects)

{

\_grid[(int) mapObject.Position.X, (int) mapObject.Position.Y] = PathFinderHelper.BLOCKED\_TILE;

}

var path = \_adaptee.FindPath(new DeenGames.Utils.Point(start.X, start.Y), new DeenGames.Utils.Point(end.X, end.Y));

if (path == null)

{

return Point.Zero;

}

var index = path.Count - 2;

return new Point(path[index].X, path[index].Y);

}

}

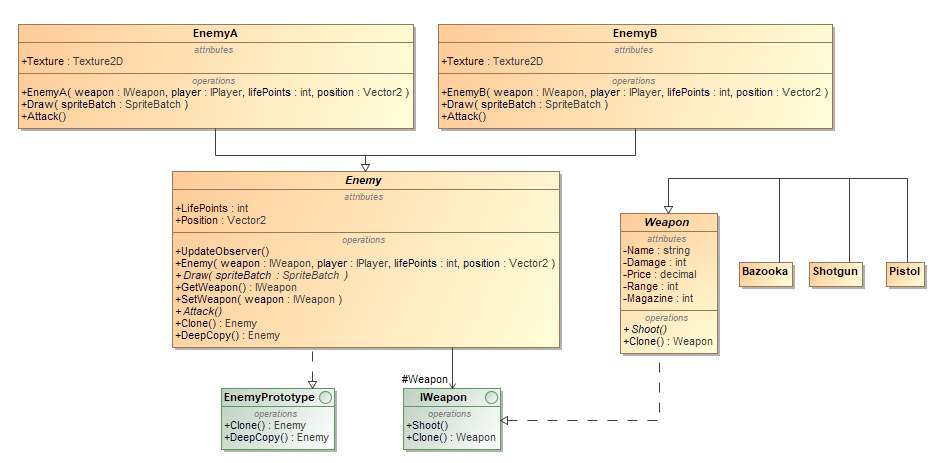
}

### Naudojimo pagrindimas

Projekte yra paskirtis naudoti A\* kelio radimo algoritmą. Dėl šios priežasties buvo nuspręsta ne implementuoti jį patiems, bet panaudoti išorinę biblioteką: AStarPathFinder.dll. Projekto kelio radimo sąsaja nėra suderinama su šios bibliotekos siūloma sąsaja. Dėl šios priežasties buvo nuspręsta naudoti Adapter dizaino šabloną, kad projektas galėtų naudoti šią išorinę biblioteką.

## Prototype

### Klasių diagrama



### Esminis kodas

using Shooter.Classes;

namespace Shooter.PatternClasses

{

public interface EnemyPrototype

{

Enemy Clone();

Enemy DeepCopy();

}

}

using Shooter.Interfaces;

using System;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Shooter.PatternClasses;

namespace Shooter.Classes

{

public abstract class Enemy : IEnemy, IEnemyObserver, IMapObject, EnemyPrototype

{

public Vector2 Position { get; set; }

public int LifePoints { get; set; }

public Texture2D Texture { get; set; }

private readonly IPlayer \_player;

protected IWeapon Weapon;

public abstract void Draw(SpriteBatch spriteBatch);

protected Enemy(IWeapon weapon, IPlayer player, int lifePoints, Vector2 position, Texture2D texture)

{

Weapon = weapon;

\_player = player;

LifePoints = lifePoints;

Position = position;

Texture = texture;

}

public IWeapon GetWeapon()

{

return Weapon;

}

public void SetWeapon(IWeapon weapon)

{

Weapon = weapon;

}

public abstract void Attack();

public virtual void UpdateObserver()

{

Console.WriteLine($"Enemy notified of life points {\_player.LifePoints}");

}

public Enemy Clone()

{

return (Enemy)this.MemberwiseClone();

}

public Enemy DeepCopy()

{

Enemy enemy = (Enemy)this.MemberwiseClone();

IWeapon weapon = enemy.GetWeapon().Clone();

enemy.SetWeapon(weapon);

return enemy;

}

}

}

using Shooter.Classes;

namespace Shooter.Interfaces

{

public interface IWeapon

{

void Shoot();

Weapon Clone();

}

}

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Shooter.Interfaces;

using System;

namespace Shooter.Classes

{

public abstract class Weapon : IWeapon, IMapObject

{

public Texture2D Texture { get; set; }

public string TextureName { get; set; }

public Vector2 Position { get; set; }

public string Name { get; set; }

public int Damage { get; set; }

public decimal Price { get; set; }

public int Range { get; set; }

public int Magazine { get; set; }

public Weapon(string texture)

{

var randomPosition = new Random();

TextureName = texture;

Position = new Vector2(randomPosition.Next(0, 32) \* GameSettings.TilesSize, randomPosition.Next(0, 32) \* GameSettings.TilesSize);

}

public abstract void Shoot();

public Weapon Clone()

{

return (Weapon)this.MemberwiseClone();

}

public void Draw(SpriteBatch spriteBatch)

{

spriteBatch.Draw(Texture, Position, Color.White);

}

}

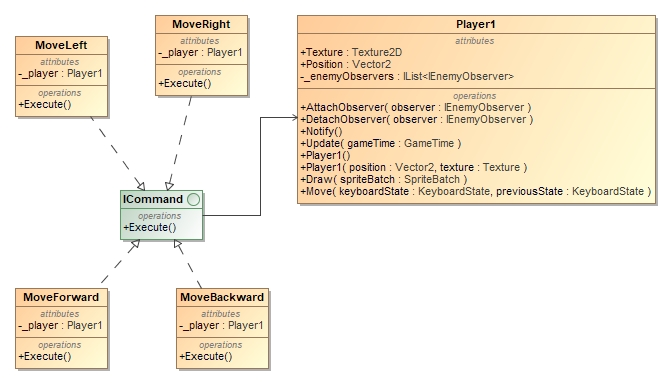
}

### Naudojimo pagrindimas

Prototype šablonas leidžia kurti objektų kopijas (šiuo atveju Enemy klasės objektų kopijas), taip sutaupydamas resursus naujo objekto kūrimo atžvilgiu. Tai taip pat palengvina vienodų Enemy objektų kūrimą, kadangi nereikia kiekvieną kartą iš naujo sukurti naują tokį patį objektą.

## Command

### Klasių diagrama



### Esminis kodas

namespace Shooter.Interfaces

{

public interface ICommand

{

void Execute();

}

}

using Microsoft.Xna.Framework;

using Shooter.Classes;

using Shooter.Interfaces;

namespace Shooter.PatternClasses

{

public class MoveBackward : ICommand

{

private readonly Player1 \_player;

public MoveBackward(Player1 player)

{

\_player = player;

}

public void Execute()

{

var position = \_player.Position;

\_player.Position = new Vector2(position.X, position.Y + GameSettings.TilesSize);

}

}

}

using Microsoft.Xna.Framework;

using Shooter.Classes;

using Shooter.Interfaces;

namespace Shooter.PatternClasses

{

public class MoveForward : ICommand

{

private readonly Player1 \_player;

public MoveForward(Player1 player)

{

\_player = player;

}

public void Execute()

{

var position = \_player.Position;

\_player.Position = new Vector2(position.X, position.Y -GameSettings.TilesSize);

}

}

}

using Microsoft.Xna.Framework;

using Shooter.Classes;

using Shooter.Interfaces;

namespace Shooter.PatternClasses

{

public class MoveLeft : ICommand

{

private readonly Player1 \_player;

public MoveLeft(Player1 player)

{

\_player = player;

}

public void Execute()

{

var position = \_player.Position;

\_player.Position = new Vector2(position.X - GameSettings.TilesSize, position.Y);

}

}

}

using Microsoft.Xna.Framework;

using Shooter.Classes;

using Shooter.Interfaces;

namespace Shooter.PatternClasses

{

public class MoveRight : ICommand

{

private readonly Player1 \_player;

public MoveRight(Player1 player)

{

\_player = player;

}

public void Execute()

{

var position = \_player.Position;

\_player.Position = new Vector2(position.X + GameSettings.TilesSize, position.Y);

}

}

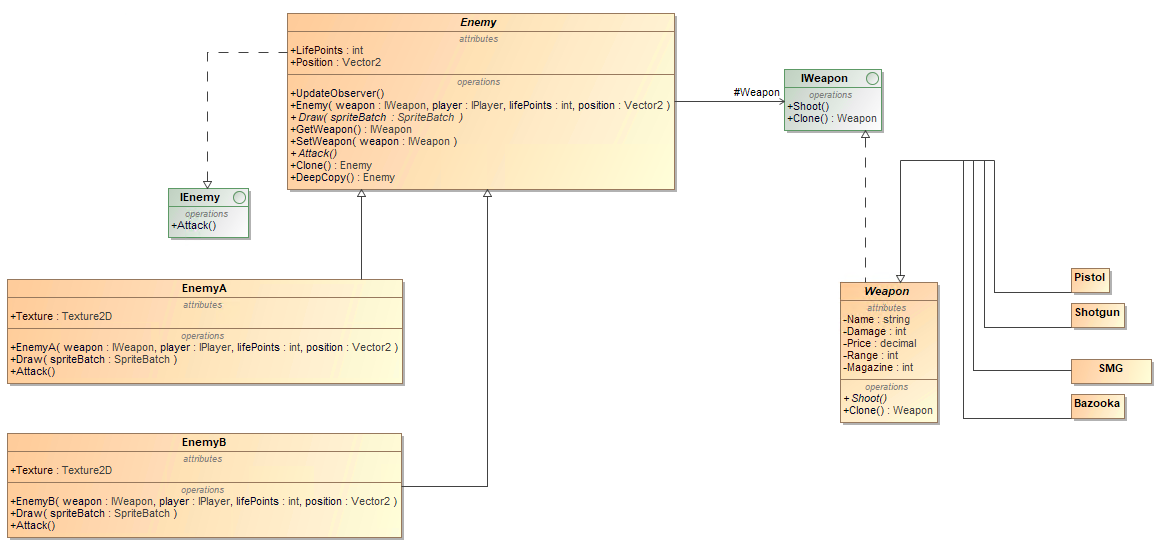
}

### Naudojimo pagrindimas

Command šablonas pritaikytas žaidėjo judėjimo valdymui. Naudojant šablona buvo atskirta klasė kuri kviečia operacija nuo objekto kuris žino kaip operaciją vykdyti.

## Bridge

### Klasių diagrama



### Esminis kodas

namespace Shooter.Classes

{

public abstract class Enemy : IEnemy, IEnemyObserver, IMapObject, EnemyPrototype

{

public Vector2 Position { get; set; }

public int LifePoints { get; set; }

private readonly IPlayer \_player;

protected IWeapon Weapon;

public abstract void Draw(SpriteBatch spriteBatch);

protected Enemy(IWeapon weapon, IPlayer player, int lifePoints, Vector2 position)

{

Weapon = weapon;

\_player = player;

LifePoints = lifePoints;

Position = position;

}

public IWeapon GetWeapon() {…}

public void SetWeapon(IWeapon weapon) {…}

public abstract void Attack();

public virtual void UpdateObserver() {…}

public Enemy Clone() {…}

public Enemy DeepCopy() {…}

}

}

namespace Shooter.Interfaces

{

public interface IWeapon

{

void Shoot();

Weapon Clone();

}

}

### Naudojimo pagrindimas

Observer dizaino šablono paskirtis yra atskirti abstrakciją nuo realizacijos, kad abi galėtų įvairuoti nepriklausomai. Projekte tai buvo įgyvendinta tarp Enemy abstrakčios klasės ir IWeapon sąsajos. Taip nuspręsta, nes projekte gali būti daug skirtingų Enemy subklasių ir daug skirtingų IWeapon realizuojančių klasių, o kiekvienas priešas turi tam tikrą ginklą. Šis ryšys tarp priešo ir ginklo buvo suabstraktintas, kad ryšyje dalyvautų tik abstrakti klasė Enemy ir ginklų sąsaja IWeapon, vietoj to kad kiekvienas atskiras priešas turėtų ryšį su kiekvienu skirtingu ginklu.

# Išvados