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

Priėmė:

lekt. Andrej Ušaniov

lekt. Dominykas Barisas

KAUNAS, 2017

Turinys

[1 Kuriamo žaidimo aprašymas 4](#_Toc500803347)

[2 Naudojami šablonai 4](#_Toc500803348)

[2.1 Singleton 4](#_Toc500803349)

[2.1.1 Klasių diagrama 4](#_Toc500803350)

[2.1.2 Esminis kodas 4](#_Toc500803351)

[2.1.3 Naudojimo pagrindimas 5](#_Toc500803352)

[2.2 Factory 5](#_Toc500803353)

[2.2.1 Klasių diagrama 5](#_Toc500803354)

[2.2.2 Esminis kodas 5](#_Toc500803355)

[2.2.3 Naudojimo pagrindimas 6](#_Toc500803356)

[2.3 Abstract factory 6](#_Toc500803357)

[2.3.1 Klasių diagrama 6](#_Toc500803358)

[2.3.2 Esminis kodas 7](#_Toc500803359)

[2.3.3 Naudojimo pagrindimas 8](#_Toc500803360)

[2.4 Observer 9](#_Toc500803361)

[2.4.1 Klasių diagrama 9](#_Toc500803362)

[2.4.2 Esminis kodas 9](#_Toc500803363)

[2.4.3 Naudojimo pagrindimas 10](#_Toc500803364)

[2.5 Adapter 11](#_Toc500803365)

[2.5.1 Klasių diagrama 11](#_Toc500803366)

[2.5.2 Esminis kodas 11](#_Toc500803367)

[2.5.3 Naudojimo pagrindimas 12](#_Toc500803368)

[2.6 Prototype 12](#_Toc500803369)

[2.6.1 Klasių diagrama 12](#_Toc500803370)

[2.6.2 Esminis kodas 13](#_Toc500803371)

[2.6.3 Naudojimo pagrindimas 15](#_Toc500803372)

[2.7 Command 15](#_Toc500803373)

[2.7.1 Klasių diagrama 15](#_Toc500803374)

[2.7.2 Esminis kodas 15](#_Toc500803375)

[2.7.3 Naudojimo pagrindimas 17](#_Toc500803376)

[2.8 Bridge 17](#_Toc500803377)

[2.8.1 Klasių diagrama 17](#_Toc500803378)

[2.8.2 Esminis kodas 18](#_Toc500803379)

[2.8.3 Naudojimo pagrindimas 19](#_Toc500803380)

[2.9 Composite 19](#_Toc500803381)

[2.9.1 Klasių diagrama 19](#_Toc500803382)

[2.9.2 Esminis kodas 19](#_Toc500803383)

[2.9.3 Naudojimo pagrindimas 21](#_Toc500803384)

[2.10 Chain of Responsibility 21](#_Toc500803385)

[2.10.1 Klasių diagrama 21](#_Toc500803386)

[2.10.2 Esminis kodas 21](#_Toc500803387)

[2.10.3 Naudojimo paskirtis 22](#_Toc500803388)

[2.11 State 23](#_Toc500803389)

[2.11.1 Klasių diagrama 23](#_Toc500803390)

[2.11.2 Esminis kodas 23](#_Toc500803391)

[2.11.3 Naudojimo paskirtis 24](#_Toc500803392)

[2.12 Mediator 24](#_Toc500803393)

[2.12.1 Klasių diagrama 24](#_Toc500803394)

[2.12.2 Esminis kodas 24](#_Toc500803395)

[2.12.3 Naudojimo paskirtis 25](#_Toc500803396)

[2.13 Visitor 25](#_Toc500803397)

[2.13.1 Klasių diagrama 25](#_Toc500803398)

[2.13.2 Esminis kodas 25](#_Toc500803399)

[2.13.3 Naudojimo pagrindimas 26](#_Toc500803400)

[2.14 Memento 27](#_Toc500803401)

[2.14.1 Klasių diagrama 27](#_Toc500803402)

[2.14.2 Esminis kodas 27](#_Toc500803403)

[2.14.3 Naudojimo pagrindimas 28](#_Toc500803404)

[2.15 Flyweight 28](#_Toc500803405)

[2.15.1 Klasių diagrama 28](#_Toc500803406)

[2.15.2 Esminis kodas 29](#_Toc500803407)

[2.15.3 Naudojimo pagrindimas 30](#_Toc500803408)

[2.16 Template method 30](#_Toc500803409)

[2.16.1 Klasių diagrama 30](#_Toc500803410)

[2.16.2 Esminis kodas 30](#_Toc500803411)

[2.16.3 Naudojimo pagrindimas 31](#_Toc500803412)

[3 Išvados 31](#_Toc500803413)

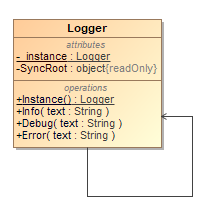
# Kuriamo žaidimo aprašymas

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 {…}

}

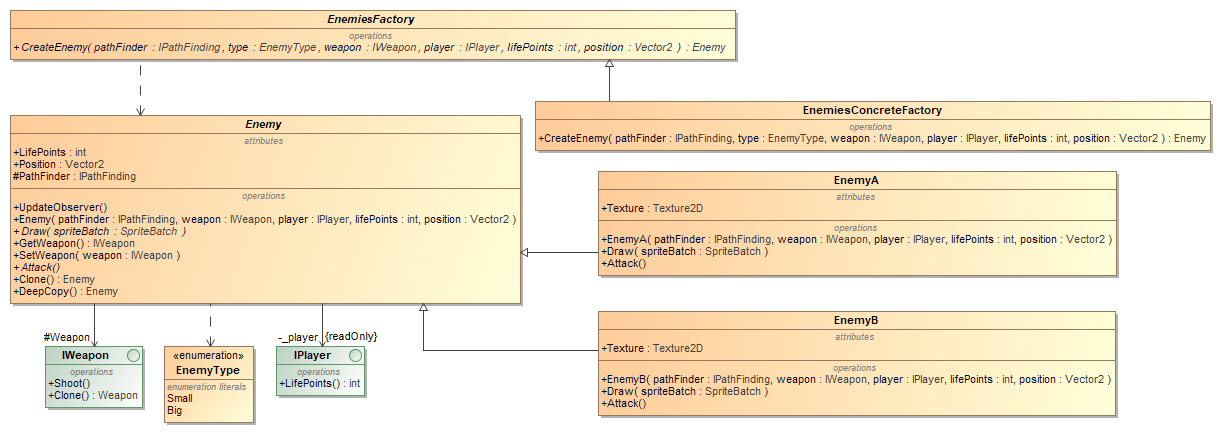
}

### 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;

}

}

}

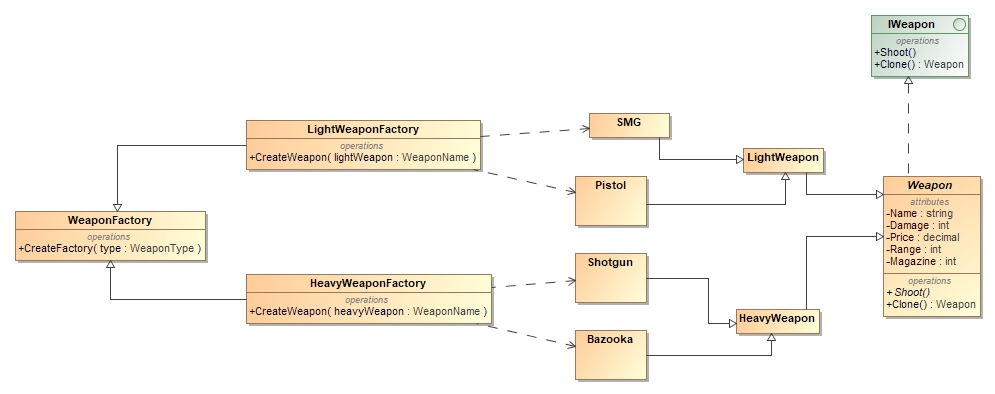
}

### 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();

}

}

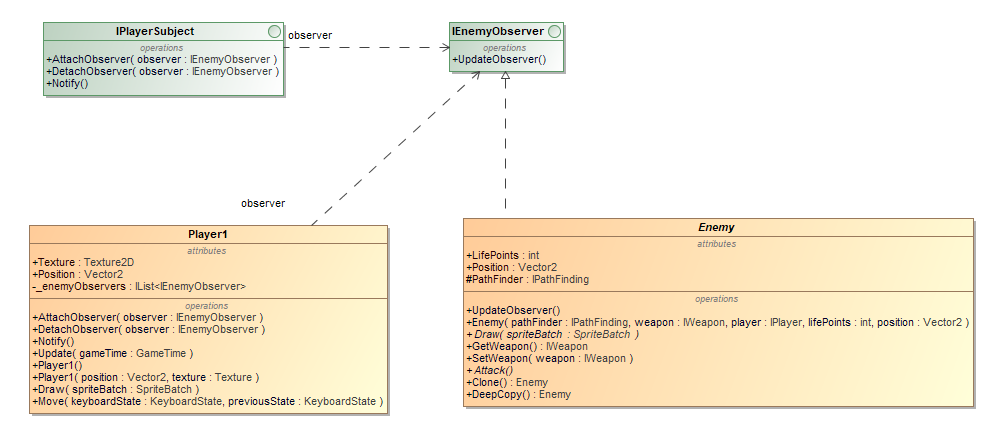
}

### 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) {…}

}

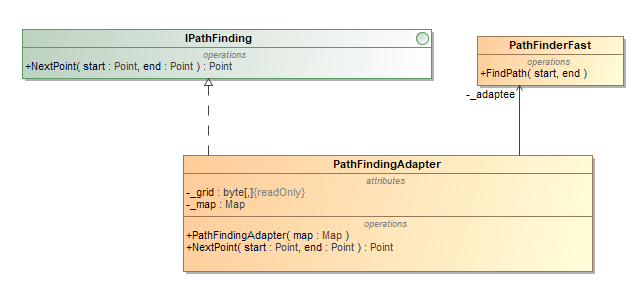
}

### 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);

}

}

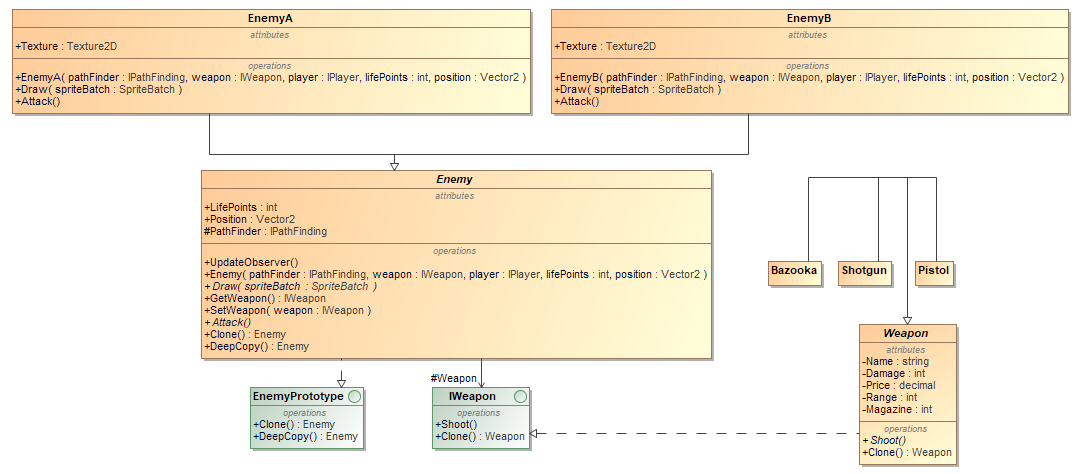
}

### 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);

}

}

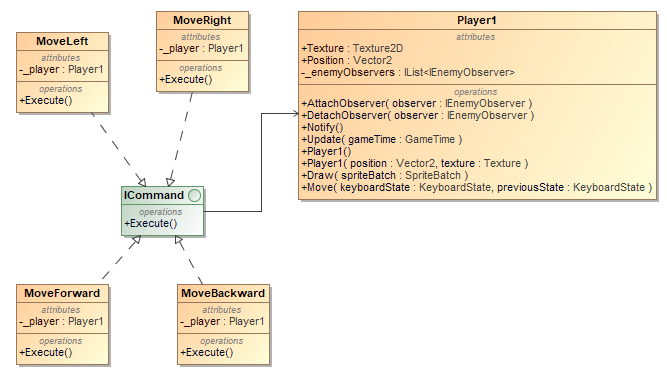
}

### 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);

}

}

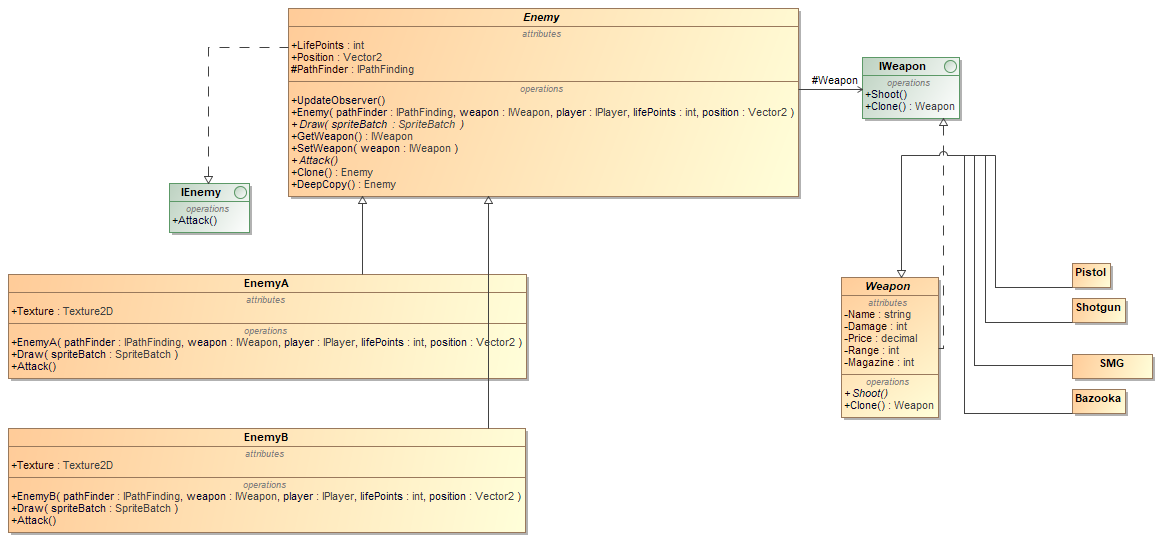
}

### 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();

}

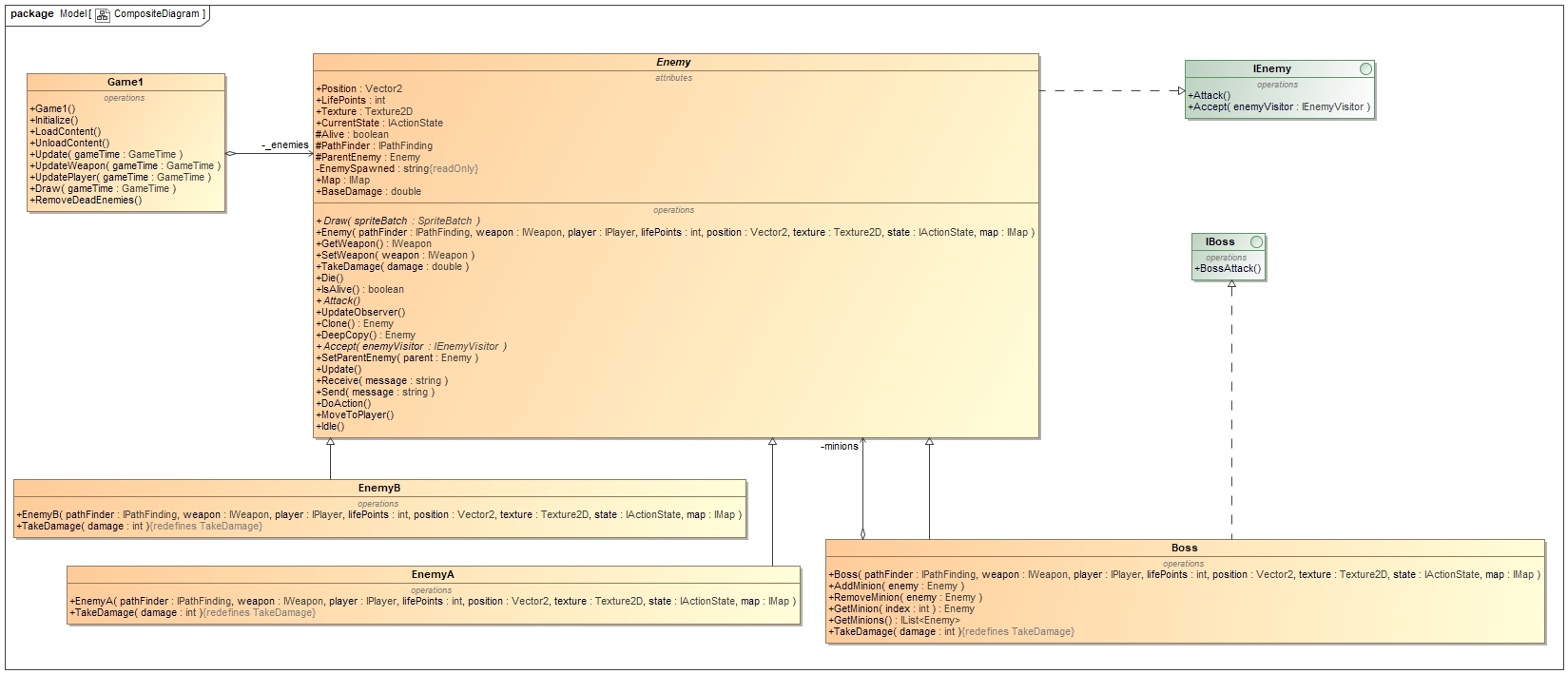
}

### Naudojimo pagrindimas

Bridge 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.

## Composite

### Klasių diagrama



### Esminis kodas

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Shooter.Interfaces;

using Shooter.PatternClasses;

namespace Shooter.Classes

{

public class EnemyA : Enemy

{

public EnemyA(IPathFinding pathFinder, IWeapon weapon, IPlayer player, int lifePoints, Vector2 position, Texture2D texture) : base(pathFinder, weapon, player, lifePoints, position, texture)

{

}

public override void Draw(SpriteBatch spriteBatch)

{

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

}

public override void Attack()

{

Weapon.Shoot();

}

public override void Accept(IEnemyVisitor enemyVisitor)

{

enemyVisitor.Visit(this);

}

}

}

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Shooter.Interfaces;

using Shooter.PatternClasses;

using System.Collections.Generic;

namespace Shooter.Classes

{

public class Boss : Enemy, IBoss

{

private IList<Enemy> minions;

public Boss(IPathFinding pathFinder, IWeapon weapon, IPlayer player, int lifePoints, Vector2 position, Texture2D texture) : base(pathFinder, weapon, player, lifePoints, position, texture)

{

minions = new List<Enemy>();

}

public override void Draw(SpriteBatch spriteBatch)

{

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

}

public override void Attack()

{

}

public void BossAttack()

{

}

public void AddMinion(Enemy enemy)

{

minions.Add(enemy);

}

public void RemoveMinion(Enemy enemy)

{

minions.Remove(enemy);

}

public override void Die()

{

Alive = false;

}

public Enemy GetMinion(int index)

{

return minions[index];

}

public IList<Enemy> GetMinions()

{

return minions;

}

public override void Accept(IEnemyVisitor enemyVisitor)

{

foreach(Enemy minion in minions)

{

minion.Accept(enemyVisitor);

}

enemyVisitor.Visit(this);

}

}

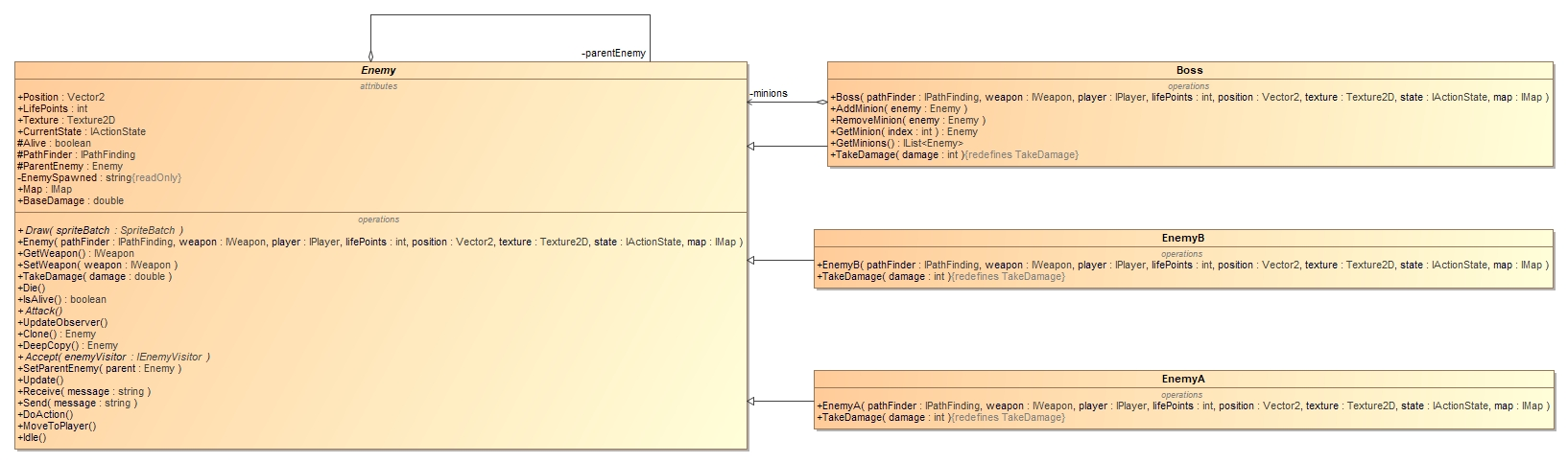
}

### Naudojimo pagrindimas

Composite projektavimo šablonas naudingas tada, kai norima sudaryti objektų hierarchinę struktūrą ir tiek su vienu objektu, tiek su visa struktūra elgtis vienodai. Šis šablonas projektiniame darbe buvo naudingas sudarant žaidimo bosų ir jų pakalikų objektų hierarchinę struktūrą: bosas gali turėti daug pakalikų, tarp kurių gali būti ir kiti bosai, kurie turi savo pakalikus ir t.t.

## Chain of Responsibility

### Klasių diagrama



### Esminis kodas

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

{

public double LifePoints { get; set; }

protected Enemy \_parentEnemy;

public void SetParentEnemy(Enemy parent)

{

\_parentEnemy = parent;

}

public void TakeDamage(double damage)

{

\_parentEnemy?.TakeDamage(damage/2);

LifePoints = LifePoints - damage;

if(LifePoints <= 0)

{

Alive = false;

}

}

}

public class EnemyA : Enemy

{

public override void TakeDamage(double damage)

{

ParentEnemy?.TakeDamage(damage / 3);

LifePoints = LifePoints - damage;

if (LifePoints <= 0)

{

Alive = false;

}

}

}

public class EnemyB : Enemy

{

public override void TakeDamage(double damage)

{

ParentEnemy?.TakeDamage(damage);

LifePoints = LifePoints - damage;

if (LifePoints <= 0)

{

Alive = false;

}

}

}

public class Boss : Enemy, IBoss

{

public override void TakeDamage(double damage)

{

ParentEnemy?.TakeDamage(damage/5);

LifePoints = LifePoints - damage;

if (LifePoints <= 0)

{

Alive = false;

}

}

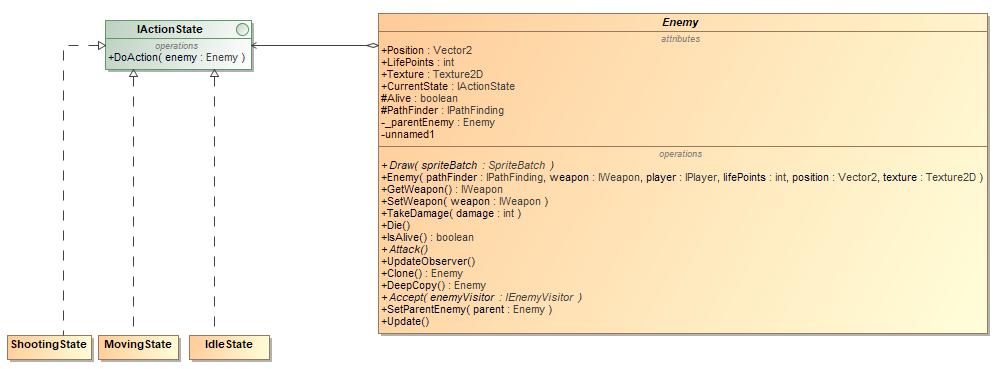
}

### Naudojimo paskirtis

Žaidime priešai gali turėti savo priešų pakalikų. Pakalikams netenkant gyvybės taškų dėl to, kad jie yra puolami, norima, kad pakalikų tėvas taip pat prarastų gyvybės taškų. Šioje situacijoje puikiai tinka pritaikyti Chain of Responsibility šabloną. Kiekvienas priešas saugosi savo tėvą, o kai priima žalą, tai dalį žalos perduoda savo tėvui. Vėliau tas tėvas taip pat jeigu turi tėvą, perduoda žalą ir jam. Neturint tėvo, žala nėra perduodama.

## State

### Klasių diagrama



### Esminis kodas

public interface IActionState

{

void DoAction(Enemy enemy);

}

class ShootingState : IActionState

{

public void DoAction(Enemy enemy)

{

Console.WriteLine("Shoots");

enemy.Attack();

}

}

public class MovingState : IActionState

{

public void DoAction(Enemy enemy)

{

Console.WriteLine("Moves");

enemy.MoveToPlayer();

}

}

public class IdleState : IActionState

{

public void DoAction(Enemy enemy)

{

// Do nothing

}

}

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

{

public IActionState CurrentState { get; set; }

public abstract void Attack();

public void DoAction()

{

CurrentState.DoAction(this);

}

public void MoveToPlayer()

{

var start = new Point((int)Position.X, (int)Position.Y);

var end = new Point((int)\_player.Position.X, (int)\_player.Position.Y);

var nextPoint = PathFinder.NextPoint(start, end);

var newPosition = new Vector2(nextPoint.X, nextPoint.Y);

if (newPosition != \_player.Position)

{

Position = newPosition;

}

}

public virtual void UpdateObserver()

{

DoAction();

Logger.Instance.Info($"Enemy notified of player position {\_player.Position}");

}

public virtual void Idle()

{

// Do nothing

}

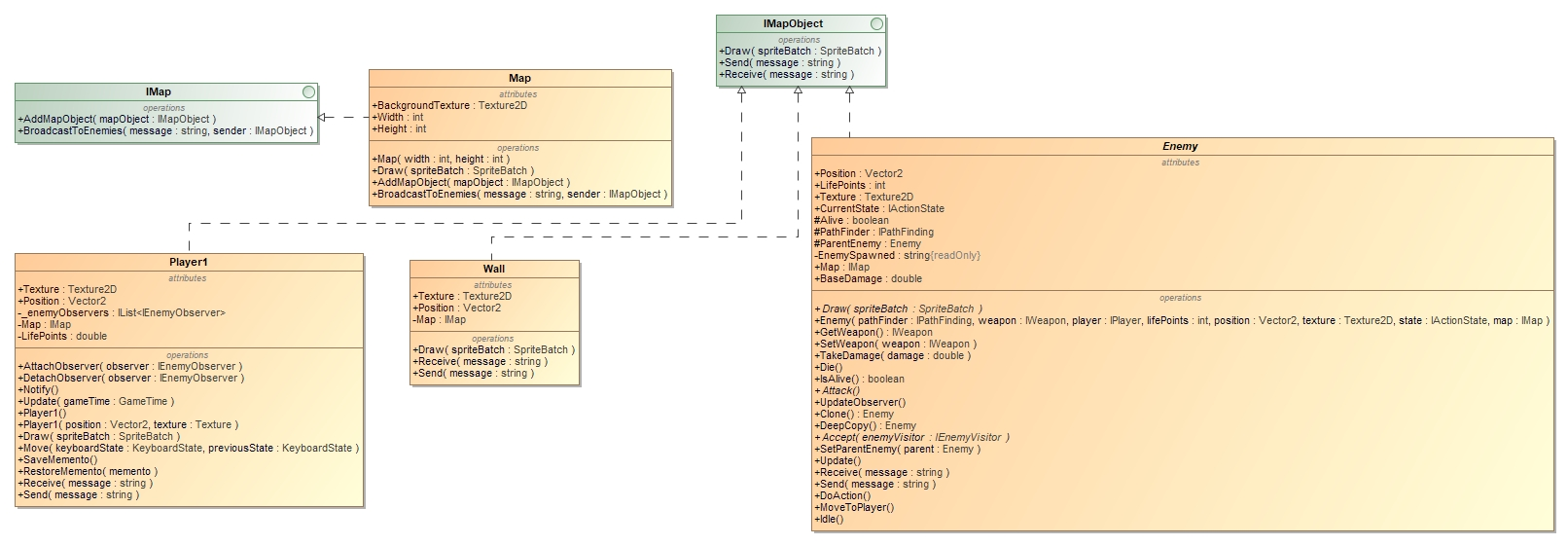
}

### Naudojimo paskirtis

Priešas vienu metu gali atlikti keletą skirtingų veiksmų: pajudėti, pulti arba nieko nedaryti. Šie veiksmai atliekami tik po vieną, priklausant kokioje būsenoje yra priešas: ar jis mato žaidėją, ar nemato. Šiuo atveju tinka pritaikyti State šabloną. Kiekvienam priešo veiksmui sukuriama būsenos klasė. Ėjimo metu, priešas tiesiog vykdo tam tikros būsenos veiksmo metodą.

## Mediator

### Klasių diagrama



### Esminis kodas

public interface IMapObject

{

IMap Map{ get; set; }

void Receive(string message);

void Send(string message);

}

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

{

public IMap Map { get; set; }

public void Receive(string message)

{

if (message == EnemySpawned)

{

Logger.Instance.Info("Enemy spawned, base damage increases");

BaseDamage += BaseDamage \* 0.2;

}

}

public void Send(string message)

{

Map?.BroadcastToEnemies(message, this);

}

protected Enemy(IPathFinding pathFinder, IWeapon weapon, IPlayer player, double lifePoints, Vector2 position, Texture2D texture, IActionState state, IMap map)

{

PathFinder = pathFinder;

Weapon = weapon;

LifePoints = lifePoints;

Position = position;

Texture = texture;

CurrentState = state;

\_player = player;

Map = map;

\_player.AttachObserver(this);

Alive = true;

Send(EnemySpawned);

}

}

public class Map : IMap

{

public IList<IMapObject> MapObjects { get; set; }

public void AddMapObject(IMapObject mapObject)

{

MapObjects.Add(mapObject);

mapObject.Map = this;

}

public void BroadcastToEnemies(string message, IMapObject sender)

{

foreach (var mapObject in MapObjects)

{

if (sender != mapObject && mapObject is Enemy)

{

mapObject.Receive(message);

}

}

}

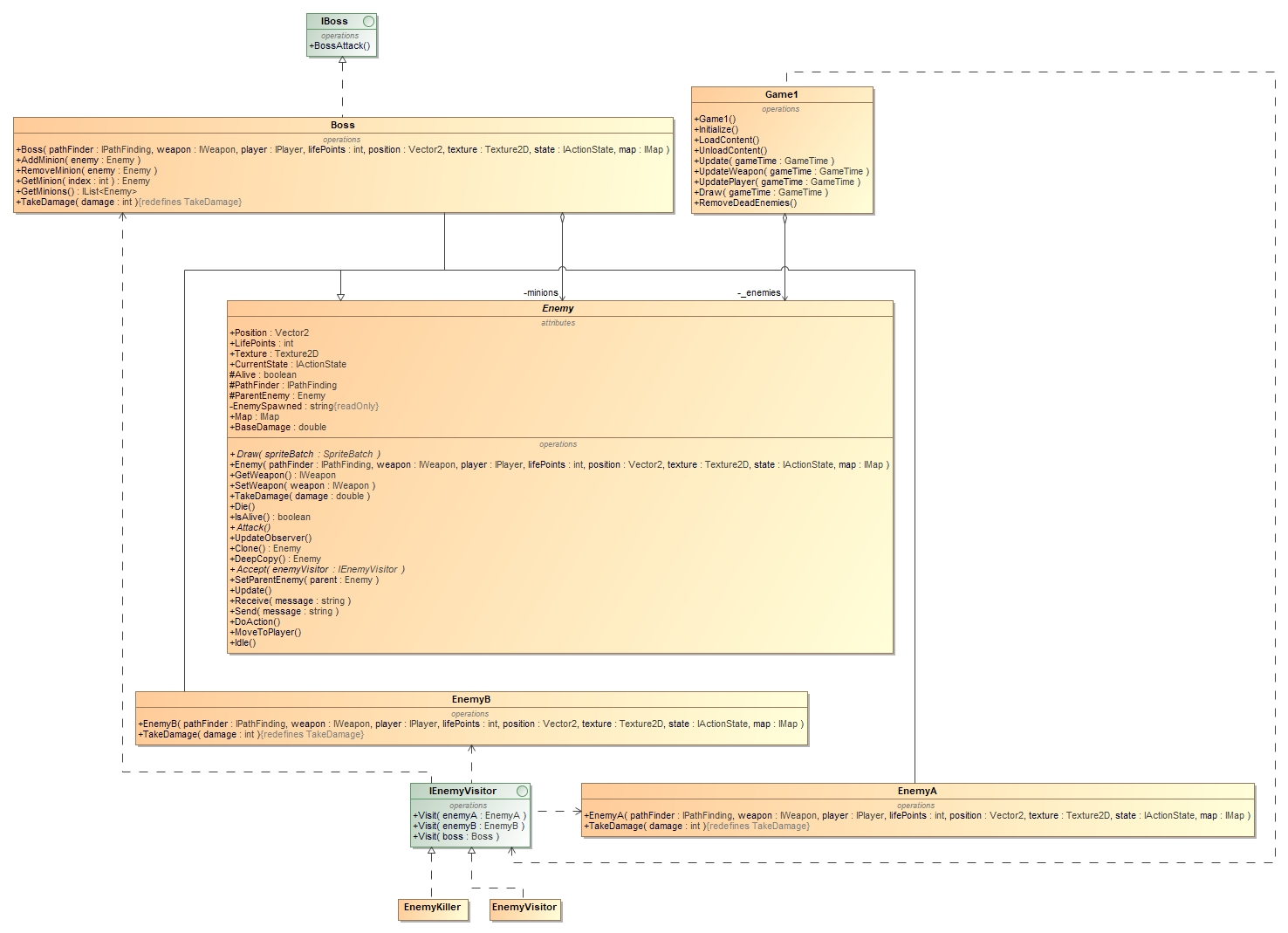
}

### Naudojimo paskirtis

Kai žaidime atsiranda priešas, norima, kad šiuo metu dar gyvi priešai sustiprėtų, t.y. darytų daugiau žalos žaidėjui. Šioje situacijoje puikiai tinka pritaikyti Mediator šabloną. Susikūręs priešas išsiunčia pranešimą mediatoriui „Map“, kuris visiems žaidime esantiems gyviems priešams nusiunčia žinutę, kad susikūrė naujas priešas. Žinutę gavusiems priešams padidėja žalos darymo taškai.

## Visitor

### Klasių diagrama



### Esminis kodas

using Shooter.Classes;

namespace Shooter.PatternClasses

{

public class EnemyKiller : IEnemyVisitor

{

public void Visit(EnemyA enemyA)

{

enemyA.Die();

}

public void Visit(EnemyB enemyB)

{

enemyB.Die();

}

public void Visit(Boss boss)

{

boss.Die();

}

}

}

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Graphics;

using Shooter.Interfaces;

using Shooter.PatternClasses;

namespace Shooter.Classes

{

public class EnemyB : Enemy

{

public EnemyB(IPathFinding pathFinder, IWeapon weapon, IPlayer player, int lifePoints, Vector2 position, Texture2D texture) : base(pathFinder, weapon, player, lifePoints, position, texture)

{

}

public override void Draw(SpriteBatch spriteBatch)

{

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

}

public override void Attack()

{

Weapon.Shoot();

}

public override void Accept(IEnemyVisitor enemyVisitor)

{

enemyVisitor.Visit(this);

}

}

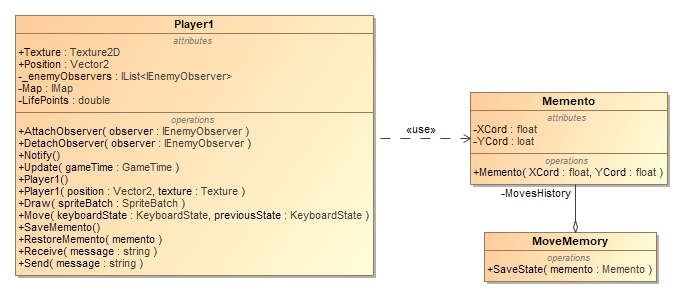
}

### Naudojimo pagrindimas

Visitor projektavimo šablonas suderinamas su Composite šablonu, kadangi viena iš jo paskirčių yra suteikti papildomą funkcionalumą objektams hierarchinėje struktūroje, nekeičiant pačios struktūros klasių vidaus. Tačiau įgyvendintas Visitor šablonas pašalina naudojamų klasių inkapsuliaciją, todėl tenka apsispręsti, ar tokia kaina verta, taikant šį šabloną. Visitor šablonas projektiniame darbe buvo panaudotas žaidimo bosų ir jų pakalikų objektų hierarchijų žudymui. Kadangi Visitor leidžia įvykdyti nurodytus veiksmus visai nurodyto objekto hierarchijai, tai palengvino bosų ir jų pakalikų pašalinimą iš žaidimo, nes jei miršta bosas, visi jo pakalikai taip pat turi mirti.

## Memento

### Klasių diagrama



### Esminis kodas

using System.Collections.Generic;

namespace Shooter.PatternClasses

{

public static class MoveMemory

{

public static List<Memento> MovesHistory = new List<Memento>();

public static void SaveState(Memento memento)

{

MovesHistory.Add(memento);

}

}

}

namespace Shooter.PatternClasses

{

public class Memento

{

public float XCord { get; set; }

public float YCord { get; set; }

public Memento(float XCord, float YCord)

{

this.XCord = XCord;

this.YCord = YCord;

}

}

}

public Memento SaveMemento()

{

return new Memento(Position.X, Position.Y);

}

public void RestoreMemento(Memento previousState)

{

Position = new Vector2(previousState.XCord + GameSettings.TileSize, previousState.YCord);

}

if (keyboardState.IsKeyDown(Keys.Space))

{

var moves = MoveMemory.MovesHistory;

if (moves.Count > 4)

{

\_player.RestoreMemento(moves[moves.Count - 4]);

MoveMemory.MovesHistory.Clear();

}

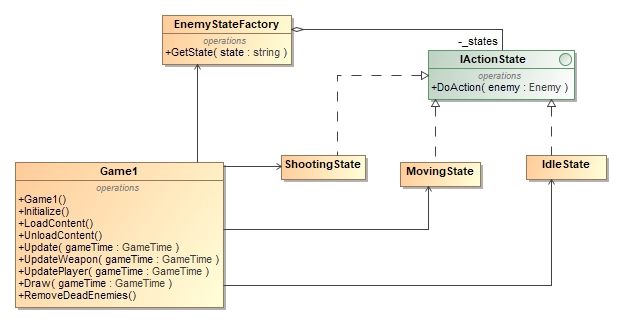
}

### Naudojimo pagrindimas

Memento projektavimo šablonas pritaikytas žaidėjo specialiai savybei realizuoti, kuri leidžia žaidėjui nusikelti į jo buvimo vietą prieš 4 ėjimus. Kiekvieno ėjimo metu yra išsaugoma žaidėjo buvimo vieta.

## Flyweight

### Klasių diagrama



### Esminis kodas

using Shooter.Classes;

using System.Collections.Generic;

namespace Shooter.PatternClasses

{

public class EnemyStateFactory

{

private Dictionary<string, IActionState> \_states = new Dictionary<string, IActionState>();

public IActionState GetState(string state)

{

IActionState actionState = null;

if (\_states.ContainsKey(state))

{

actionState = \_states[state];

}

else

{

switch (state)

{

case "Moving":

actionState = new MovingState();

break;

case "Idle":

actionState = new IdleState();

break;

case "Shooting":

actionState = new ShootingState();

break;

}

\_states.Add(state, actionState);

}

return actionState;

}

}

}

var enemiesFactory = new EnemiesConcreteFactory();

\_enemies = new List<Enemy>

{

enemiesFactory.CreateEnemy(\_pathFinder, EnemyType.Small, new Bazooka(), \_player, 100, new Vector2(5\*GameSettings.TileSize, 5\*GameSettings.TileSize), \_enemyATexture, enemyState.GetState("Moving")),

enemiesFactory.CreateEnemy(\_pathFinder, EnemyType.Big, new Pistol(), \_player, 100, new Vector2(3\*GameSettings.TileSize, 5\*GameSettings.TileSize), \_enemyBTexture, enemyState.GetState("Moving")),

enemiesFactory.CreateEnemy(\_pathFinder, EnemyType.Boss, new Pistol(), \_player, 500, new Vector2(1 \* GameSettings.TileSize, 1 \* GameSettings.TileSize), \_bossTexture, enemyState.GetState("Moving")),

enemiesFactory.CreateEnemy(\_pathFinder, EnemyType.Boss, new Pistol(), \_player, 250, new Vector2(1 \* GameSettings.TileSize, 2 \* GameSettings.TileSize), \_bossTexture, enemyState.GetState("Moving") };

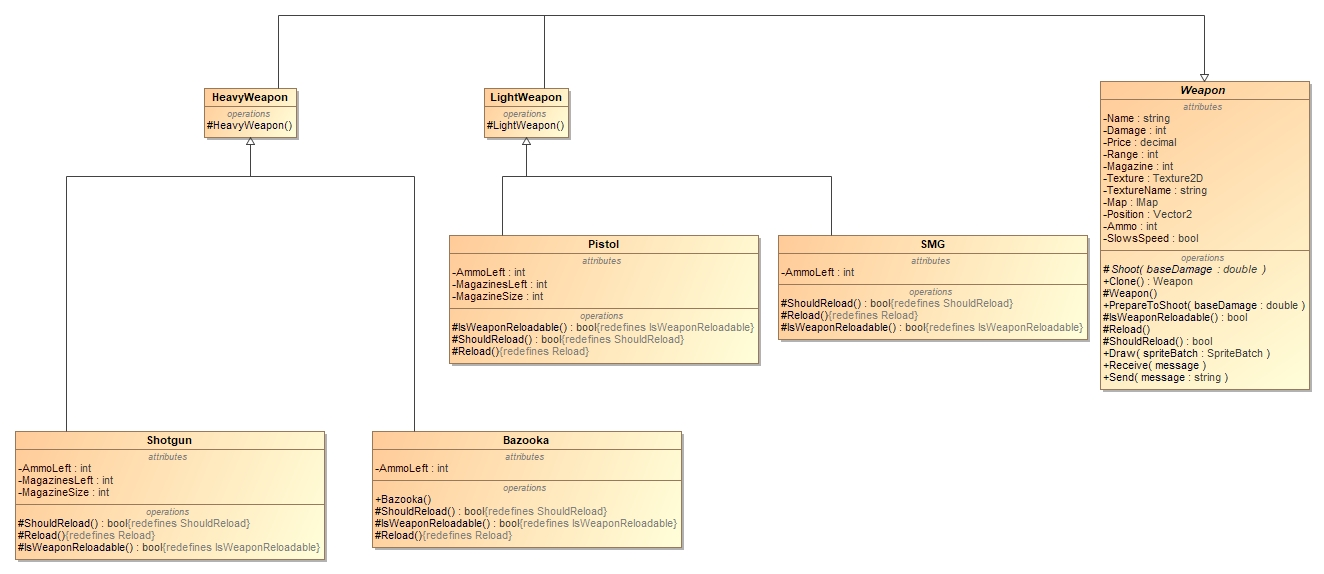
};

### Naudojimo pagrindimas

Flyweight projektavimo šablonas buvo naudojamas Enemy būsenos gavimui. Kadangi naujos būsenos kūrimas gali pasitaikyti dažnai, šis projektavimo šablonas didina sistemos greitaveiką.

## Template method

### Klasių diagrama



### Esminis kodas

public void PrepareToShoot(double baseDamage)

{

if (IsWeaponReloadable())

{

if (ShouldReload())

{

Reload();

}

else

{

Shoot(baseDamage);

}

}

else

{

Shoot(baseDamage);

}

}

protected abstract bool IsWeaponReloadable();

protected abstract void Reload();

protected abstract bool ShouldReload();

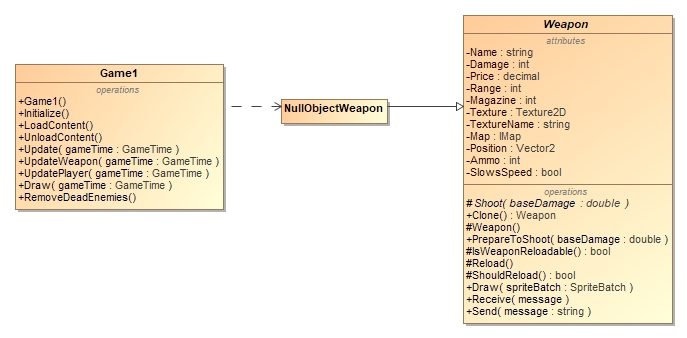
protected abstract void Shoot(double baseDamage);

### Naudojimo pagrindimas

Template method projektavimo šablonas naudojamas siekiant supaprastinti kiekvieno ginklo užtaisymo logiką, kadangi kiekvienam ginklui visi veiksmai vykdomi vienoda tvarka, tačiau skiriasi kiekvieno ginklo apkabos dydis.

## Null Object

### Klasių diagrama



### Esminis kodas

public class NullObjectWeapon : Weapon

{

protected override bool IsWeaponReloadable()

{

return false;

}

protected override void Reload()

{

}

protected override bool ShouldReload()

{

return false;

}

protected override void Shoot(double baseDamage)

{

}

}

protected void UpdateWeapon(GameTime gameTime)

{

var val = StaticRandom.Instance.Next(0, 1000);

if (val == 0)

{

Weapon weapon = new NullObjectWeapon();

var factory = WeaponFactory.CreateFactory((WeaponType)StaticRandom.Instance.Next(0, 2));

switch (factory)

{

case HeavyWeaponFactory h:

var num = StaticRandom.Instance.Next(0, 2);

weapon = factory.CreateWeapon((WeaponName)num);

break;

case LightWeaponFactory l:

var num2 = StaticRandom.Instance.Next(2, 4);

weapon = factory.CreateWeapon((WeaponName)num2);

break;

}

\_weapons.Add(weapon);

weapon.Texture = \_weaponTextures[weapon.Name];

\_map.MapObjects.Add(weapon);

}

}

### Naudojimo pagrindimas

Null Object projektavimo šabloną panaudojome kodo vietoje, kurioje buvo galima Null Reference Exception klaida, nenaudojant šio šablono. Jį pritaikę šios problemos išvengiame: vietoj null reiškmės, kuri gaunama nerandant nurodyto tipo ginklo, gauname NullObjectWeapon tipo objektą.

# Išvados

Projektinio darbo metu buvo susipažinta su įvairiais projektavimo šablonais. Taip pat buvo mokomasi realiai pritaikyti dalį šių šablonų, kuriant žaidimą. Pirmoje darbo pusėje buvo įgyvendinti Singleton (įvykių registratoriui), Adapter (kelio paieškos bibliotekos pritaikymui), Factory (priešų kūrimui), Abstract Factory (skirtingų tipų ginklų kūrimui), Prototype (priešų klonavimui), Observer (žaidėjo buvimo vietos nurodymui priešams), Command (žaidėjo valdymui) ir Bridge (ryšio tarp priešo ir ginklo suabstraktinimui) šablonai. Antroje darbo pusėje projektas buvo toliau plėtojamas, pritaikant daugiau projektavimo šablonų. Buvo įgyvendinti Composite (žaidimo bosų ir jų pakalikų hierarchinei struktūrai sukurti), Visitor (žaidimo bosų ir jų pakalikų hierarchinių struktūrų funkcionalumo praplėtimui) šablonai.