

Informatikos fakultetas

**T120B516 Objektinis programų projektavimas**

**Laboratorinių darbų ataskaita**

**Žaidimas „Tower Defence“**

|  |  |  |
| --- | --- | --- |
|  | Studentai: | Mangirdas Kazlauskas, IFF-4/1 Simonas Baltulionis, IFF-4/1  Tautvydas Traška, IFF-4/2 |
| Dėstytojai: | Andrej Ušaniov  Dominykas Barisas |

Kaunas 2017

Turinys

[1. Projekto aprašymas 5](#_Toc500467626)

[2. Laboratorinis darbas Nr. 1 (1 dalis) 6](#_Toc500467627)

[2.1. Pradinė klasių diagrama 6](#_Toc500467628)

[2.2. Vienintelio objekto (Singleton) projektavimo pavyzdys 7](#_Toc500467629)

[2.2.1. Projektavimo šablono realizacija 7](#_Toc500467630)

[2.2.2. Projektavimo šablono naudojimo pagrindimas 8](#_Toc500467631)

[2.3. Abstraktaus fabriko (Abstract factory) projektavimo pavyzdys (Simonas) 8](#_Toc500467632)

[2.3.1. Projektavimo šablono realizacija 8](#_Toc500467633)

[2.3.2. Projektavimo šablono naudojimo pagrindimas 10](#_Toc500467634)

[2.4. Fabriko metodo (Factory method) projektavimo pavyzdys (Mangirdas) 10](#_Toc500467635)

[2.4.1. Projektavimo šablono realizacija 10](#_Toc500467636)

[2.4.2. Projektavimo šablono naudojimo pagrindimas 12](#_Toc500467637)

[2.5. Strategijos (Strategy) projektavimo pavyzdys (Tautvydas) 12](#_Toc500467638)

[2.5.1. Projektavimo šablono realizacija 12](#_Toc500467639)

[2.5.2. Projektavimo šablono naudojimo pagrindimas 14](#_Toc500467640)

[3. Laboratoris darbas Nr. 1 (2 dalis) 15](#_Toc500467641)

[3.1. Klasių diagrama 15](#_Toc500467642)

[3.2. Pritaikančio objekto (Adapter) projektavimo pavyzdys 16](#_Toc500467643)

[3.2.1. Projektavimo šablono realizacija 16](#_Toc500467644)

[3.2.2. Projektavimo šablono naudojimo pagrindimas 17](#_Toc500467645)

[3.3. Dekoratoriaus (Decorator) projektavimo pavyzdys (Simonas) 18](#_Toc500467646)

[3.3.1. Projektavimo šablono realizacija 18](#_Toc500467647)

[3.3.2. Projektavimo šablono naudojimo pagrindimas 19](#_Toc500467648)

[3.4. Komandos (Command) projektavimo pavyzdys (Mangirdas) 19](#_Toc500467649)

[3.4.1. Projektavimo šablono realizacija 19](#_Toc500467650)

[3.4.2. Projektavimo šablono naudojimo pagrindimas 20](#_Toc500467651)

[3.5. Tilto (Bridge) projektavimo pavyzdys (Tautvydas) 20](#_Toc500467652)

[3.5.1. Projektavimo šablono realizacija 20](#_Toc500467653)

[3.5.2. Projektavimo šablono naudojimo pagrindimas 23](#_Toc500467654)

[4. Laboratorinis darbas nr. 2 (1 dalis) 24](#_Toc500467655)

[4.1. Template method projektavimo pavyzdys 24](#_Toc500467656)

[4.1.1. Klasių diagrama 24](#_Toc500467657)

[4.1.2. Projektavimo šablono realizacija 24](#_Toc500467658)

[4.1.3. Projektavimo šablono naudojimo pagrindimas 25](#_Toc500467659)

[4.2. Flyweight projektavimo pavyzdys (Tautvydas) 26](#_Toc500467660)

[4.2.1. Klasių diagrama 26](#_Toc500467661)

[4.2.2. Projektavimo šablono realizacija 26](#_Toc500467662)

[4.2.3. Projektavimo šablono naudojimo pagrindimas 27](#_Toc500467663)

[4.3. State projektavimo pavyzdys 28](#_Toc500467664)

[4.3.1. Klasių diagrama 28](#_Toc500467665)

[4.3.2. Projektavimo šablono realizacija 28](#_Toc500467666)

[4.3.3. Projektavimo šablono naudojimo pagrindimas 30](#_Toc500467667)

[4.4. Proxy projektavimo pavyzdys 31](#_Toc500467668)

[4.4.1. Klasių diagrama 31](#_Toc500467669)

[4.4.2. Projektavimo šablono realizacija 31](#_Toc500467670)

[4.4.3. Projektavimo šablono naudojimo pagrindimas 32](#_Toc500467671)

[5. Laboratorinis darbas nr. 2 (2 dalis) 33](#_Toc500467672)

[5.1. Chain of responsibility projektavimo pavyzdys 33](#_Toc500467673)

[5.1.1. Klasių diagrama 33](#_Toc500467674)

[5.1.2. Projektavimo šablono realizacija 33](#_Toc500467675)

[5.1.3. Projektavimo šablono naudojimo pagrindimas 35](#_Toc500467676)

[5.2. Null object projektavimo pavyzdys 36](#_Toc500467677)

[5.2.1. Klasių diagrama 36](#_Toc500467678)

[5.2.2. Projektavimo šablono realizacija 36](#_Toc500467679)

[5.2.3. Projektavimo šablono naudojimo pagrindimas 37](#_Toc500467680)

[5.3. Mediator projektavimo pavyzdys 38](#_Toc500467681)

[5.3.1. Klasių diagrama 38](#_Toc500467682)

[5.3.2. Projektavimo šablono realizacija 38](#_Toc500467683)

[5.3.3. Projektavimo šablono naudojimo pagrindimas 38](#_Toc500467684)

[5.4. Memento projektavimo pavyzdys 39](#_Toc500467685)

[5.4.1. Klasių diagrama 39](#_Toc500467686)

[5.4.2. Projektavimo šablono realizacija 39](#_Toc500467687)

[5.4.3. Projektavimo šablono naudojimo pagrindimas 40](#_Toc500467688)

[5.5. Visitor projektavimo pavyzdys 41](#_Toc500467689)

[5.5.1. Klasių diagrama 41](#_Toc500467690)

[5.5.2. Projektavimo šablono realizacija 41](#_Toc500467691)

[5.5.3. Projektavimo šablono naudojimo pagrindimas 43](#_Toc500467692)

# Projekto aprašymas

Laboratorinių darbų metu kuriamas žaidimas „Tower Defence“. „Tower Defence“ – tai vienas iš strateginių vaizdo žaidimų žanrų, kurio tikslas – apginti tam tikrą teritoriją nuo vis atsirandančių priešų. Gynybai nuo priešų statomi įvairių tipų pastatai – gynybinės struktūros, kurios atakuoja priešus, judančius tam tikru keliu iki turimos apginti teritorijos. Taip pat žaidime egzistuoja įvairių tipų priešai su skirtingais gyvybiniais parametrais, pvz., gyvybės taškais. Vienas iš tokio tipo žaidimų pavyzdžių pateiktas 1 paveikslėlyje.

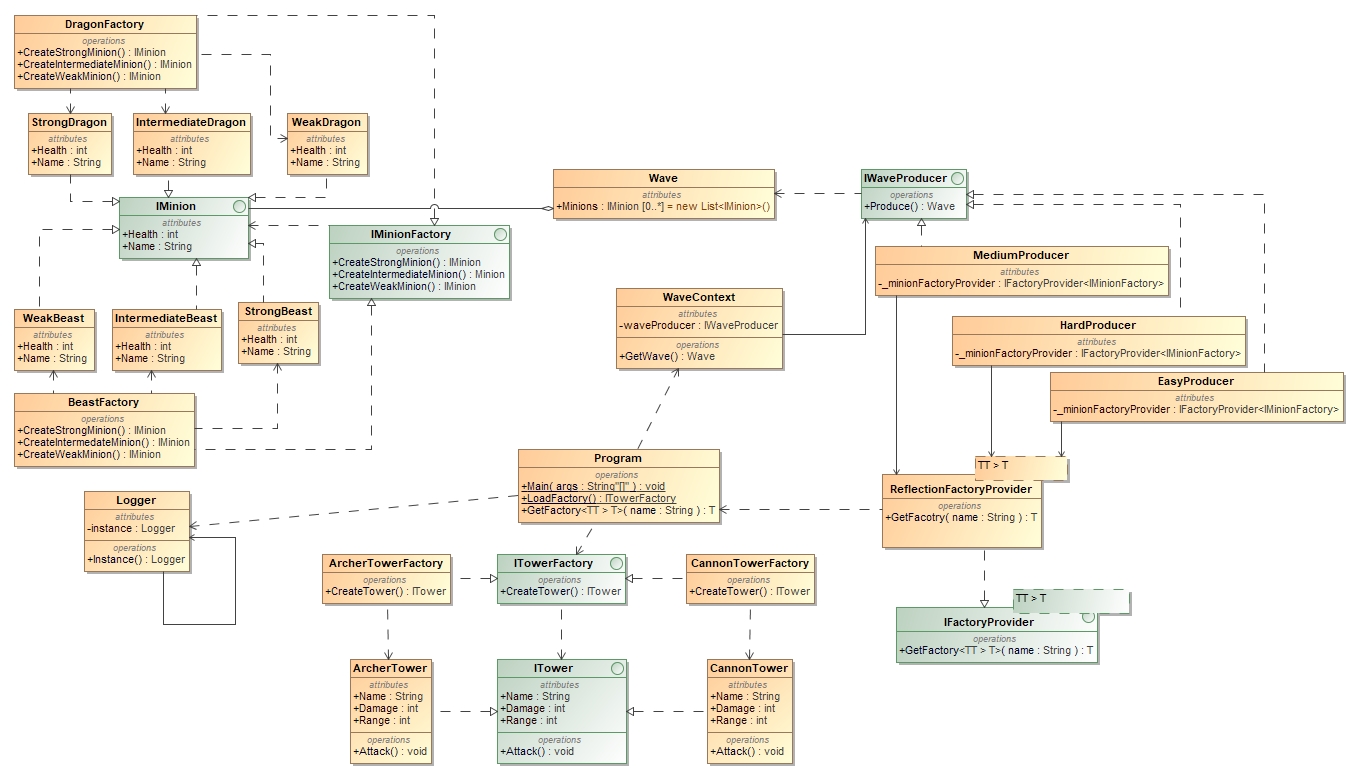


1 pav. Tower Defence tipo žaidimo pavyzdys (šaltinis: https://i.pinimg.com/originals/02/f2/3f/02f23faf46a076f590a4f4167f365e4c.jpg)

Tokio tipo žaidimą nutarta kurti dėl to, kad jį labai lengva plėtoti, skirtingi pastatų bei priešų tipai suteikia daug laisvės įvairių programinių struktūrų realizavimui bei dizaino šablonų pritaikymui.

# Laboratorinis darbas Nr. 1 (1 dalis)

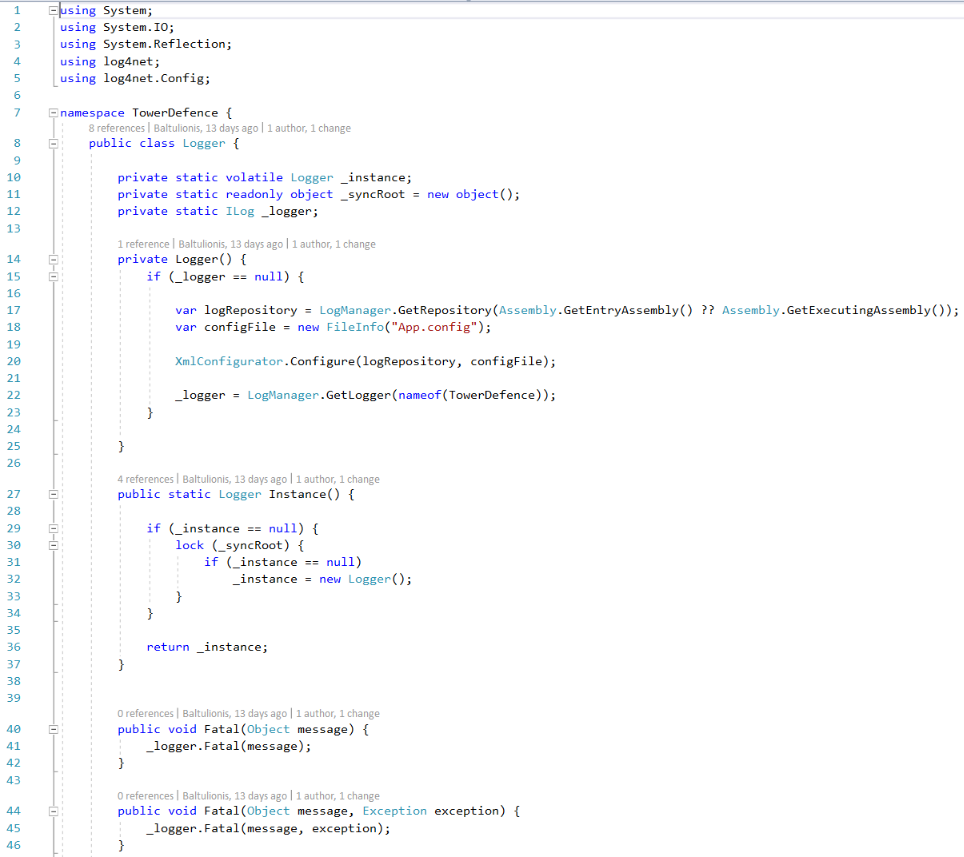
## Pradinė klasių diagrama

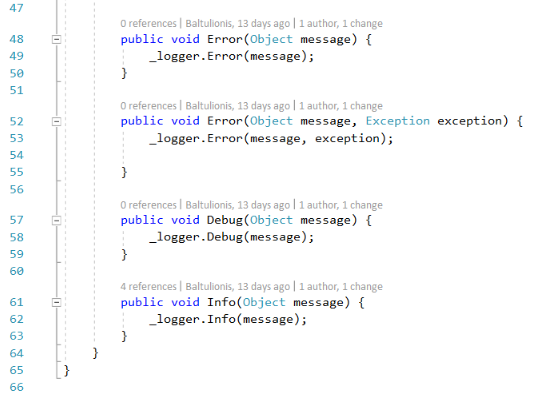
2 paveikslėlyje pateikta pradinė kuriamo žaidimo klasių diagrama.

2 pav. Pradinė klasių diagrama

## Vienintelio objekto (Singleton) projektavimo pavyzdys

### Projektavimo šablono realizacija





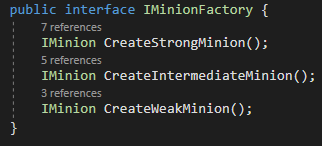
3 pav. Vienintelio objekto (Singleton) realizacija

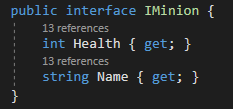
### Projektavimo šablono naudojimo pagrindimas

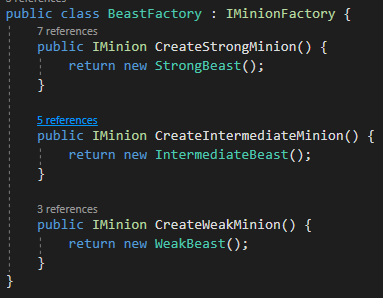
Vienintelio objekto (Singleton) projektavimo šablonas yra naudojamas tada, kai kuri nors klasė projekte yra pastoviai naudojama ir pakartotinai reikia sukurti tos klasės objektą, kas kainuoja resursų. Mūsų atveju tokia klasė – Logger klasė, kuri skirta programos pranešimų išvedimui. Kadangi ši klasė turi daug metodų, savyje talpina nemažai logikos, todėl tokio objekto sukūrimas kas kartą kainuotų nemažai resursų, todėl tikslingai panaudotas Singleton šablonas šią problemą išsprendžia.

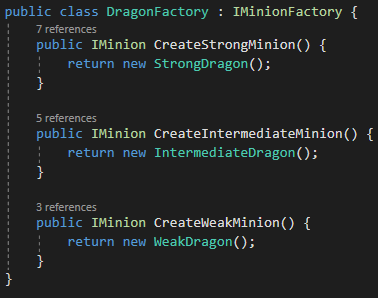
## Abstraktaus fabriko (Abstract factory) projektavimo pavyzdys (Simonas)

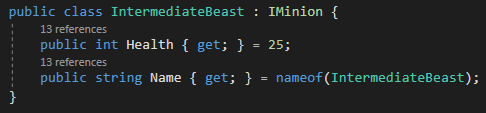
### Projektavimo šablono realizacija

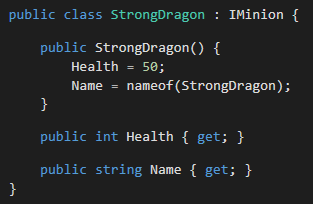












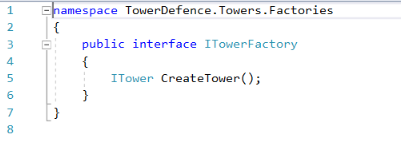
4 pav. Abstraktaus fabriko (Abstract factory) realizacija

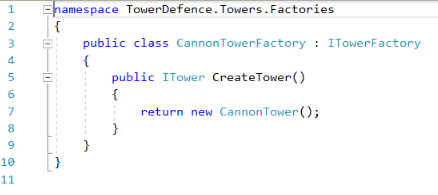
### Projektavimo šablono naudojimo pagrindimas

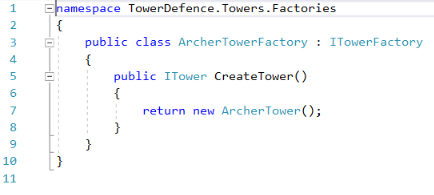
„Abstract factory“ šablonas yra skirtas sukurti objektų šeimoms. Mūsų atveju šablonas naudojamas sukurti priešų objektų šeimoms – galima sukurti trijų tipų žvėris arba tų pačių trijų tipų drakonus. Šie abstraktūs fabrikai gali būti ir bus panaudojami kuriant įvairaus tipo priešų bangas skirtinguose lygiuose.

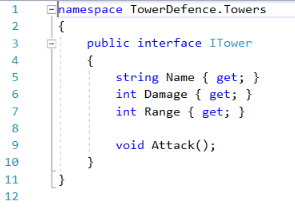
## Fabriko metodo (Factory method) projektavimo pavyzdys (Mangirdas)

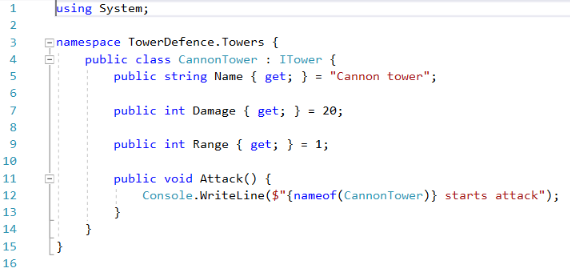
### Projektavimo šablono realizacija

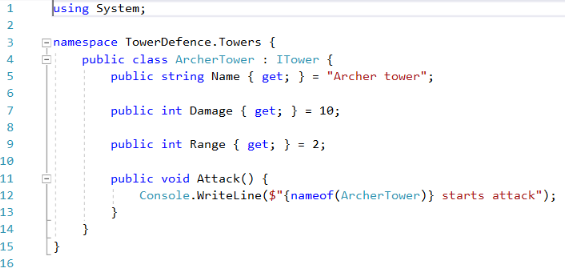


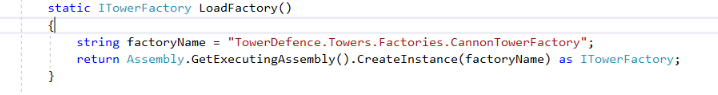












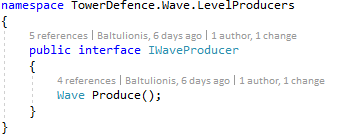
5 pav. Fabriko metodo (Factory method) realizacija

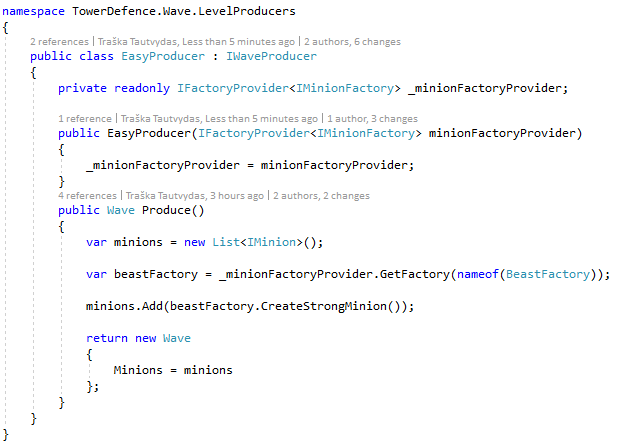
### Projektavimo šablono naudojimo pagrindimas

Fabriko metodo (Factory method) projektavimo šablono paskirtis – palengvinti objektų sukūrimą, viduje paslepiant pačią objekto sukūrimo logiką, o vietoj to naudojant sukurtą sąsają. Taip pat šablonas padeda sukurti objektus, tiksliai neapibrėžiant jų klasės – remiantis šia savybe, buvo sukurtos dvi sąsajos: gynybinių pastatų sąsaja ITower, bei gynybinių pastatų objektų sukūrimo sąsaja ITowerFactory. Naudojantis ITowerFactory sąsaja, galima lengvai sukurti skirtingų tipų fabrikus, kuriančius skirtingų ITower tipų pastatus. Toks projektavimo sprendimas leidžia lengvai plėsti galimų sukurti skirtingų tipų pastatų sąrašą, neapsunkinant ir nekeičiant jau esamų bei naujų pastatų objektų sukūrimo metodų.

## Strategijos (Strategy) projektavimo pavyzdys (Tautvydas)

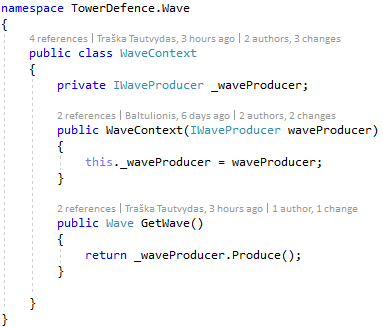
### Projektavimo šablono realizacija











6 pav. Strategijos (Strategy) realizacija

### Projektavimo šablono naudojimo pagrindimas

„Strategy“ šablonas yra skirtas keisti objekto elgseną ar algoritmą programos vykdymo metu. Mūsų atveju šis šablonas naudojamas siekiant parinkti priešų bangų sudėtingumą priklausomai nuo žaidimo lygio. Atsakinga priešų bangos konteksto klasė, objekto kūrimo metu gali priimti skirtingas strategijų implementacijas, lemiančias žaidimo įvairovę ir sudėtingumą.

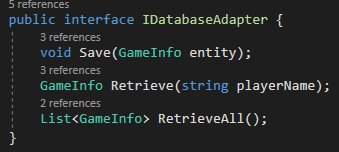
# Laboratoris darbas Nr. 1 (2 dalis)

## C:\Users\mangi\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ClassDiagram.jpgKlasių diagrama

7 pav. 1 laboratorinio darbo klasių diagrama po antrosios dalies atlikimo

## Pritaikančio objekto (Adapter) projektavimo pavyzdys

### Projektavimo šablono realizacija





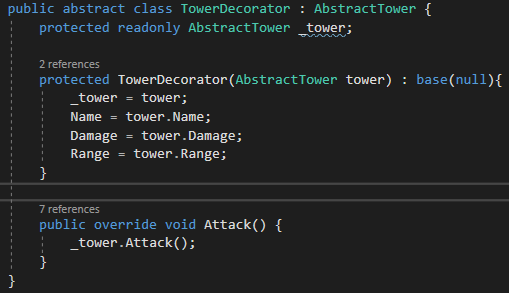
8 pav. Dekoratoriaus (Decorator) realizacija

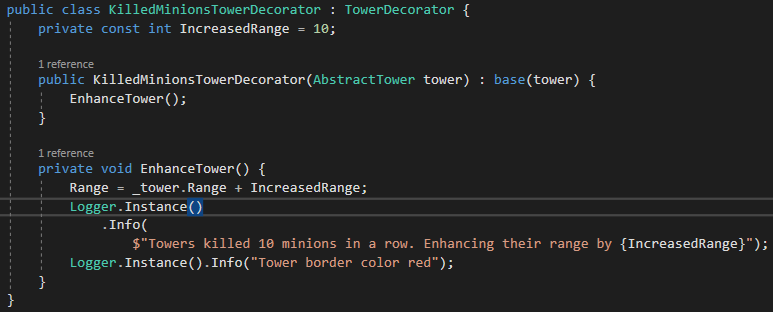
### Projektavimo šablono naudojimo pagrindimas

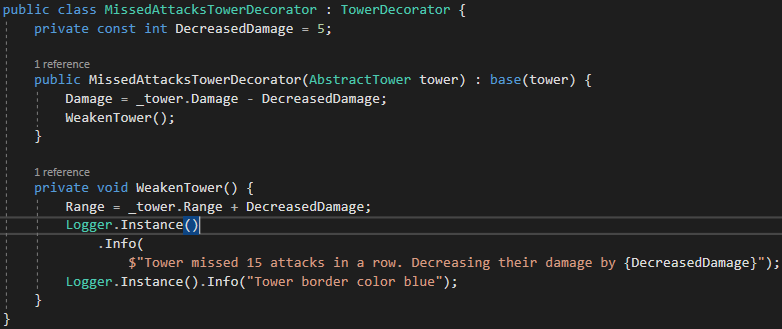
Mūsų žaidimui reikalinga duomenų bazė išsaugoti žaidėjų rezultatus, tačiau nenorime, kad žaidimas būtų priklausomas nuo konkrečios duomenų bazės ir bibliotekos, todėl sukūrėme savo sąsają, kurią realizavome naudodami Adapter šabloną. Tam, kad pademonstruotume Adapter šablono pritaikymą, realizavome RavenDB ir LiteDB sąsajas.

## Dekoratoriaus (Decorator) projektavimo pavyzdys (Simonas)

### Projektavimo šablono realizacija







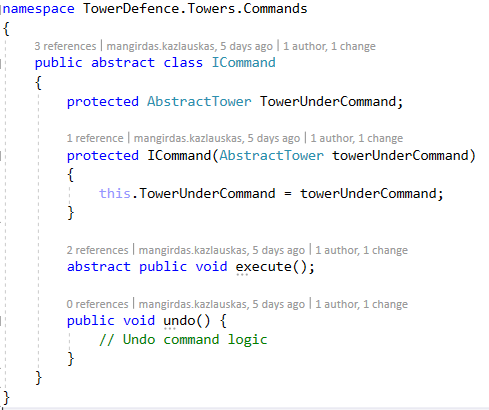
9 pav. Dekoratoriaus (Decorator) realizacija

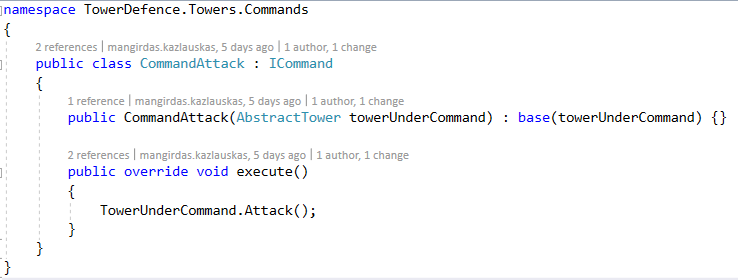
### Projektavimo šablono naudojimo pagrindimas

Žaidime priklausomai nuo to, kaip statome bokštus, priklauso žaidimo rezultatai. Jeigu bokštai sunaikina tam tikrą skaičių priešų iš eilės, įgauna papildomų savybių bei pakeičia spalvą arba nepataikius daug kartų iš eilės, bokštai yra susilpninami bei taip pat keičia spalvą. Tą reikia atlikti žaidimo veikimo metu (angl. runtime). Tam tinkamas Decorator šablonas, kuris leidžia praplėsti bokšto savybes, pridėti naujų funkcijų nekeičiant esamo bokšto realizacijos. Tam panaudojamas esamas bokšto objektas su jau esamomis savybėmis bei pridedama (dekoruojama) naujų funkcijų bei savybių.

## Komandos (Command) projektavimo pavyzdys (Mangirdas)

### Projektavimo šablono realizacija





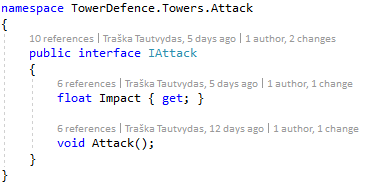
10 pav. Komandos (Command) realizacija

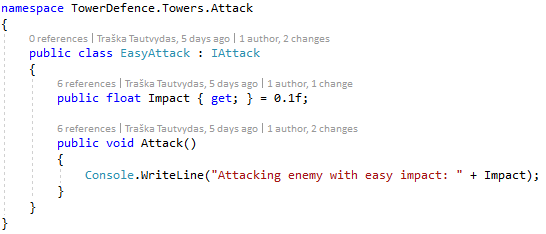
### Projektavimo šablono naudojimo pagrindimas

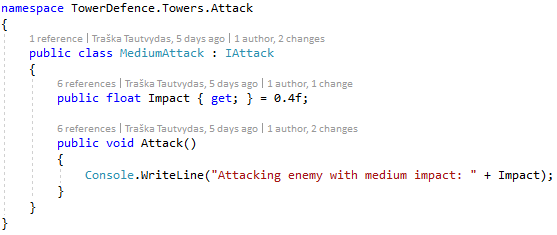
Komandos (angl. Command) projektavimo šablonas taikomas tada, kai norima, kad tam tikras veiksmas būtų ne klasės metodas, bet atskiras objektas, t.y., kai norima atskirti klientą (naudotoją), kuris vykdo komandą, nuo pačios komandos vykdymo detalių, priklausomybių nuo kitų objektų ir jų logikos. Mūsų pavyzdyje bokštų šaudymo komanda yra kaip atskiras objektas (klasė), o kai iš kliento pusės norima vykdyti bokšto šaudymo veiksmą, užtenka sukurti tos komandos objektą ir įvykdyti komandą execute() – nereikia komandai paduoti jokių parametrų, nereikia suprasti komandos logikos, užtenka ją iškviesti.

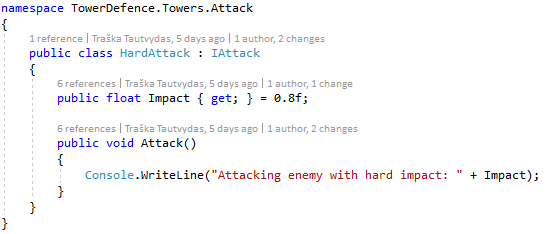
## Tilto (Bridge) projektavimo pavyzdys (Tautvydas)

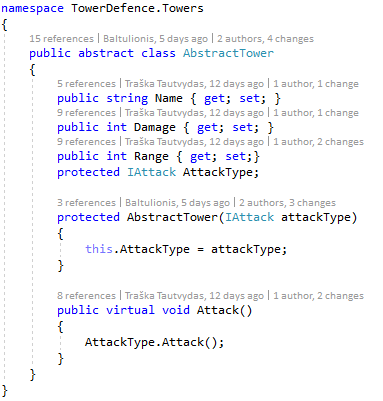
### Projektavimo šablono realizacija

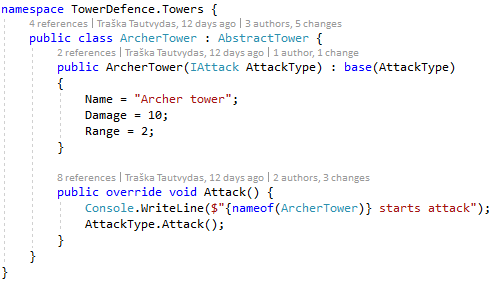


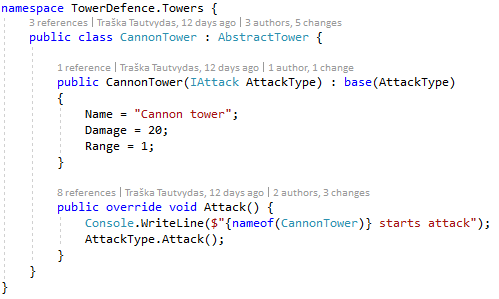












11 pav. Tilto (Bridge) realizacija

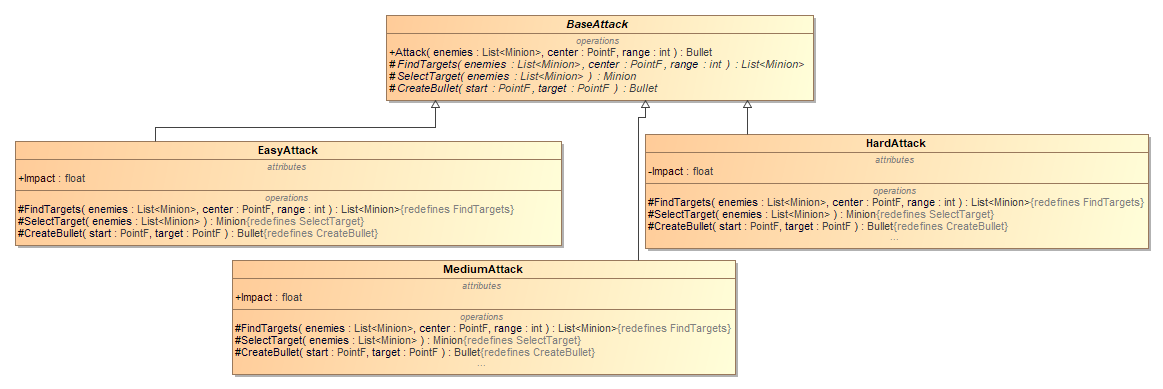
### Projektavimo šablono naudojimo pagrindimas

Bridge šablonas yra skirtas, kai norima atskirti abstrakciją nuo implementacijos, taigi šios gali kisti nepriklausomai viena nuo kitos. „Tower Defence“ žaidimo atveju, šis šablonas panaudotas apsauginiams bokštams, kurie gali pulti priešus skirtinga jėga. Vienas konkretus bokštas, kuris paveldi abstraktaus bokšto savybes, gali turėti skirtingus atakos tipus, nepriklausomai nuo to, koks bokštas yra kuriamas. Taigi, atsiranda daugiau ir įvairesnių galimybių sukurti bokštą, kuris tampa labiau dinamiškas, nes nėra tiesiogiai susietas su konkrečiu atakos tipu.

# Laboratorinis darbas nr. 2 (1 dalis)

## Template method projektavimo pavyzdys

### Klasių diagrama



12 pav. Template šablono klasių diagrama

### Projektavimo šablono realizacija

public abstract class BaseAttack

{

float Impact { get; }

public Bullet Attack(List<Minion> enemies, PointF center, int range) {

Bullet bullet = null;

var targets = FindTargets(enemies, center, range);

var enemy = SelectTarget(targets);

if (enemy != null)

{

bullet = CreateBullet(center, enemy.Center);

}

return bullet;

}

protected abstract List<Minion> FindTargets(List<Minion> enemies, PointF center, int range);

protected abstract Minion SelectTarget(List<Minion> enemies);

protected abstract Bullet CreateBullet(PointF start, PointF target);

}

public class MediumAttack : BaseAttack

{

public float Impact { get; } = 0.4f;

protected override List<Minion> FindTargets(List<Minion> enemies, PointF center, int range)

{

List<Minion> enemiesInRange = new List<Minion>();

foreach (var item in enemies)

{

double dist = Calc.Distance(center, item.Center);

if (range >= dist / 2)

enemiesInRange.Add(item);

}

return enemiesInRange;

}

protected override Minion SelectTarget(List<Minion> enemies) {

return enemies.LastOrDefault();

}

protected override Bullet CreateBullet(PointF start, PointF target) {

return new SimpleBullet(start, target) {

TargetType = TargetType.Ground,

GameObjectType = GameObjectTypeFactoryProvider.GetGameObjectType("heavyBullet")

};

}

}

public class HardAttack : BaseAttack {

public float Impact { get; } = 0.8f;

protected override List<Minion> FindTargets(List<Minion> enemies, PointF center, int range) {

List<Minion> enemiesInRange = new List<Minion>();

foreach (var item in enemies) {

double dist = Calc.Distance(center, item.Center);

if (range >= dist)

enemiesInRange.Add(item);

}

return enemiesInRange;

}

protected override Minion SelectTarget(List<Minion> enemies) {

return enemies.FirstOrDefault();

}

protected override Bullet CreateBullet(PointF start, PointF target) {

return new HeavyBullet(start, target) {

TargetType = TargetType.Ground,

GameObjectType = GameObjectTypeFactoryProvider.GetGameObjectType("heavyBullet")

};

}

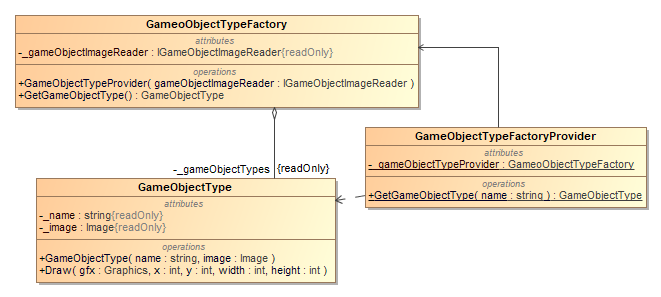
}

### Projektavimo šablono naudojimo pagrindimas

Template šablonas naudojamas, kai yra numatoma griežta algoritmo vykdymo tvarka, tačiau paliekama galimybė keisti kai kurias vykdymo dalis, priklausomai nuo konkrečios implementacijos. Mūsų žaidimo atveju yra numatytas atakos algoritmo veiksmų atlikimo eiliškumas, tačiau skirtingi atakos tipai skirtingai implementuoja atskiras algoritmo dalis. Norint sukurti naują atakos tipą, reikia paveldėti *BaseAttack* klasę ir realizuoti šiuos abstrakčius metodus: *FindTargets()*, *SelectTarget()*, *CreateBullet()*.

## Flyweight projektavimo pavyzdys (Tautvydas)

### Klasių diagrama



13 pav. Flyweight šablono klasių diagrama

### Projektavimo šablono realizacija

public class GameObjectType {

private readonly string \_name;

private readonly Image \_image;

public GameObjectType(string name, Image image) {

\_name = name;

\_image = image;

}

public void Draw(Graphics gfx, int x, int y, int width, int height) {

gfx.DrawImageUnscaled(new Bitmap(\_image, new Size(width, height)), x, y, width, height);

}

}

public class GameObjectTypeFactory {

private readonly IGameObjectImageReader \_gameObjectImageReader;

private readonly Dictionary<string, GameObjectType> \_gameObjectTypes;

public GameObjectTypeFactory(IGameObjectImageReader gameObjectImageReader) {

\_gameObjectImageReader = gameObjectImageReader;

\_gameObjectTypes = new Dictionary<string, GameObjectType>();

}

public GameObjectType GetGameOjObjectType(string name) {

if (\_gameObjectTypes.TryGetValue(name, out var gameObjectType)) {

return gameObjectType;

}

gameObjectType = new GameObjectType(name, \_gameObjectImageReader.GetGameObjectImage(name));

\_gameObjectTypes.Add(name, gameObjectType);

return gameObjectType;

}

}

public class GameObjectTypeFactoryProvider {

private static GameObjectTypeFactory \_gameObjectTypeFactory;

public static GameObjectType GetGameObjectType(string name) {

if (\_gameObjectTypeFactory == null) {

var proxy = new GameObjectImageCacheProxy();

proxy.GetGameObjectImages();

\_gameObjectTypeFactory = new GameObjectTypeFactory(proxy);

}

return \_gameObjectTypeFactory.GetGameOjObjectType(name);

}

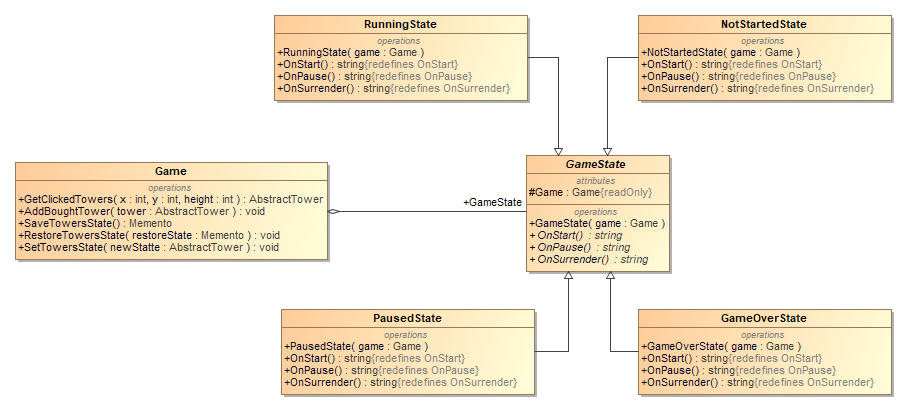
}

### Projektavimo šablono naudojimo pagrindimas

Flyweight šablonas naudojamas siekiant padidinti veikimo našumą ir sumažinti reikalingos atminties kiekį. Kartą sukurtas objektas išsaugomas *Map* duomenų struktūroje su numatytu unikaliu požymiu ir sekantį kartą pareikalavus tokio objekto, naujo nebekuriame, o grąžiname iš karto iš išsaugoto *Map‘e*. Žaidime šablonas pritaikytas kuriant žaidimo objektų tipus pagal pavadinimą (*beast, dragon, archerTower, haevyBullet, cannonTower*) ir bet kada paprašius dar kartą jau sukurto, grąžiname ne naujai sukurtą, o išsaugotą *Factory* objekto struktūroje.

## State projektavimo pavyzdys (Mangirdas)

### Klasių diagrama



14 pav. State šablono klasių diagrama

### Projektavimo šablono realizacija

public abstract class GameState {

protected readonly Game Game;

protected GameState(Game game) {

Game = game;

}

public abstract string OnStart();

public abstract string OnPause();

public abstract string OnSurrender();

}

public class GameOverState : GameState{

public GameOverState(Game game) : base(game) {

Game.Running = false;

}

public override string OnStart() {

Game.Running = true;

Game.GameState = new RunningState(Game);

return "Running";

}

public override string OnPause() {

return "Game over";

}

public override string OnSurrender() {

return "Game over";

}

}

public class NotStartedState : GameState {

public NotStartedState(Game game) : base(game) {

game.Running = false;

}

public override string OnStart() {

Game.Running = true;

Game.GameState = new RunningState(Game);

return "Running";

}

public override string OnPause() {

return "Not started";

}

public override string OnSurrender() {

return "Not started";

}

}

public class PausedState : GameState

{

public PausedState(Game game) : base(game) {

Game.Running = false;

}

public override string OnStart() {

Game.Running = true;

Game.GameState = new RunningState(Game);

return "Running";

}

public override string OnPause() {

return "Paused";

}

public override string OnSurrender() {

Game.Running = false;

Game.GameState = new GameOverState(Game);

return "Game over";

}

}

public class RunningState : GameState {

public RunningState(Game game) : base(game) {

}

public override string OnStart() {

return "Running";

}

public override string OnPause() {

Game.Running = false;

Game.GameState = new PausedState(Game);

return "Paused";

}

public override string OnSurrender() {

Game.Running = false;

Game.GameState = new GameOverState(Game);

return "Game over";

}

}

public class Game {

private List<AbstractTower> \_towers;

public GameState GameState { get; set; }

public Game(int nextWaveCounterSec) {

\_designatedNextLevelCounterSeconds = nextWaveCounterSec;

NextLevelCounterSeconds = nextWaveCounterSec;

GameState = new NotStartedState(this);

Clock.Interval = 1000;

Clock.Start();

Clock.Tick += Timer\_Tick;

Map = new Map();

Enemies = new List<Minion>();

Bullets = new List<Bullet>();

\_towers = new List<AbstractTower>();

TowersToAdd = new List<AbstractTower>();

Levels = new List<GameLevel>();

GameInfos = new List<GameInfo>();

AmountCalculatorVisitor = new TowersAmountCalculatorVisitor();

ArcherTowersAmount = 0;

CannonTowersAmount = 0;

}

public List<AbstractTower> GetClickedTowers(int x, int y, int height) {

return \_towers.Where(o =>

o.Dummy && (float)x > o.Center.X - o.Width / 2 && x < o.Center.X + o.Width / 2 &&

y > o.Center.Y - height / 2 && y < o.Center.Y + height / 2).ToList();

}

public void AddBoughtTower(AbstractTower tower) {

this.\_towers.Add(tower);

}

public Memento.Memento SaveTowersState() {

return new Memento.Memento(\_towers.Select(o => (AbstractTower)o.Clone()).ToList());

}

public void RestoreTowersState(Memento.Memento restoreState) {

restoreState.GetState(this);

}

public void SetTowersState(List<AbstractTower> newState) {

\_towers = newState;

}

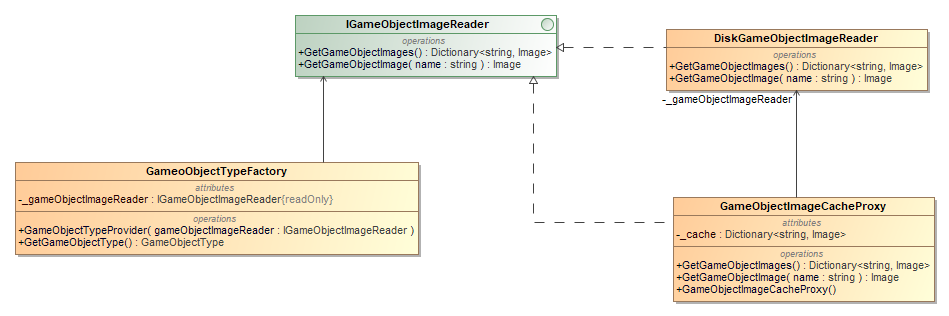
}

### Projektavimo šablono naudojimo pagrindimas

State šablonas skirtas objekto būsenų ir elgsenos, priklausomai nuo būsenos, atskyrimui. Žaidime būsena suteikiama visam žaidimo objektui *Game*. Galimi keli variantai, tokie kaip *RunningState, PausedState*. Pakeitus žaidimo būseną, žaidimo objektas perduodamas atitinkamam būsenos objektui, kur yra atliekami numatyti veiksmai (pvz. žaidimas stabdomas ar paleidžiamas).

## Proxy projektavimo pavyzdys (Simonas)

### Klasių diagrama



15 pav. Proxy šablono klasių diagrama

### Projektavimo šablono realizacija

public interface IGameObjectImageReader {

Dictionary<string, Image> GetGameObjectImages();

Image GetGameObjectImage(string name);

}

public class DiskGameObjectImageReader : IGameObjectImageReader {

public Dictionary<string, Image> GetGameObjectImages() {

var map = new Dictionary<string, Image>();

foreach (var imageName in new [] {"archerTower.png", "beast.png", "cannonTower.png", "dragon.png", "heavyBullet.png", "simpleBullet.png"})

{

var image = Image.FromFile(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, $"pics/{imageName}"));

map.Add(imageName.Substring(0, imageName.Length - ".png".Length), image);

}

return map;

}

public Image GetGameObjectImage(string name) {

return Image.FromFile(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, $"pics/{name}.png"));

}

}

public class GameObjectImageCacheProxy : IGameObjectImageReader {

private readonly IGameObjectImageReader \_gameObjectImageReader;

private Dictionary<string, Image> \_cache;

public GameObjectImageCacheProxy() {

\_gameObjectImageReader = new DiskGameObjectImageReader();

\_cache = new Dictionary<string, Image>();

}

public Dictionary<string, Image> GetGameObjectImages() {

if (\_cache.Count == 0) {

\_cache = \_gameObjectImageReader.GetGameObjectImages();

}

return \_cache;

}

public Image GetGameObjectImage(string name) {

if (\_cache.TryGetValue(name, out var image)) {

return image;

}

image = \_gameObjectImageReader.GetGameObjectImage(name);

return image;

}

}

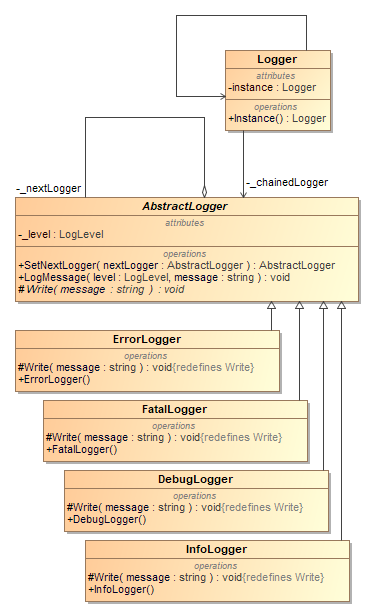
### Projektavimo šablono naudojimo pagrindimas

Proxy šablonas naudojamas siekiant paslėpti sudėtingus originalaus objekto kūrimo veiksmus bei kontroliuoti paties objekto ir jo metodų tiesioginį pasiekiamumą. Proxy dėka reikiami objektai užkraunami tik tuomet, kai yra paprašoma. Žaidime šablonas pritaikytas žaidimo personažų įvaizdžio užkrovimui (objektų paveiksliukai).

# Laboratorinis darbas nr. 2 (2 dalis)

## Chain of responsibility projektavimo pavyzdys

### Klasių diagrama



16 pav. Chain of responsibility šablono klasių diagrama

### Projektavimo šablono realizacija

public enum LogLevel

{

INFO = 1,

DEBUG = 2,

ERROR = 3,

FATAL = 4

};

abstract class AbstractLogger

{

protected AbstractLogger \_nextLogger;

protected LogLevel \_level;

public AbstractLogger SetNextLogger(AbstractLogger nextLogger)

{

return \_nextLogger = nextLogger;

}

public void LogMessage(LogLevel level, String message)

{

if (\_level <= level)

{

Write(message);

}

else if (\_nextLogger != null)

{

\_nextLogger.LogMessage(level, message);

}

}

abstract protected void Write(String message);

}

class DebugLogger : AbstractLogger

{

public DebugLogger()

{

\_level = LogLevel.DEBUG;

}

protected override void Write(string message)

{

Logger.Instance().GetLogger().Debug(message);

}

}

class ErrorLogger : AbstractLogger

{

public ErrorLogger()

{

\_level = LogLevel.ERROR;

}

protected override void Write(string message)

{

Logger.Instance().GetLogger().Error(message);

}

}

class FatalLogger : AbstractLogger

{

public FatalLogger()

{

\_level = LogLevel.FATAL;

}

protected override void Write(string message)

{

Logger.Instance().GetLogger().Fatal(message);

}

}

class InfoLogger : AbstractLogger

{

public InfoLogger()

{

\_level = LogLevel.INFO;

}

protected override void Write(string message)

{

Logger.Instance().GetLogger().Info(message);

}

}

public class Logger {

private static volatile Logger \_instance;

private static readonly object \_syncRoot = new object();

private static ILog \_logger;

private AbstractLogger \_chainedLogger;

private Logger() {

if (\_logger == null) {

var logRepository = LogManager.GetRepository(Assembly.GetEntryAssembly() ?? Assembly.GetExecutingAssembly());

var configFile = new FileInfo("App.config");

XmlConfigurator.Configure(logRepository, configFile);

\_logger = LogManager.GetLogger(nameof(TowerDefence));

\_chainedLogger = new FatalLogger();

\_chainedLogger

.SetNextLogger(new ErrorLogger())

.SetNextLogger(new DebugLogger())

.SetNextLogger(new InfoLogger());

}

}

public static Logger Instance() {

if (\_instance == null) {

lock (\_syncRoot) {

if (\_instance == null)

\_instance = new Logger();

}

}

return \_instance;

}

public ILog GetLogger()

{

return \_logger;

}

public void Log(LogLevel level, string message)

{

\_chainedLogger.LogMessage(level, message);

}

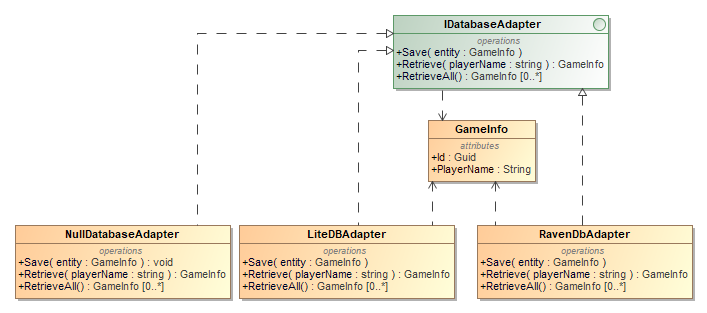
}

### Projektavimo šablono naudojimo pagrindimas

Chain of responsibility šablonas naudojamas, kai yra daug užklausą galinčių įvykdyti objektų, tačiau reikia parinkti tinkamiausią. Taip sudaroma panašių objektų grandinėlė, kurioje užklausą visuomet gauna pirmasis objektas ir bando įvykdyti, o jei to padaryti negali, perleidžia sekančiam objektui, kuris atlieka tokius pačius veiksmus. Taip vyksta tol, kol užklausa pasiekia reikiamą objektą. Žaidime šablonas pritaikytas realizuojant *Logger‘į*. Sukurti keli užklausas apdorojantys objektai ir kiekvienam suteiktas lygis. Pagal lygį nusprendžiama, kuris grandinėlės objektas imasi veiksmų.

## Null object projektavimo pavyzdys

### Klasių diagrama



17 pav. Null object šablono klasių diagrama

### Projektavimo šablono realizacija

public interface IDatabaseAdapter {

void Save(GameInfo entity);

GameInfo Retrieve(string playerName);

List<GameInfo> RetrieveAll();

}

public class LiteDBAdapter : IDatabaseAdapter {

private readonly LiteRepository \_repository;

public LiteDBAdapter(LiteRepository repository) {

\_repository = repository;

}

public void Save(GameInfo entity) {

\_repository.Insert(entity);

}

public GameInfo Retrieve(string playerName) {

return \_repository.Query<GameInfo>().Where(o => o.PlayerName == playerName).FirstOrDefault();

}

public List<GameInfo> RetrieveAll() {

return \_repository.Query<GameInfo>().ToList();

}

}

public class RavenDbAdapter : IDatabaseAdapter {

private readonly IDocumentStore \_store;

public RavenDbAdapter(IDocumentStore store) {

\_store = store;

}

public void Save(GameInfo entity) {

using (IDocumentSession session = \_store.OpenSession()) {

session.Store(entity);

session.SaveChanges();

}

}

public GameInfo Retrieve(string playerName) {

using (IDocumentSession session = \_store.OpenSession()) {

return session.Query<GameInfo>().FirstOrDefault(o => o.PlayerName == playerName);

}

}

public List<GameInfo> RetrieveAll() {

using (IDocumentSession session = \_store.OpenSession()) {

return session.Load<GameInfo>().ToList();

}

}

}

public class NullDatabaseAdapter : IDatabaseAdapter {

public void Save(GameInfo entity) {

Logger.Instance().GetLogger().Info("Game info is saved to database");

}

public GameInfo Retrieve(string playerName) {

Logger.Instance().GetLogger().Info("Game info retrieved from database");

return new GameInfo {

Id = Guid.NewGuid(),

PlayerName = "Default game info"

};

}

public List<GameInfo> RetrieveAll() {

Logger.Instance().GetLogger().Info("Game info list retrieved from database");

return new List<GameInfo> {

new GameInfo {

Id = Guid.NewGuid(),

PlayerName = "Default game info list member"

}

};

}

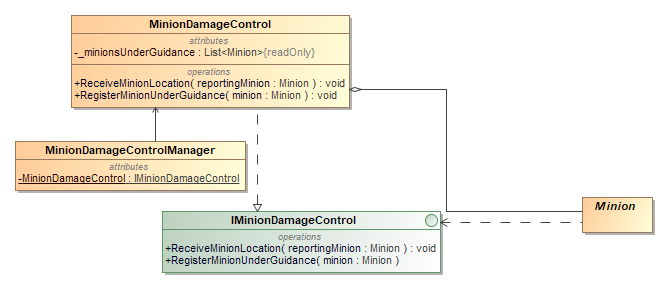
}

### Projektavimo šablono naudojimo pagrindimas

Null object šablonas skirtas sukurti tam tikros klasės tuščiai implementacijai, kuri būtų naudojama tuomet, kai nėra reikiamo konkretaus atitikmens. Šabloną susiejome su Adapter šablonu sukuriant tuščią duomenų bazės adapterį tam atvejui, jei nebūtų galimybės panaudoti nei vienos kitos adapterio implementacijos.

## Mediator projektavimo pavyzdys (Simonas)

### Klasių diagrama



18 pav. Mediator šablono klasių diagrama

### Projektavimo šablono realizacija

public interface IMinionDamageControl {

void ReceiveMinionLocation(Minion reportingMinion);

void RegisterMinionUnderGuidance(Minion minion);

}

public class MinionDamageControlManager {

public static IMinionDamageControl MinionDamageControl { get; } = new MinionDamageControl();

}

public class MinionDamageControl : IMinionDamageControl {

private readonly IList<Minion> \_minionsUnderGuidance = new List<Minion>();

public void ReceiveMinionLocation(Minion reportingMinion) {

foreach (var currentMinionUnderGuidance in \_minionsUnderGuidance.Where(o => o != reportingMinion)) {

if (Calc.Distance(currentMinionUnderGuidance.Center, reportingMinion.Center) < 2) {

currentMinionUnderGuidance.HitPoints -= reportingMinion.LastReceivedDamage / 4;

}

}

}

public void RegisterMinionUnderGuidance(Minion minion) {

if (!\_minionsUnderGuidance.Contains(minion)) {

\_minionsUnderGuidance.Add(minion);

}

}

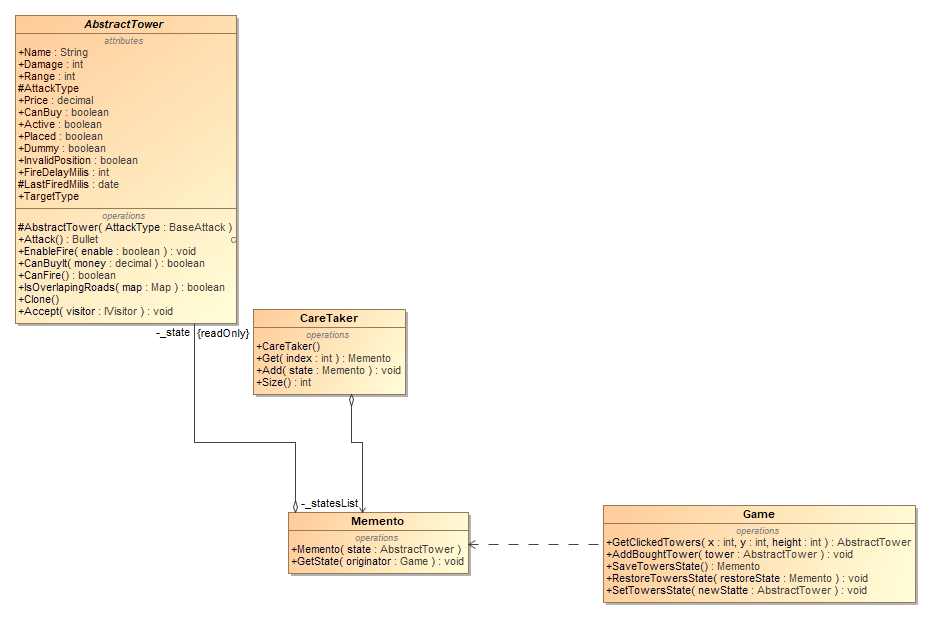
}

### Projektavimo šablono naudojimo pagrindimas

Mediator šablonas skirtas bendravimui tarp atskirų programos objektų ar modulių siekiant išvengti tiesioginės programinių vienetų priklausomybės. Tam sukuriama terpė, per kurią objektai gali komunikuoti. Žaidime šablonas pritaikytas bendravimui tarp priešų. Jeigu vienas iš priešų pašaunamas, informuojami visi kiti ir pagal atstumą nustatoma, ar žala buvo padaryta ir šalia buvusiam priešui.

## Memento projektavimo pavyzdys (Tautvydas)

### Klasių diagrama



19 pav. Memento šablono klasių diagrama

### Projektavimo šablono realizacija

public class Memento {

private readonly List<AbstractTower> \_state;

public Memento(List<AbstractTower> state) {

\_state = state;

}

public void GetState(Game originator) {

originator.SetTowersState(this.\_state);

}

}

public class CareTaker {

private readonly List<Memento> \_statesList;

public CareTaker() {

\_statesList = new List<Memento>();

}

public void Add(Memento state) {

\_statesList.Add(state);

}

public Memento Get(int index) {

Memento restoredState = \_statesList[index];

\_statesList.RemoveAt(index);

return restoredState;

}

public int Size() {

return \_statesList.Count;

}

}

public class Game {

private List<AbstractTower> \_towers;

public List<AbstractTower> GetClickedTowers(int x, int y, int height) {

return \_towers.Where(o =>

o.Dummy && (float)x > o.Center.X - o.Width / 2 && x < o.Center.X + o.Width / 2 &&

y > o.Center.Y - height / 2 && y < o.Center.Y + height / 2).ToList();

}

public void AddBoughtTower(AbstractTower tower) {

this.\_towers.Add(tower);

}

public Memento.Memento SaveTowersState() {

return new Memento.Memento(\_towers.Select(o => (AbstractTower)o.Clone()).ToList());

}

public void RestoreTowersState(Memento.Memento restoreState) {

restoreState.GetState(this);

}

public void SetTowersState(List<AbstractTower> newState) {

\_towers = newState;

}

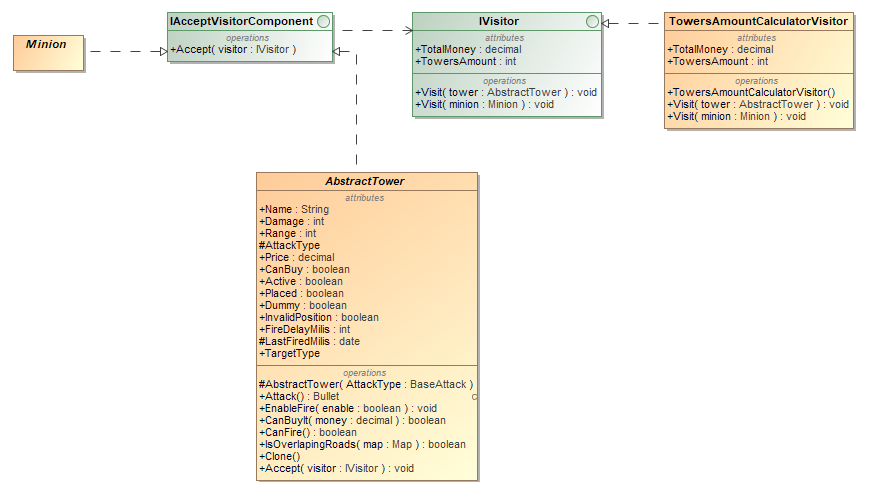
}

### Projektavimo šablono naudojimo pagrindimas

Šablonas skirtas atstatyti tam tikru momentu išsaugotai būsenai. Žaidime ši funkcija realizuota suteikiant žaidėjui galimybę atsatyti bokštų kiekį po atlikto pirkimo. Atgaunama tik dalis sumokėtų pinigų.

## Visitor projektavimo pavyzdys (Mangirdas)

### Klasių diagrama



20 pav. Visitor šablono klasių diagrama

### Projektavimo šablono realizacija

public interface IVisitor

{

decimal TotalMoney { get; set; }

int TowersAmount { get; set; }

void Visit(AbstractTower tower);

void Visit(Minion minion);

}

public interface IAcceptVisitorComponent

{

void Accept(IVisitor visitor);

}

public class TowersAmountCalculatorVisitor : IVisitor

{

public decimal TotalMoney { get; set; }

public int TowersAmount { get; set; }

public TowersAmountCalculatorVisitor()

{

TotalMoney = 0;

TowersAmount = 0;

}

public void Visit(AbstractTower tower)

{

TowersAmount = Convert.ToInt32(Math.Floor(TotalMoney/tower.Price));

}

public void Visit(Minion minion)

{

TotalMoney += minion.Money;

}

}

public abstract class Minion : GameObject, IAcceptVisitorComponent {

public void Accept(IVisitor visitor)

{

visitor.Visit(this);

}

}

public abstract class AbstractTower : GameObject, ICloneable, IAcceptVisitorComponent

{

public void Accept(IVisitor visitor)

{

visitor.Visit(this);

}

}

public class Game {

public IVisitor AmountCalculatorVisitor { get; set; }

public int ArcherTowersAmount { get; set; }

public int CannonTowersAmount { get; set; }

public Game(int nextWaveCounterSec) {

\_designatedNextLevelCounterSeconds = nextWaveCounterSec;

NextLevelCounterSeconds = nextWaveCounterSec;

GameState = new NotStartedState(this);

Clock.Interval = 1000;

Clock.Start();

Clock.Tick += Timer\_Tick;

Map = new Map();

Enemies = new List<Minion>();

Bullets = new List<Bullet>();

\_towers = new List<AbstractTower>();

TowersToAdd = new List<AbstractTower>();

Levels = new List<GameLevel>();

GameInfos = new List<GameInfo>();

AmountCalculatorVisitor = new TowersAmountCalculatorVisitor();

ArcherTowersAmount = 0;

CannonTowersAmount = 0;

}

public void Update(bool force)

{

AmountCalculatorVisitor.TotalMoney = 0;

if (!Running && !force)

return;

if (NextLevelCounterSeconds == 0) {

SendNextLevel();

}

foreach (var item in Levels) {

if (item.Active && item.CanSpawn()) {

Enemies.Add(item.SpawnOne(Map));

}

}

// remove destroyed enemies and calc money and points

List<Minion> remove = Enemies.Where(x => x.HitPoints < 1 || x.EndReached).ToList();

foreach (var item in remove) {

if (item.EndReached)

Life--;

if (item.HitPoints < 1) {

Money = Money + item.Money;

Points = Points + item.Points;

}

Enemies.Remove(item);

\_destroyedEnemies++;

}

foreach (var enemy in Enemies)

{

enemy.Accept(AmountCalculatorVisitor);

}

foreach (var tower in \_towers.Where(t => t.Dummy))

{

tower.Accept(AmountCalculatorVisitor);

if (tower.GetType() == typeof(ArcherTower))

{

ArcherTowersAmount = AmountCalculatorVisitor.TowersAmount;

}

else

{

CannonTowersAmount = AmountCalculatorVisitor.TowersAmount;

}

}

if (\_destroyedEnemies >= 5 && !\_decorated) {

for (int i = 0; i < \_towers.Count; i++) {

if (\_towers[i].Active) {

\_towers[i] = new KilledMinionsTowerDecorator(\_towers[i]);

}

}

\_decorated = true;

}

}

}

### Projektavimo šablono naudojimo pagrindimas

Visitor šablonas skirtas aplankyti jam žinomus objektus ir priklausomai nuo lankomo objekto, įvykdyti atitinkamą algoritmą. Žaidime visitor šablonas veikia skaičiuodamas, kiek bokštų bus galima nupirkti, jeigu bus nužudyti visi esamu metu matomi priešai.