**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**T120B516 Objektinis programų projektavimas**

**(Design Patterns)**

**Projektinio darbo ataskaita**

Atliko: Karolina Jašauskaitė, IFF-4/1,

Žygimantas Skinkys, IFF-4/1,

Aidas Balčaitis, IFF-4/1

Kaunas 2017

Turinys

[Įvadas 3](#_Toc500235812)

[Darbo tikslas 3](#_Toc500235813)

[1. Pirmas labaratorinis darbas 3](#_Toc500235814)

[Programavimo šablonai: 3](#_Toc500235815)

[1.1. Singleton 3](#_Toc500235816)

[1.2. Factory 4](#_Toc500235817)

[1.3. Abstract Factory 6](#_Toc500235818)

[1.4. Strategy 7](#_Toc500235819)

[1.5. Observer 8](#_Toc500235820)

[2. Antras labaratorinis darbas 12](#_Toc500235821)

[Programavimo šablonai 12](#_Toc500235822)

[2.1. Adapter 12](#_Toc500235823)

[2.2. Decorator 14](#_Toc500235824)

[2.3. Facade 16](#_Toc500235825)

[2.4. Command 18](#_Toc500235826)

[1 ir 2 Labaratorinių darbų klasių diagramos 22](#_Toc500235827)

[Išvados 22](#_Toc500235828)

# Įvadas

## Darbo tikslas

Kuriant „Pacman” žaidimą išmokti taikyti projektavimo šablonus (angl. „design patterns“) ir susipažinti su jų naudojimo ypatumais.

„Pacman” žaidime vartotojas valdys pagrindinį herojų, kurio tikslas surinkti kuo daugiau taškį renkant “Candy”, ir išvengi priešų “Enemy”.

Žaidime naudojami 5 pagrindiniai objektai: žaidimas, žaidimo lenta, priešai, saldainiai ir žaidėjo valdomas objektas.

# Pirmas labaratorinis darbas

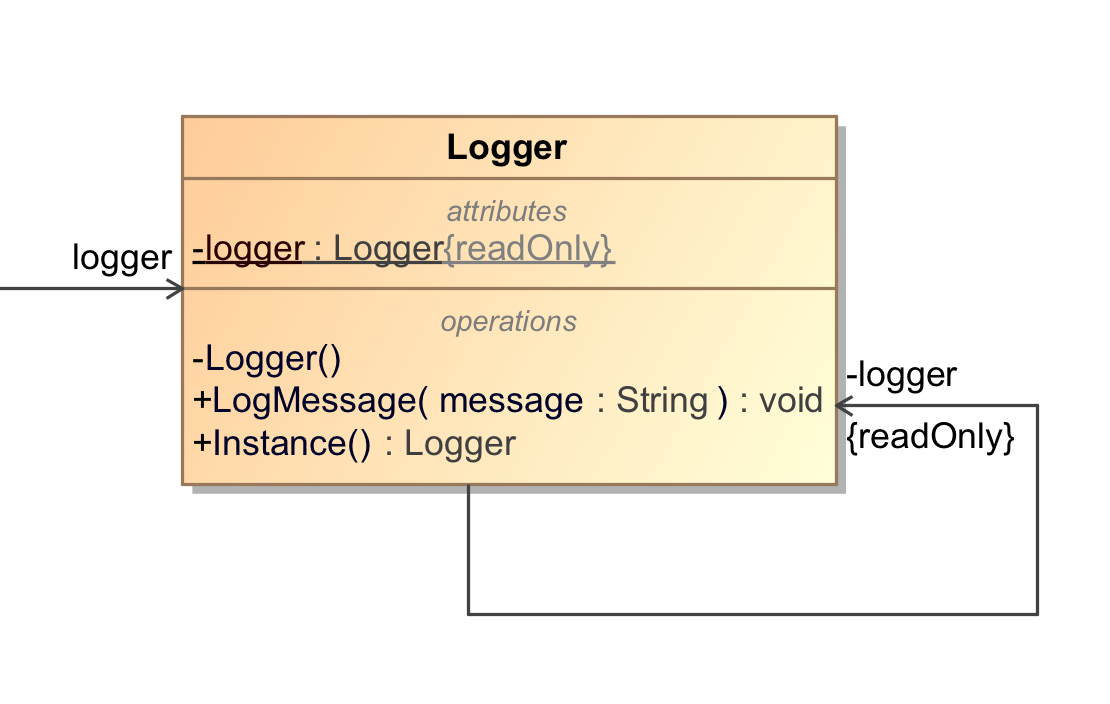
## Programavimo šablonai:

## Singleton

Panaudojimo tikslas:

Panaudojome Singleton programavimo šabloną, kad užtikrintume, kad vienu metu žaidėjas žaidžia tik vieną žaidimą.

Uml diagrama:



Pav. 1 Singleton programavimo šablonas

Kodas:

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Pacman.Classes

{

class Logger

{

private static readonly Logger logger = new Logger();

static Logger() { } // Make sure it's truly lazy

private Logger() { } // Prevent instantiation outside

public static Logger Instance { get { return logger; } }

public void LogMessage(string message)

{

Debug.WriteLine(message);

}

}

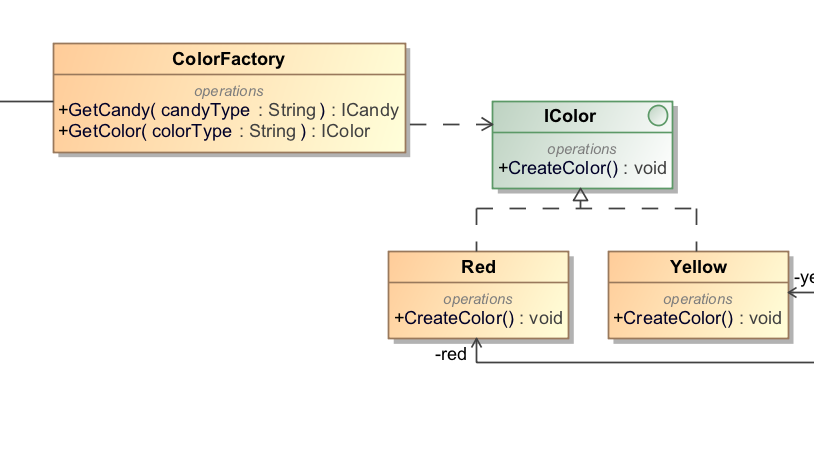
}

## Factory

Panaudojimo tikslas:

Kadangi turime 4 skirtingų tipų saldainius (Banana, Big, Small, Cherry), norint juos realizuoti ir panaudoti savo žaidime, pasinaudojome “Factory” programavimo šabloną, kuris pagal gautą atributą sukurs saldainius.

Uml diagrama:



Pav. 2 Factory programavimo šablonas

Kodas:

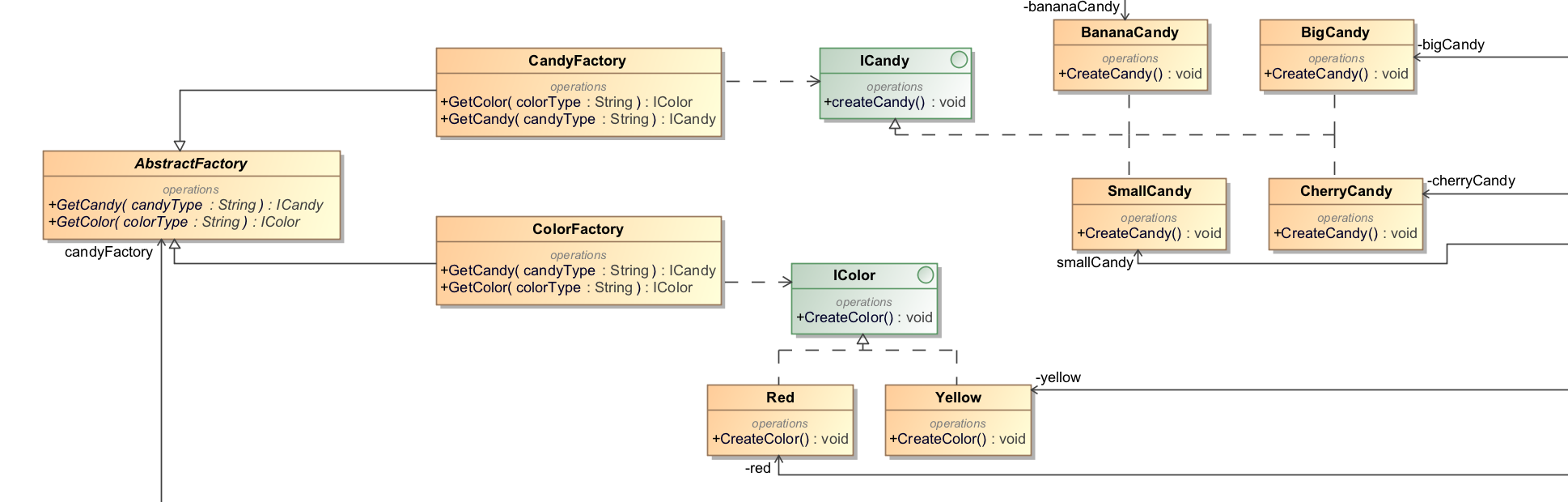
|  |
| --- |
| ColorFactory.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Pacman.Classes.Colors;  namespace Pacman.Classes  {  class ColorFactory : AbstractFactory  {  public override ICandy GetCandy(string candyType)  {  return null;  }  public override IColor GetColor(string colorType)  {  if (colorType == null)  {  return null;  }  if (colorType.Equals("yellow", StringComparison.InvariantCultureIgnoreCase))  {  return new Yellow();  }  if (colorType.Equals("red", StringComparison.InvariantCultureIgnoreCase))  {  return new Red();  }  return null;  }  }  } |
| Red.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes.Colors  {  class Red : IColor  {  public void CreateColor()  {  Debug.WriteLine("Red color was created.");  }  }  } |
| Yellow.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes.Colors  {  class Yellow : IColor  {  public void CreateColor()  {  Debug.WriteLine("Yellow color was created.");  }  }  } |

## Abstract Factory

Panaudojimo tikslas:

Abstract Factory šabloną panaudojom apjungti ankščiau panaudotų Factory šablonų apjungimui.

Uml diagrama:



Pav. 3 Abstract Factory programavimo šablonas

Kodas:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Pacman.Classes

{

abstract class AbstractFactory

{

public abstract ICandy GetCandy(string candyType);

public abstract IColor GetColor(string colorType);

}

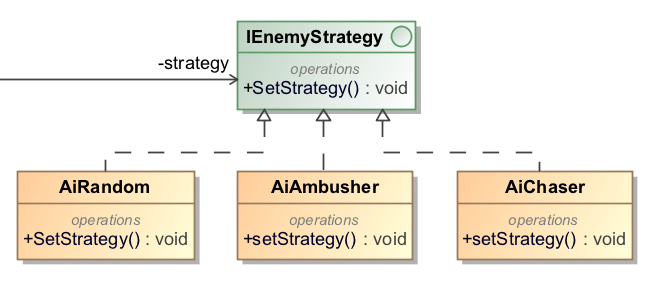
}

## Strategy

Panaudojimo tikslas:

Strategy šabloną panaudojom apibrėžti kelis Enemy algoritmus, juos paslėpti. Strategy šablonas leidžia pakeisti strategiją nekeičiant nuo Enemy obj

Uml diagrama:



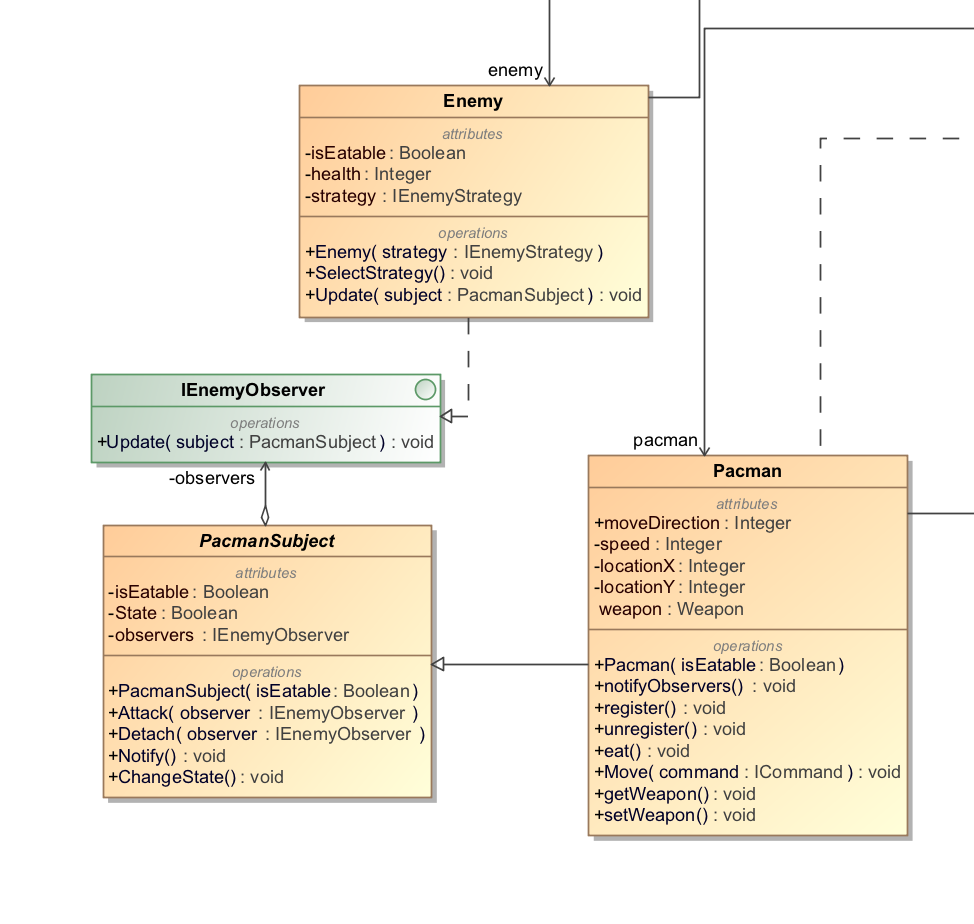
Pav. 4 Strategy programavimo šablonas

Kodas:

|  |
| --- |
| IEnemyStrategy.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  interface IEnemyStrategy  {  void SetStrategy();  }  } |
| AiRandom.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class AiRandom : IEnemyStrategy  {  public void SetStrategy()  {  Debug.WriteLine("Setting strategy to random");  }  }  } |
| AiAmbusher.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class AiAmbusher : IEnemyStrategy  {  public void SetStrategy()  {  Debug.WriteLine("Setting strategy to ambusher");  }  }  } |
| AiChaser.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class AiChaser : IEnemyStrategy  {  public void SetStrategy()  {  Debug.WriteLine("Setting strategy to chaser.");  }  }  } |

## Observer

Panaudojimo tikslas:



Uml diagrama:

Pav. 5 Observer programavimo šablonas

Kodas:

|  |
| --- |
| IEnemyObserver.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  interface IEnemyObserver  {  void Update(PacmanSubject subject);  }  } |
| PacmanSubject.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Runtime.CompilerServices;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  abstract class PacmanSubject  {  private bool isEatable;  private List<IEnemyObserver> observers = new List<IEnemyObserver>();  public PacmanSubject(bool isEatable)  {  this.isEatable = isEatable;  }  public void Attach(IEnemyObserver observer)  {  observers.Add(observer);  }  public void Detach(IEnemyObserver observer)  {  observers.Remove(observer);  }  public void Notify()  {  foreach (IEnemyObserver observer in observers)  {  observer.Update(this);  }  }  public bool State  {  get { return isEatable; }  set  {  if (isEatable != value)  {  isEatable = value;  Notify();  }  }  }  public void ChangeState()  {  isEatable = !isEatable;  Notify();  }  }  } |
| Pacman.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class Pacman : PacmanSubject  {  public int moveDirection { set; get; }  private int speed;  private int locationX;  private int locationY;  public Weapon weapon { set; get; }  public Pacman(bool isEatable) : base(isEatable)  {  }  public void notifyObservers()  {    }  public void register()  {    }  public void unregister()  {    }  public void eat()  {    }  public void Move(ICommand command)  {  command.Move();  }  public void getWeapon()  {    }  public void setWeapon()  {    }  }  } |
| Enemy.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class Enemy : IEnemyObserver  {  private bool isEatable;  private int health;  public IEnemyStrategy strategy { get; set; }  public Enemy(IEnemyStrategy strategy)  {  this.strategy = strategy;  }  public void SelectStrategy()  {  strategy.SetStrategy();  }  public void Update(PacmanSubject subject)  {  Debug.WriteLine("Pacman changed it's state to " + subject.State);  }  }  } |

# Antras labaratorinis darbas

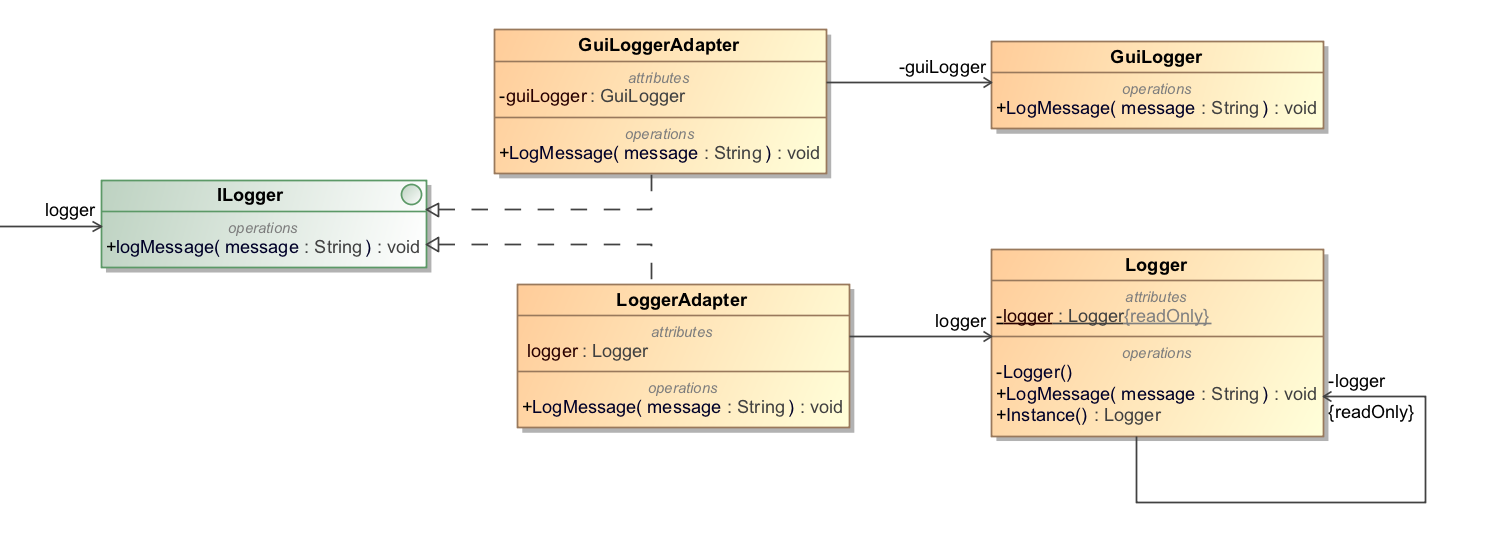
## Programavimo šablonai

## Adapter

Panaudojimo tikslas:

Adapter programavimo šabloną naudojame, kad turėtume galimybę paleisti programą ir su grafine sąsaja ir be jos.

Uml diagrama:



Pav. 6 Adapter programavimo šablonas

Kodas:

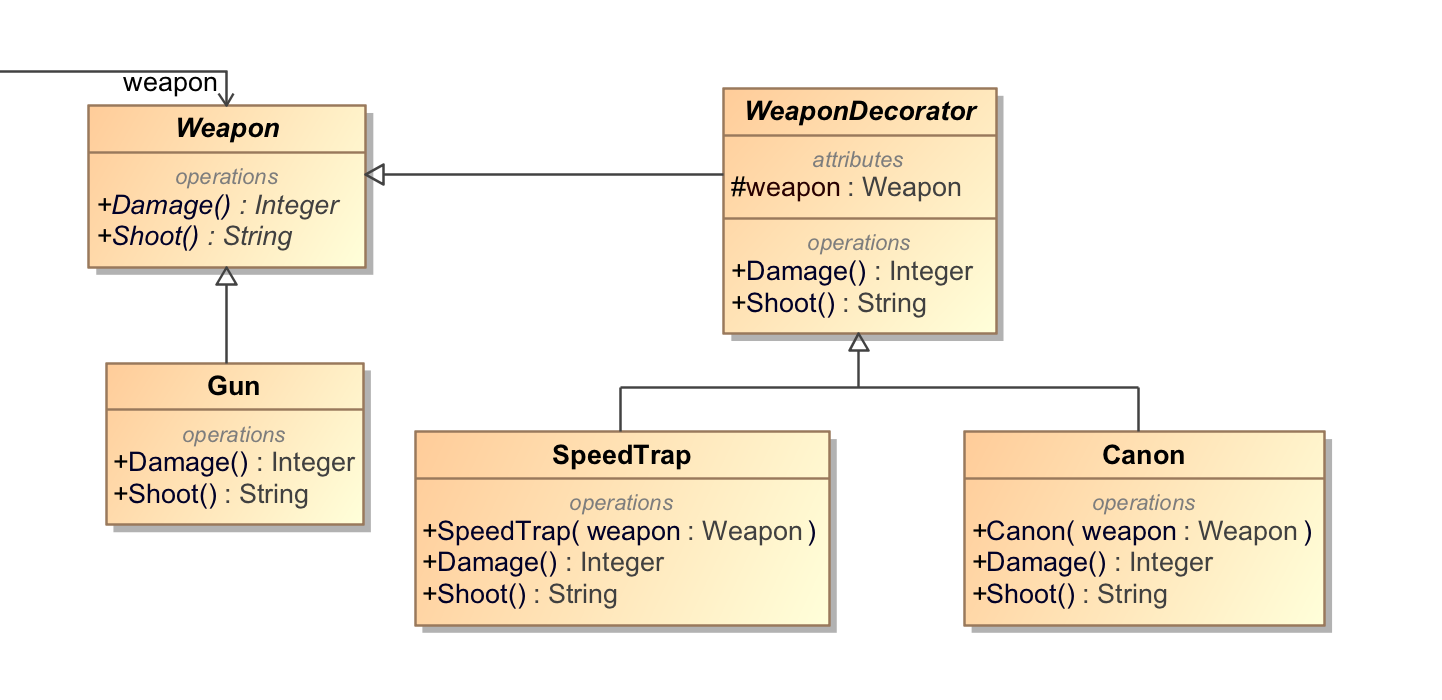
|  |
| --- |
| GuiLoggerAdapter.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class GuiLoggerAdapter : ILogger  {  private GuiLogger guiLogger = new GuiLogger();  public void LogMessage(string message)  {  guiLogger.LogMessage(message);  }  }  } |
| LoggerAdapter.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class LoggerAdapter : ILogger  {  public void LogMessage(string message)  {  Logger.Instance.LogMessage(message);  }  }  } |

## Decorator

Panaudojimo tikslas:

Pasirinkome naudoti Decorator programavimo šabloną, nes Pacman gali turėti vieną iš dviejų ginklų. Vietoje to kad naudoti switch case ar if sakinius priklausomai nuo pasirinkto ginklo, panaudojome Decorator programavimo šabloną, kad ginklai būtų pridedami be didesnių sunkumų.

Uml diagrama:



Pav. 7 Decorator programavimo šablonas

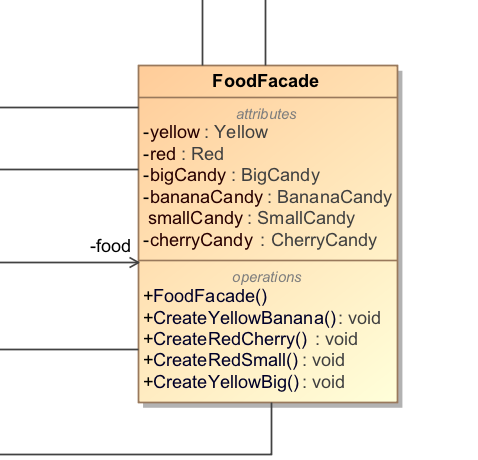
Kodas:

|  |
| --- |
| Gun.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class Gun : Weapon  {  public override int Damage()  {  return 10;  }  public override string Shoot()  {  return "Shooting simple bullet";  }  }  } |
| WeaponDecorator.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  abstract class WeaponDecorator : Weapon  {  protected Weapon weapon = null;  public override int Damage()  {  return weapon.Damage();  }  public override string Shoot()  {  return weapon.Shoot();  }  }  } |
| Weapon.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  abstract class Weapon  {  public abstract int Damage();  public abstract string Shoot();  }  } |
| SpeedTrap.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class SpeedTrap : WeaponDecorator  {  public SpeedTrap(Weapon weapon)  {  this.weapon = weapon;  }  public override int Damage()  {  return base.Damage() + 10;  }  public override string Shoot()  {  if (this.weapon != null)  {  return base.Shoot() + " and slowing bullet";  }  return string.Empty;  }  }  } |
| Cannon.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class Cannon : WeaponDecorator  {  public Cannon(Weapon weapon)  {  this.weapon = weapon;  }  public override int Damage()  {  return base.Damage() + 100;  }  public override string Shoot()  {  if (weapon != null)  {  return base.Shoot() + " and instant-kill bullets";  }  return string.Empty;  }  }  } |

## Facade

Panaudojimo tikslas:

Uml diagrama:



Pav. 8 Facade programavimo šablonas

Kodas:

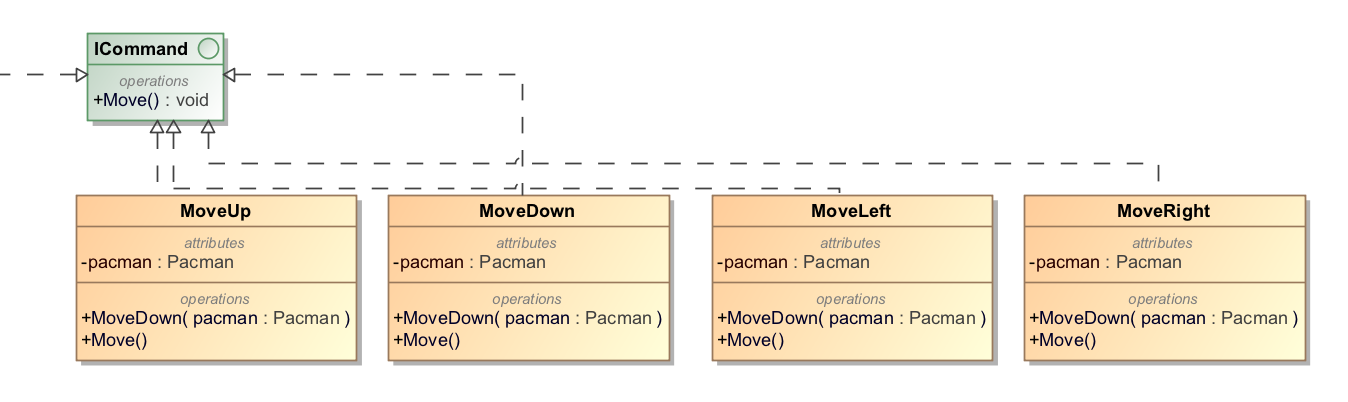
|  |
| --- |
| FoodFacade.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using Pacman.Classes.Candies;  using Pacman.Classes.Colors;  namespace Pacman.Classes  {  class FoodFacade  {  private BananaCandy bananaCandy;  private BigCandy bigCandy;  private SmallCandy smallCandy;  private CherryCandy cherryCandy;  private Yellow yellow;  private Red red;  public FoodFacade()  {  bananaCandy = new BananaCandy();  bigCandy = new BigCandy();  smallCandy = new SmallCandy();  cherryCandy = new CherryCandy();  yellow = new Yellow();  red = new Red();  }  public void CreateYellowBanana()  {  Debug.WriteLine("Creating colorful candy");  bananaCandy.CreateCandy();  yellow.CreateColor();  Debug.WriteLine("");  }  public void CreateRedCherry()  {  Debug.WriteLine("Creating colorful candy");  cherryCandy.CreateCandy();  red.CreateColor();  Debug.WriteLine("");  }  public void CreateRedSmall()  {  Debug.WriteLine("Creating colorful candy");  smallCandy.CreateCandy();  red.CreateColor();  Debug.WriteLine("");  }  public void CreateYellowBig()  {  Debug.WriteLine("Creating colorful candy");  bigCandy.CreateCandy();  yellow.CreateColor();  Debug.WriteLine("");  }  }  } |

## Command

Panaudojimo tikslas:

Command šabloną panaudojom tam, kad paslėpti komandų apdorojimų logiką. Tai mums leis vėliau lengvai pridėti naujų kamandų ir jas apdoroti objektiškai.

Uml diagrama:

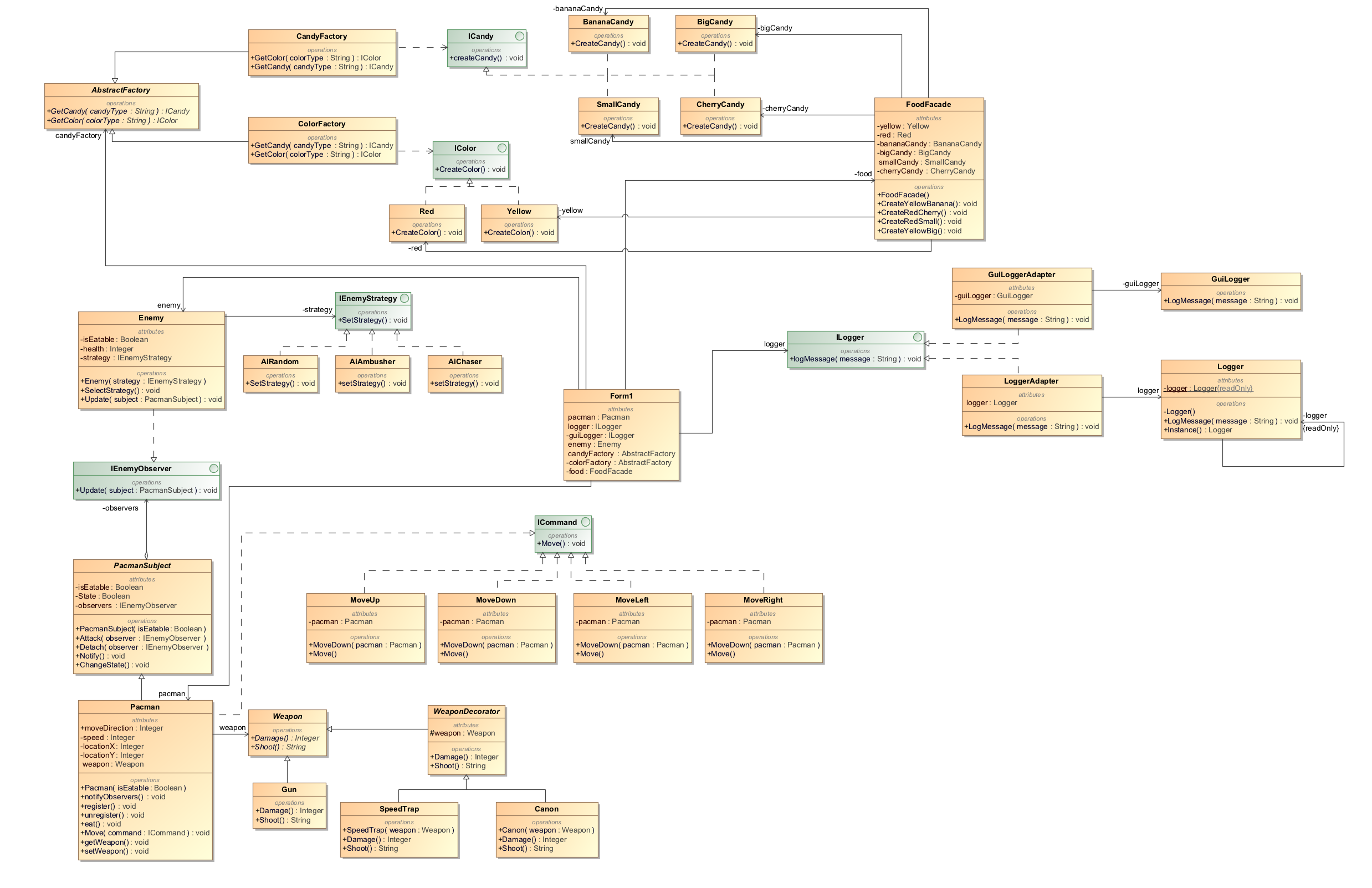


Pav. 9 Command programavimo šablonas

Kodas:

|  |
| --- |
| ICommand.cs |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  interface ICommand  {  void Move();  }  } |
| MoveUp.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class MoveUp : ICommand  {  private Pacman pacman;  public MoveUp(Pacman pacman)  {  this.pacman = pacman;  }  public void Move()  {  pacman.moveDirection = 0;  Debug.WriteLine("Pacman changed direction to: up");  }  }  } |
| MoveRight.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class MoveRight : ICommand  {  private Pacman pacman;  public MoveRight(Pacman pacman)  {  this.pacman = pacman;  }  public void Move()  {  pacman.moveDirection = 1;  Debug.WriteLine("Pacman changed direction to: right");  }  }  } |
| MoveLeft.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class MoveLeft : ICommand  {  private Pacman pacman;  public MoveLeft(Pacman pacman)  {  this.pacman = pacman;  }  public void Move()  {  pacman.moveDirection = 3;  Debug.WriteLine("Pacman changed direction to: left");  }  }  } |
| MoveDown.cs |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Pacman.Classes  {  class MoveDown : ICommand  {  private Pacman pacman;  public MoveDown(Pacman pacman)  {  this.pacman = pacman;  }  public void Move()  {  pacman.moveDirection = 2;  Debug.WriteLine("Pacman changed direction to: down");  }  }  } |

# 1 ir 2 Labaratorinių darbų klasių diagramos



Pav. 9 Bendra UML diagrama

# Išvados

Programos įgyvendinimui panaudoti programavimo šablonai: Observer, Strategy, Factory, Singleton, Facade, Command, Adapter ir Decorator. Kai kuriose vietose šie šablonai padėjo sumažinti programos sudėtingumą, padaryti kodą labiau skaitomą ir lengviau modifikuojamą.

Tačiau, supratome, kad perteklinis šablonų panaudojimas, apsunkina kodo rašymą ir pailgina įgyvendinimą.

1. Observers – šio šablono dėka yra lengviau pridėti bei pašalinti priešus, taip pat keivkieną kartą atnaujinus ekrano vaizdą paskaičiuoti priešų gyvybes.
2. Strategy – leido implementuoti skirtingus žaidimo lygius, kuriuos galime keisti programos veikimo metu.
3. Factory – padėjo lokalizuoti strategijos objekto kūrimą ir paslėpti jo kūrimo sudėtingumą nuo kliento.
4. Singleton – šis šablonas padeda užtikrinti, kad vieno žaidimo metu bus prieinamas tik vienas žaidimo objektas.
5. Facade – padeda paslėpti nereikalingą sudėtingumą ir funkcionalumą nuo kliento, atskleisdamas tik jam reikalingus metodus.
6. Command – leido pakeisti „string“ tipo komandas į „strongly typed“ objektus.
7. Adapter – šis šablonas padeda lengvai praplėsti objekto funkcionalumą. Kadangi nežinome kokį patobulinimą po kokio pasirinks žaidėjas, tai kiekvieno patobulinimo metu, mes tiesiog apgaubiama jau esantį objektą, kad neprarasti senų patobulinimu, bet pridėti ir naujus.
8. Decorator – leido nekeičiant jau esamos „spaceship“ klasės pridėti prie jos papildomą funkcionalumą