KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

OBJEKTINIO PROGRAMAVIMO PROJEKTAVIMAS (Design Patterns) (T120B516)

Laboratorinis darbas Nr. 1, 2
„Begalinis bėgikas“

Atliko: IFF – 4/1 gr. Studentas
Audrius Andrijaitis
Miglė Beresinevičiūtė
Matas Balčaitis
Priėmė: Dėst. A. Ušaniov

KAUNAS 2017

Turinys

Paveikslėlių sąrašas

Projekto aprašymas

Mūsų komanda siekdama įgyvendinti projektą bei išpildyti visus reikalavimus nusprendė kurti žaidimo „Endless Runner“ analogą. Pasirinkome būtent šį žaidimą dėl jo lengvo iš plečiamumo bei lengvai perkandamos pagrindinės žaidimo idėjos, kuri patiko visiems komandos nariams.

Mūsų tikslas sukurti „Endless Runner“ žaidimo varikliuką nerealizuojant grafinės sąsajos – žaidimo tęstinumą bei įvykius bus galima stebėti per CLI (valdymas vyks per komandinę eilutę).

Mūsų tikslas įgyvendinti pagrindinį žaidimo funkcionalumą bei pridėti kelis naujus patobulinimus (objektų rinkimas, kliūčių griovimas).

Funkciniai projekto reikalavimai

1. Žaidėjo gyvavimo trukmė begalinė arba iki pirmo susidūrimo su kliūtimi;
2. Žaidėjas gali rinkti objektus, kurie jam suteikia papildomų galių arba jas sumažina;
3. Žaidėjas gali atlikti tokius veiksmus kaip šokti, tūpti bei trenkti;
4. Žemėlapio generavimas automatizuotas – objektų pozicijos atsitiktinės, tačiau su realizuotomis apsaugomis (turi egzistuoti minimalus atstumas tarp objektų);
5. Papildomų objektų generavimas automatizuotas.

Projekto realizacija

Projektavimo modelių sąrašas

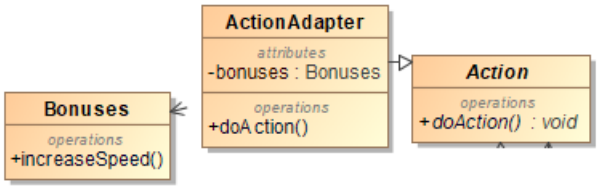
Projekte panaudojome šiuos projektavimo modelius:

- Elgsenos (behavior):
  - Strategy pattern
  - Template method
  - State
  - Chain of responsibility
  - Interpreter
  - Mediator
  - Memento
- Kūrimo (creational):
  - Singleton pattern
  - Factory pattern
  - Abstract factory pattern
- Struktūriniai (structural):
  - Facade pattern
  - Decorator pattern
  - Bridge
  - Proxy
  - Flyweight

Projektavimo modelių pritaikymas

Adapter pattern

Šį projektavimo modelį pritaikėme apjungti žaidėjo galimybę sustiprinti savo savybes (greitį) su objektais, kurių paėmimas gali atitinkamai sustiprinti ar susilpninti jas. Žemėlapyje atsitiktinai atsiranda obuoliai ir persikai (objektai), juos „paėmus“ atitinkamai paveikiama žaidėjo greitis. Poveikio stiprumas priklauso nuo objekto spalvos. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant adapter pattern apjungti šie objektai su poveikio metodais.



1 Pav. Adapter pattern pritaikymas

```
namespace EndlessRunner
{
    2 references | audand1, 11 hours ago | 1 author, 1 change
    class ActionAdapter : Action
    {
        private Bonuses bonuses;
        1 reference | audand1, 11 hours ago | 1 author, 1 change
        public ActionAdapter(Bonuses bonuses) {
            this.bonuses = bonuses;
        }
        5 references | audand1, 11 hours ago | 1 author, 1 change
        public override void doAction()
        {
            bonuses.increaseSpeed();
        }
    }
}
```

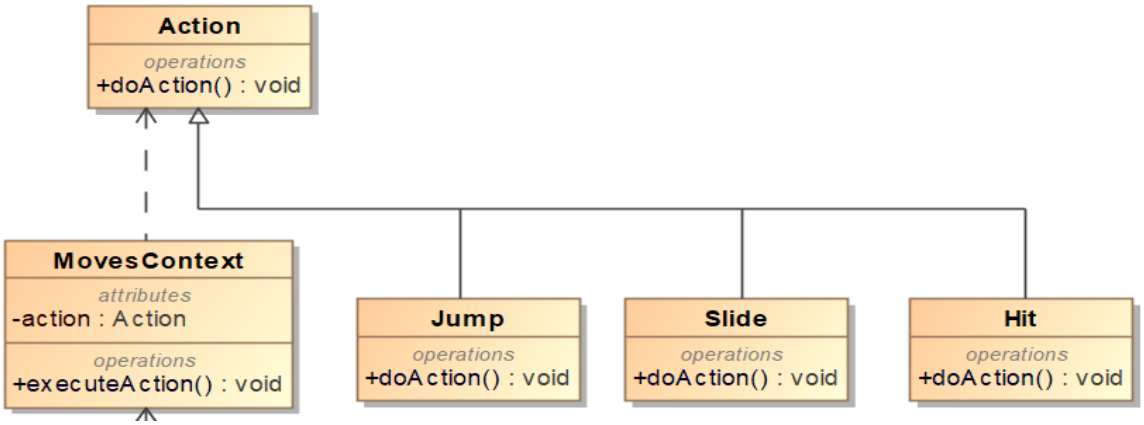
2 Pav. Adapter pattern realizacijos kodas

```
namespace EndlessRunner
{
    3 references | audand1, 11 hours ago | 1 author, 1 change
    class Bonuses
    {
        1 reference | audand1, 11 hours ago | 1 author, 1 change
        public void increaseSpeed()
        {
            Console.WriteLine("increase speed");
        }
    }
}
```

3 Pav. Adapter pattern realizacijos kodas (bonuses realizacija)

Strategy pattern

Šį projektavimo modelį pritaikėme žaidėjo veiksmų (šokti, čiuožti, trenkti) realizacijai. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių mūsų projektui ir geriausiai atspindi mūsų norimo funkcionalumo realizaciją. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant stategy pattern realizuoti veiksmai, kurios gali atlikti žaidėjas.



4 Pav. Strategy pattern pritaikymas

```
namespace EndlessRunner
{
    class MovesContext
    {
        private Action action;
        public MovesContext(Action action)
        {
            this.action = action;
        }
        public void executeAction()
        {
            action.doAction();
        }
    }
}
```

5 Pav. Strategy pattern realizacijos kodas

```
namespace EndlessRunner
{
    abstract public class Action
    {
        public abstract void doAction();
        public Action() { }
    }
}
```

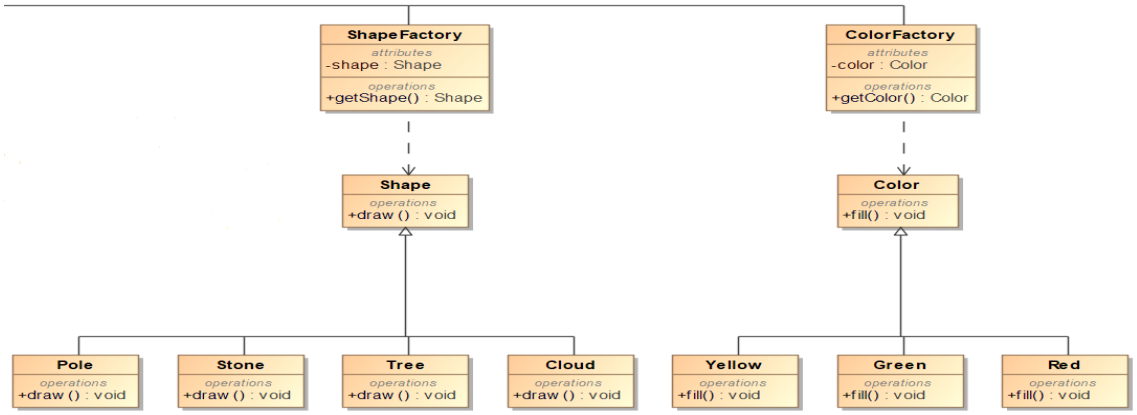
6 Pav. Strategy pattern realizacija (veiksmo realizacija)

```
namespace EndlessRunner
{
    class Jump : Action
    {
        public override void doAction()
        {
            System.Console.WriteLine("Jump");
        }
    }
}
```

7 Pav. Strategy pattern realizacija (6uolio realizacija)

Factory pattern

Šį projektavimo modelį pritaikėme objektų (kliūčių žemėlapyje) bei jų spalvų kūrimui. Pasirinkome būtent šį iš kūrimo modelių, nes tai vienas populiariausių šio tipo modelių bei atitinka mūsų tikslą – galimybę, kurti objektus, kurių kūrimo logika neprieinama vartotojui. Žemiau pateiktame paveikslėlyje matyti kaip realizuotas factory pattern žemėlapio objektų kūrimui bei jų spalvų kūrimui.



8 Pav. Factory pattern pritaikymas

```
namespace EndlessRunner
{
    1 reference | audand1, 15 hours ago | 2 authors, 2 changes
    public class ColorFactory : AbstractFactory
    {
        2 references | Migle Beresineviciute, 1 day ago | 1 author, 1 change
        public override Shape getShape(string shape)
        {
            return null;
        }

        2 references | audand1, 15 hours ago | 2 authors, 2 changes
        public override Color getColor(string color)
        {
            if (color == null)
            {
                return null;
            }
            if (string.Equals(color.ToUpper(), "GREEN"))
            {
                return new Green();
            }
            else if (string.Equals(color.ToUpper(), "YELLOW"))
            {
                return new Yellow();
            }
            else if (string.Equals(color.ToUpper(), "RED"))
            {
                return new Red();
            }
            return null;
        }
    }
}
```

9 Pav. Factory pattern realizacijos kodas

```
namespace EndlessRunner
{
    8 references | audand1, 15 hours ago | 2 authors, 2 changes
    public abstract class Color
    {
        10 references | audand1, 15 hours ago | 2 authors, 2 changes
        abstract public void fill();
    }
}
```

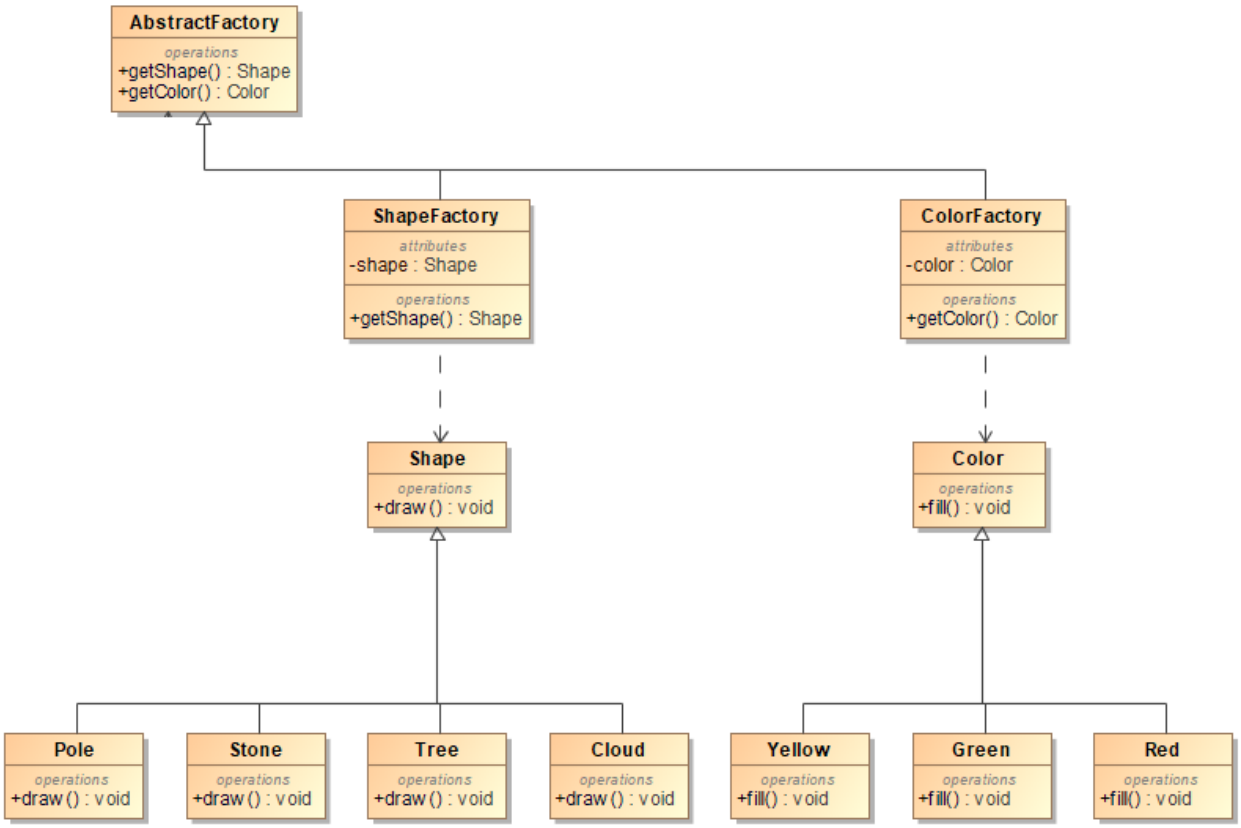
10Pav. Factory pattern realizacijos kodas (spalvos realizacija)

```
namespace EndlessRunner
{
    2 references | audand1, 11 hours ago | 2 authors, 3 changes
    public class Blue : Color
    {
        10 references | audand1, 11 hours ago | 2 authors, 3 changes
        public override void fill()
        {
            Console.Write("Blue - ");
        }
    }
}
```

11 Pav. Factory pattern realizacijos kodas (m4lynos spalvos realizacija)

Abstract factory pattern

Šį projektavimo modelį pritaikėme jau anksčiau minėtų factory pattern panaudojimo atveju apjungimui. Pasirinkome būtent šį kūrimo modelį, nes jo paskirtis ir yra apjungti keliems factory patterns. Žemiau pateiktame paveikslėlyje matyti kaip du fatory patterns (žemėlapio objektų bei spalvų) apjungiami naudojant abstract factory



12 Pav.Abstract factory pattern pritaikymas

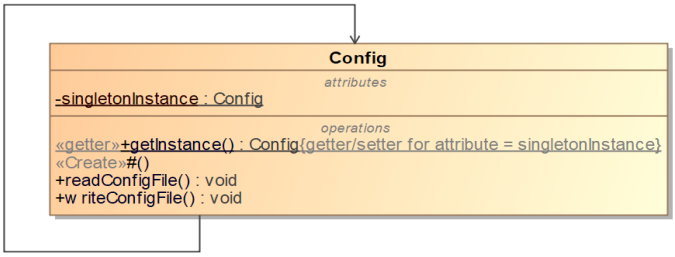
```
namespace EndlessRunner
{
    3 references | audand1, 15 hours ago | 2 authors, 2 changes
    public abstract class AbstractFactory
    {
        2 references | audand1, 15 hours ago | 2 authors, 2 changes
        abstract public Color getColor(string color);
        2 references | audand1, 15 hours ago | 2 authors, 2 changes
        abstract public Shape getShape(string shape);
    }
}
```

13 Pav. Abstract factory pattern realizacijos kodas

Singleton pattern

Šį projektavimo modelį pritaikėme konfigūracijų failo kūrimui. Singleton projektavimo modelis yra vienas paprasčiausių kūrimo modelių skirtų sukurti objektui, kuriam reikia užtikrinti, jog jis bus toks vienintelis. Būtent todėl ir pasirinkome šį projektavimo modelį konfigūracijų failo

kūrimui (projekte reikalingas tik vienas konfigūracijų failas). Žemiau pateiktame paveikslėlyje matyti kaip pritaikėme singleton projektavimo modelį savo konfigūracijų failo kūrimui.



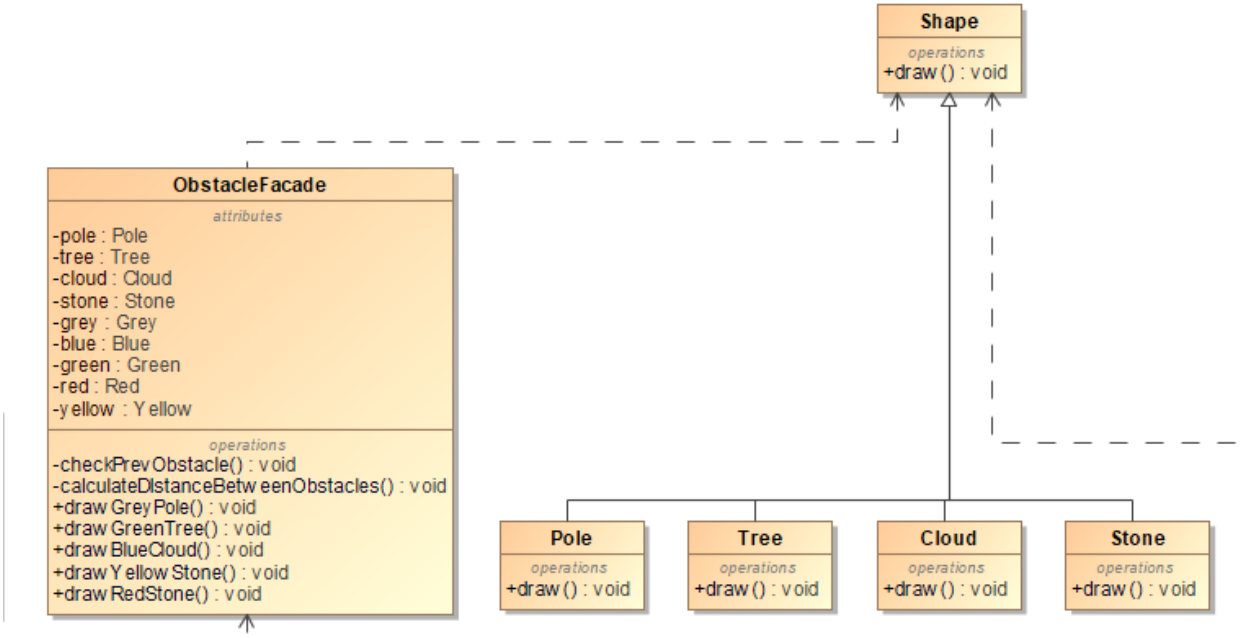
14 Pav. Sigletop pattern pritaikymas

```
namespace EndlessRunner
{
    class Config
    {
        private static Config singletonInstance;
        private Config() {}

        public static Config getInstance
        {
            get
            {
                if (singletonInstance == null)
                {
                    singletonInstance = new Config();
                }
                return singletonInstance;
            }
        }


        public void readConfigFile()
        {
            Console.WriteLine("read config file");
        }
        public void writeConfigFile()
        {
            Console.WriteLine("write config file");
        }
    }
}
```

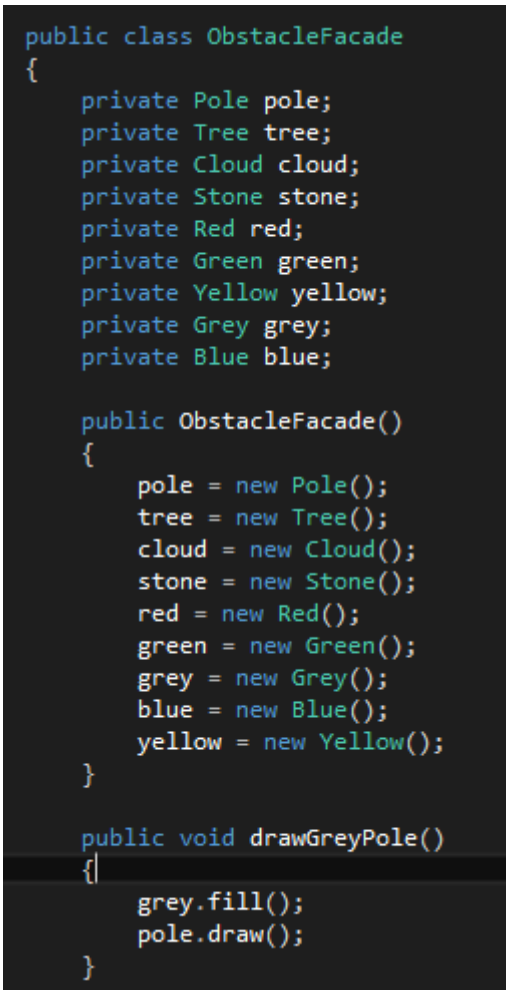
15 Pav. Sigleton pattern realizacijos kodas

Facade pattern

Šį projektavimo modelį pritaikėme kliūčių žemėlapyje atsiradimo logikai realizuoti. Mūsų nuomone, façade struktūrinis projektavimo modelis labiausiai atitiko mūsų poreikius, t.y. suteikia galimybę sudėtinga posistemę apibendrinti į vieną bendrą paprastą sąsaja. Tai suteikia mums galimybę paslėpti sudėtinga kliūčių generavimo logiką bei taip pagerinti projekto suprantamumą bei skaitomumą. Žemiau pateiktame paveikslėlyje matyti kaip pritaikėme façade projektavimo modelį kliūčių generavimo logikai apibendrinti.



16 Pav. Façade pattern pritaikymas

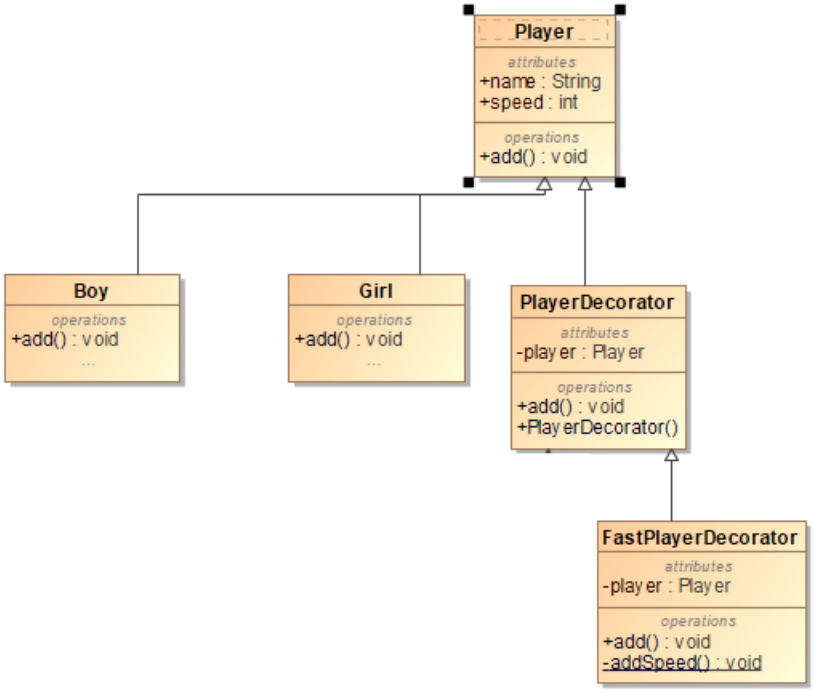


```
public class ObstacleFacade
{
    private Pole pole;
    private Tree tree;
    private Cloud cloud;
    private Stone stone;
    private Red red;
    private Green green;
    private Yellow yellow;
    private Grey grey;
    private Blue blue;

    public ObstacleFacade()
    {
        pole = new Pole();
        tree = new Tree();
        cloud = new Cloud();
        stone = new Stone();
        red = new Red();
        green = new Green();
        grey = new Grey();
        blue = new Blue();
        yellow = new Yellow();
    }

    public void drawGreyPole()
    {
        grey.fill();
        pole.draw();
    }
```

17 Pav.Facade pattern realizacijos kodas

Decorate pattern

Šį projektavimo modelį pritaikėme žaidėjo savybių (jėgos bei greičio) keitimui. Decorate pattern priklauso struktūrinių projektavimo modelių šeimai bei yra skirtas pridėti naują funkcionalumą jau egzistuojančiam objektui nepažeidžiant jo struktūros. Kadangi tai atitiko visus mūsų išsikeltus reikalavimus šiai posistemei, nusprendėme, kad decorate pattern yra būtent tai ko mums reikia. Žemiau pateiktame paveikslėlyje galima matyti decorate pattern pritaikymą žaidėjo savybių pridėjimui.



18 Pav. Decorate pattern pritaikymas

```
namespace EndlessRunner
{
    public abstract class Player
    {
        public string name;
        public int speed;
        public abstract void add();
    }
}
```

19Pav. Decorator pattern realizacijos kodas (žaidėjo realizacija)

```
namespace EndlessRunner
{
    class Boy: Player
    {
        public override void add()
        {
            Console.WriteLine("Player Boy");
        }
    }
}
```

20 Pav. Decorator pattern realizacija (žaidėjo lyties realizacija)

```
namespace EndlessRunner
{
    class PlayerDecorator: Player
    {
        protected Player decoratedPlayer;
        public PlayerDecorator() { }
        public PlayerDecorator(Player decoratedPlayer)
        {
            this.decoratedPlayer = decoratedPlayer;
        }

        public override void add()
        {
            decoratedPlayer.add();
        }
    }
}
```

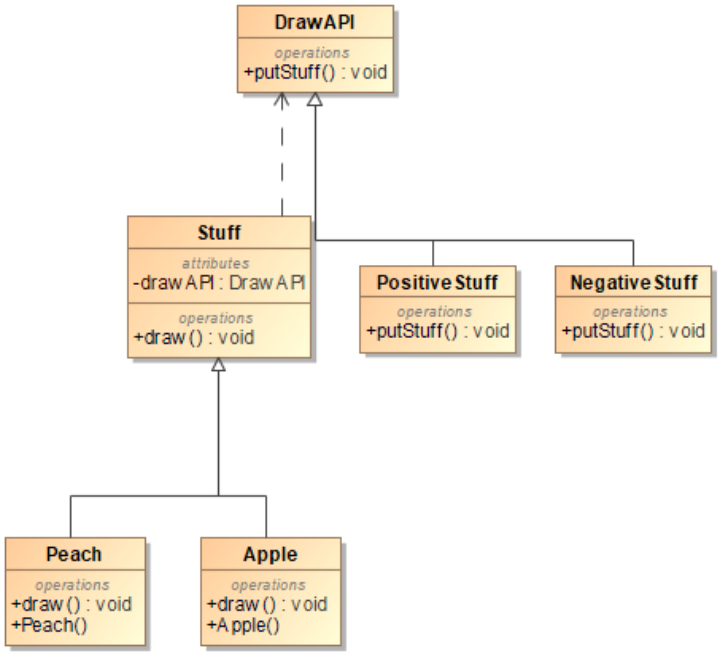
21 Pav. Decorator pattern realizacija (žaidėjo realizacija)

```
namespace EndlessRunner
{
    class FastPlayerDecorator : PlayerDecorator
    {
        public FastPlayerDecorator(Player decoratedPlayer)
        {
            this.decoratedPlayer = decoratedPlayer;
        }

        public override void add()
        {
            decoratedPlayer.add();
            addSpeed(decoratedPlayer,50);
        }

        private void addSpeed(Player player,int speed)
        {
            Console.WriteLine("Change speed: " + speed);
            player.speed += speed;
        }
    }
}
```

22 Pav. Decorator pattern realizacija (greito žaidėjo decorator)

Bridge pattern

Šį projektavimo modelį pritaikėme objektų, kuriuos gali rinkti žaidėjas kūrimui. Bridge projektavimo modelis taip pat priklauso struktūrinių projektavimo modelių šeimai ir yra skirtas implementacijos atskyrimui nuo abstrakcijos, kad abi dalys galėtų gyvuoti nepriklausomai. Būtent todėl nusprendėme, kad jis mums yra tinkamiausias šioje situacijoje.



23 Pav. Bridge pattern pritaikymas

```
namespace EndlessRunner
{
    public abstract class DrawAPI
    {
        public abstract  void putStuff();
        public DrawAPI() { }
    }
}
```

24 Pav. Bridge pattern realicazija (draw api)

```
namespace EndlessRunner
{
    abstract class Stuff
    {
        protected DrawAPI drawAPI;
        protected Stuff() { }
        protected Stuff(DrawAPI drawAPI)
        {
            this.drawAPI = drawAPI;
        }
        public abstract void draw();
    }
}
```

25 Pav. Bridge pattern realizacija (daiktų realizacija)

```
namespace EndlessRunner
{
     class Apple : Stuff
    {
      //  DrawAPI drawAPI;

        public Apple(DrawAPI drawAPI)
        {
            this.drawAPI = drawAPI;
        }

        public override void draw()
        {
            drawAPI.putStuff();
        }

    }
}
```

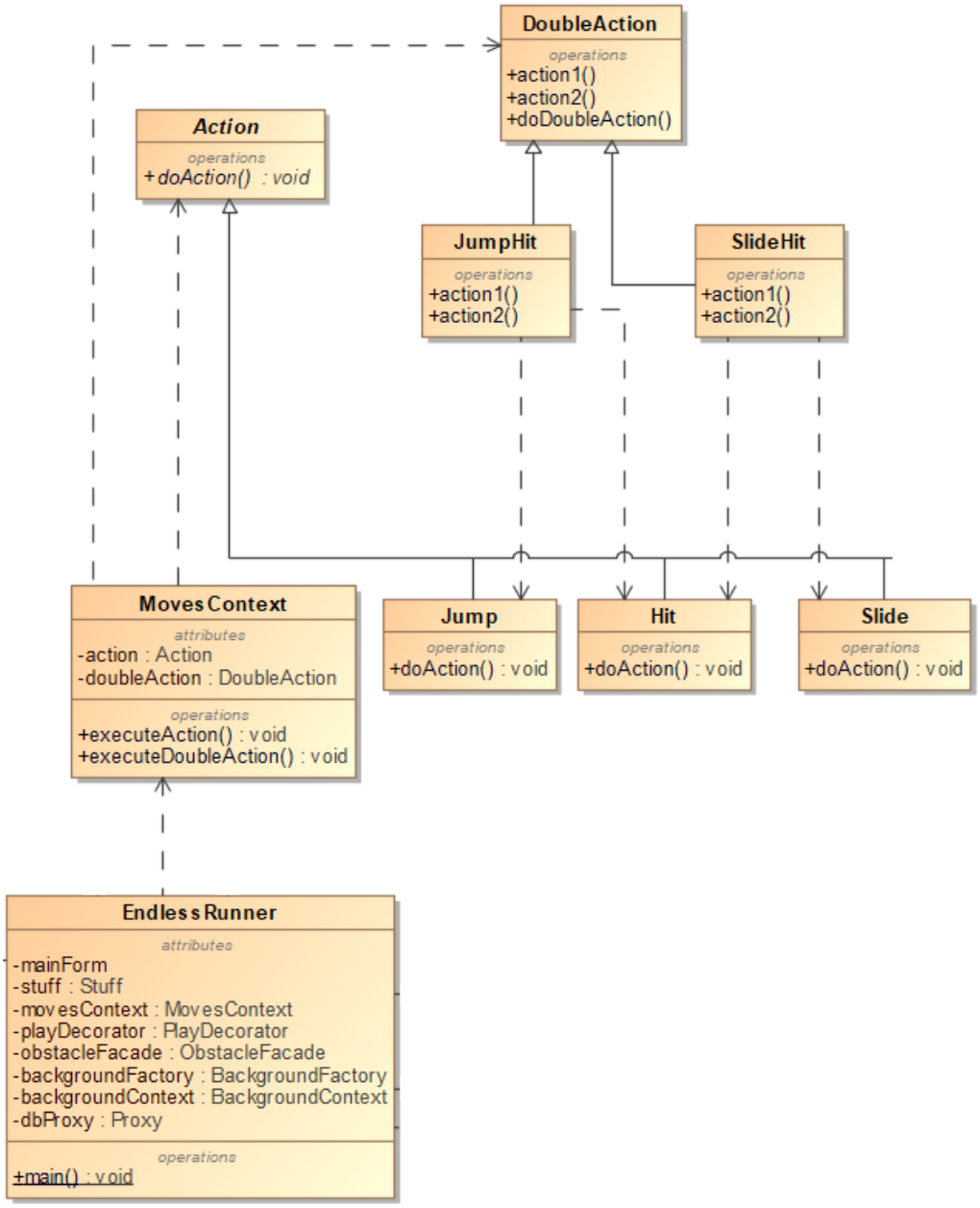
26 Pav. Bridge pattern realizacija (obuolio realizacija)

```
namespace EndlessRunner
{
    public class PositiveStuff:DrawAPI
    {
        public override void putStuff()
        {
            System.Console.WriteLine("Positive stuff");
        }
    }
}
```

27 Pav. Bridge pattern realizacija (teigiamo poveikio daiktų realizacija)

Template method pattern

Šį projektavimo modelį pritaikėme kombinuotų žaidėjo veiksmų (šokti ir trenkti, čiuožti ir trenkti) realizacijai. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių mūsų projektui ir geriausiai atspindi mūsų norimo funkcionalumo realizaciją. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant template method pattern realizuoti kombinuoti veiksmai, kurios gali atlikti žaidėjas.



28 Pav. Template method pattern pritaikymas

```
namespace EndlessRunner
{
    13 references | audand1, 2 hours ago | 1 author, 2 changes
    class MovesContext
    {
        private Action action;
        private DoubleAction doubleAction;
        4 references | audand1, 35 days ago | 1 author, 1 change
        public MovesContext(Action action)
        {
            this.action = action;
        }
        2 references | audand1, 2 hours ago | 1 author, 1 change
        public MovesContext(DoubleAction action)
        {
            this.doubleAction = action;
        }
        4 references | audand1, 35 days ago | 1 author, 1 change
        public void executeAction()
        {
            action.doAction();
        }
        2 references | audand1, 2 hours ago | 1 author, 1 change
        public void executeDoubleAction()
        {
            doubleAction.doDoubleAction();
        }
    }
}
```

29 Pav. Template method pattern realizacija (MovesContext realizacija)

```
namespace EndlessRunner
{
    4 references | audand1, 2 hours ago | 1 author, 1 change
    abstract class DoubleAction
    {
        3 references | audand1, 2 hours ago | 1 author, 1 change
        public abstract void action1();
        3 references | audand1, 2 hours ago | 1 author, 1 change
        public abstract void action2();

        1 reference | audand1, 2 hours ago | 1 author, 1 change
        public void doDoubleAction()
        {
            Console.WriteLine("-- Start double action ---");
            action1();
            action2();
            Console.WriteLine("-- Done double action ---");
        }
    }
}
```

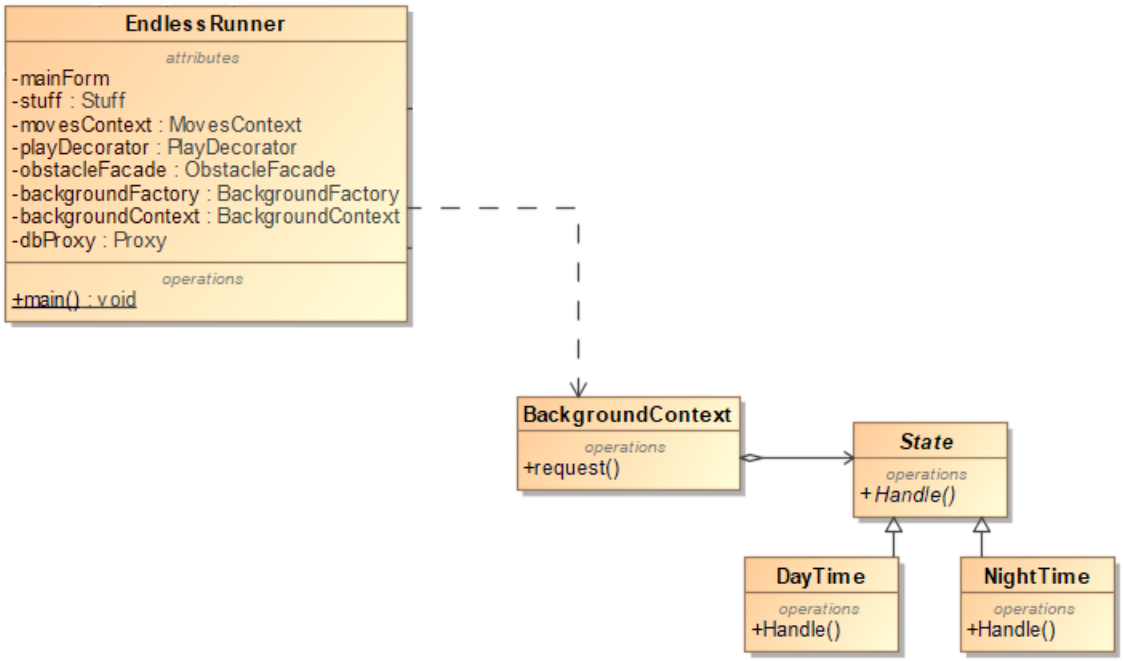
30Pav. Template method pattern realizacija (DoubleAction realizacija)

```
namespace EndlessRunner
{
    1 reference | audand1, 2 hours ago | 1 author, 1 change
    class JumpHit : DoubleAction
    {
        3 references | audand1, 2 hours ago | 1 author, 1 change
        public override void action1()
        {
            Jump jump = new Jump();
            jump.doAction();
        }
        3 references | audand1, 2 hours ago | 1 author, 1 change
        public override void action2()
        {
            Hit hit = new Hit();
            hit.doAction();
        }
    }
}
```

31 Pav. Template method pattern realizacija (JumpHit realizacija)

State pattern

Šį projektavimo modelį pritaikėme paros meto nustatymo (diena arba naktis) realizacijai. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių šios projekto dalies realizacijai ir geriausiai atspindi mūsų norimo funkcionalumo realizaciją. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant state pattern realizuoti paros meto nustatymui.



32 Pav.State pattern pritaikymas

```
namespace EndlessRunner
{
    2 references | audand1, 2 hours ago | 1 author, 1 change
    class DayTime : State
    {
        3 references | audand1, 2 hours ago | 1 author, 1 change
        public override void Handle(BackgroundContext context)
        {
            context.State = new NightTime();
        }
    }
}
```

33 Pav. State pattern realizacija (DayTime realizacija)

```
namespace EndlessRunner
{
    6 references | audand1, 2 hours ago | 1 author, 1 change
    class BackgroundContext
    {
        private State _state;

        1 reference | audand1, 2 hours ago | 1 author, 1 change
        public BackgroundContext(State state)
        {
            this._state = state;
        }

        2 references | audand1, 2 hours ago | 1 author, 1 change
        public State State
        {
            get { return _state; }
            set
            {
                _state = value;
                Console.WriteLine("State: " +  _state.GetType().Name);
                }
            }

        3 references | audand1, 2 hours ago | 1 author, 1 change
        public void Request()
        {
            _state.Handle(this);
        }
    }
}
```

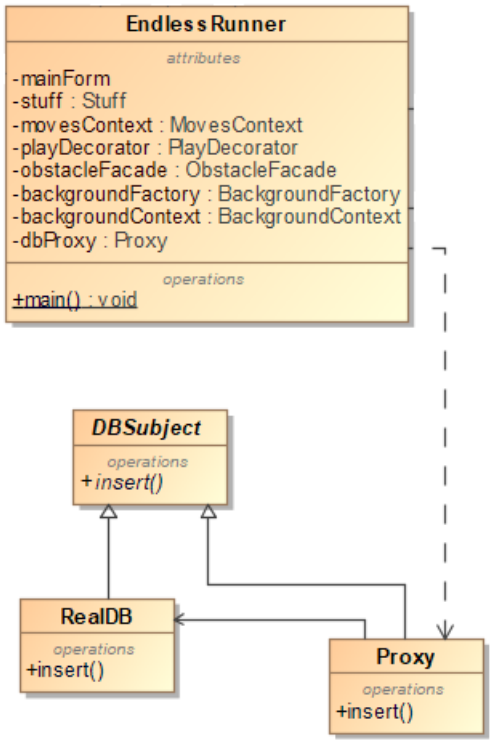
34 Pav. State pattern realizacija (BackgroundContet realizacija)

```
namespace EndlessRunner
{
    5 references | audand1, 2 hours ago | 1 author, 1 change
    abstract class State
    {
        3 references | audand1, 2 hours ago | 1 author, 1 change
        public abstract void Handle(BackgroundContext context);
    }
}
```

35 Pav. State pattern realizacija (State metodo realizacija)

Proxy pattern

Šį projektavimo modelį pritaikėme duomenų į duombazę įrašymui. Mūsų nuomone, struktūrinis projektavimo modelis proxy labiausiai atitiko mūsų poreikius, t.y. galimybė bendrauti su duomenų baze išvengiant tiesioginių kreipinių iš kliento. Žemiau pateiktame paveikslėlyje matyti kaip pritaikėme proxy projektavimo modelį bendravimo su duomenų baze realizacijai.



36 Pav. Proxy pattern pritaikymas

```
namespace EndlessRunner
{
    2 references | audand1, 6 days ago | 1 author, 1 change
    class Proxy : DBSubject
    {
        private RealDB realDB;

        4 references | audand1, 6 days ago | 1 author, 1 change
        public override void insert(string text)
        {
            if (realDB == null)
            {
                realDB = new RealDB();
            }

            realDB.insert(text);

        }
    }
}
```

37Pav. Proxy pattern realizacija (Proxy metodo realizacija)

```
namespace EndlessRunner
{
    2 references | audand1, 6 days ago | 1 author, 1 change
    abstract class DBSubject
    {
        4 references | audand1, 6 days ago | 1 author, 1 change
        public abstract void insert(string text);
    }
}
```

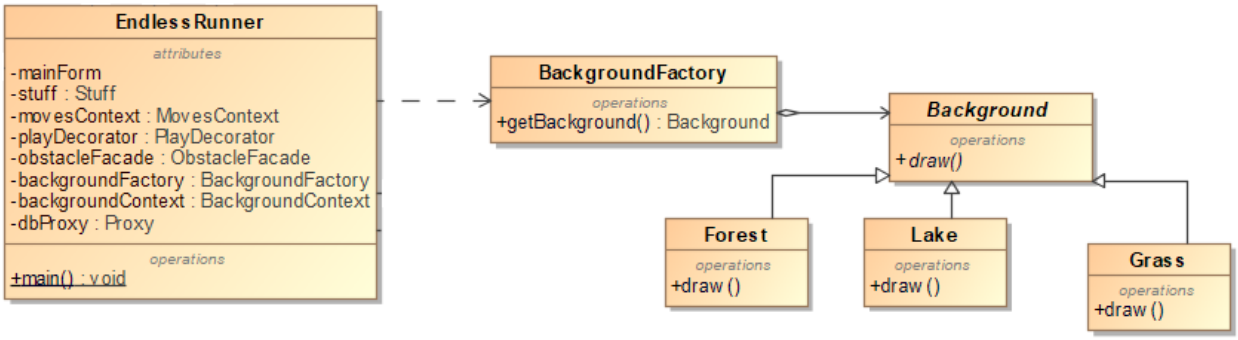
38Pav. Proxy realizacija (DbSubject realizacija)

```
namespace EndlessRunner
{
    2 references | audand1, 6 days ago | 1 author, 1 change
    class RealDB: DBSubject
    {
        4 references | audand1, 6 days ago | 1 author, 1 change
        public override void insert(string text)
        {
            Console.WriteLine("the text (" + text + ") is inserted in the database");
        }
    }
}
```

39 Pav. Proxy realizacija (RealDB realizacija)

Flyweight pattern

Šį projektavimo modelį pritaikėme antrame plane esančių objektų kūrimui. Mūsų nuomone, struktūrinis projektavimo modelis flyweight labiausiai atitiko mūsų poreikius, t.y. galimybė kurti daug vienodų objektų tausojant atmintį. Žemiau pateiktame paveikslėlyje matyti kaip pritaikėme flyweight projektavimo modelį antraplanių objektų kūrimui.



40 Pav. Flyweight pattern pritaikymas

```
namespace EndlessRunner
{
    2 references | audand1, 2 hours ago | 1 author, 1 change
    class BackgroundFactory
    {
        private Hashtable background = new Hashtable();


        6 references | audand1, 2 hours ago | 1 author, 1 change
        public Background GetBackground(string backgroundType)
        {
            Background b = (Background)background[backgroundType];

            if (b == null)
            {
                if (backgroundType.Equals("Lake"))
                {
                    b = new Lake();
                }
                else if (backgroundType.Equals("Grass"))
                {
                   b = new Grass();
                }
                else if (backgroundType.Equals("Forest"))
                {
                    b = new Forest();
                }

                background.Add(backgroundType,b);
            }
            return b;
        }
    }
}
```

41 Pav. Flyweight realizacija (BackgroundFactory realizacija)

```
namespace EndlessRunner
{
    7 references | audand1, 2 hours ago | 1 author, 1 change
    abstract class Background
    {
        9 references | audand1, 2 hours ago | 1 author, 1 change
        public abstract void draw();
    }
}
```

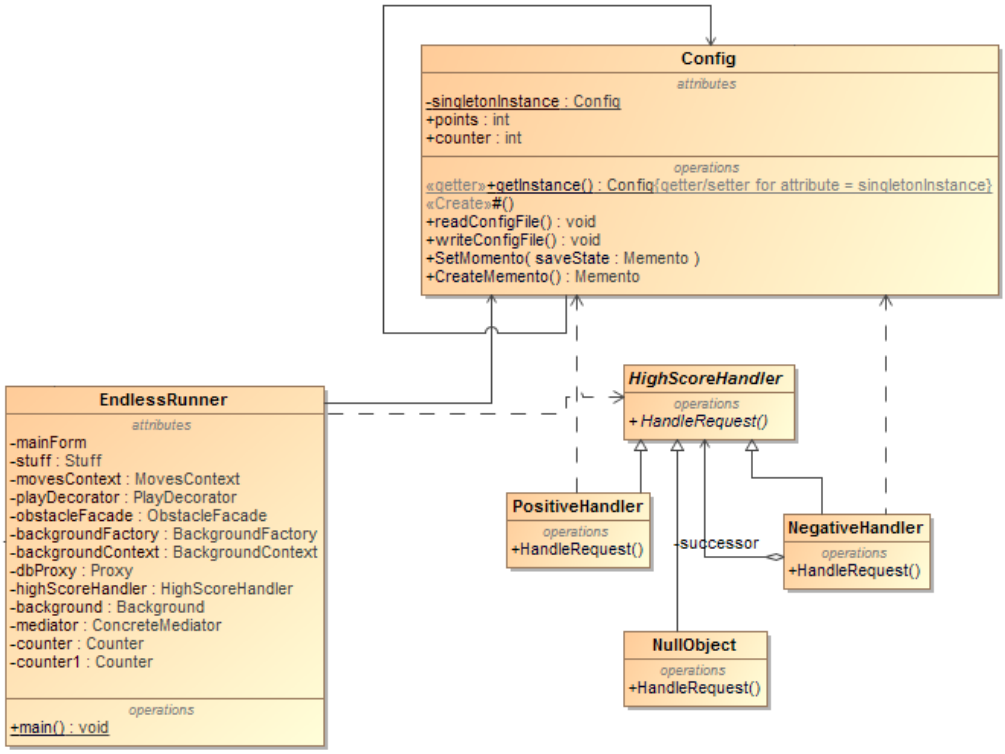
42 Pav. Flyweight realizacija (Background realizacija)

```
namespace EndlessRunner
{
    2 references | audand1, 2 hours ago | 1 author, 1 change
    class Lake : Background
    {
        1 reference | audand1, 2 hours ago | 1 author, 1 change
        public Lake()
        {
            Console.WriteLine("init LAKE object");
        }
        9 references | audand1, 2 hours ago | 1 author, 1 change
        public override void draw()
        {
            Console.WriteLine("draw LAKE ");
        }
    }
}
```

43 Pav. Flyweight realizacija (Lake realizacija)

Chain of responsibility pattern

Šį projektavimo modelį pritaikėme žaidėjo surinktų taškų skaičiavimui. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių „highscores“ skaičiavimui. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant chain of responsibility pattern realizuoti paros meto nustatymui.



44 Pav. Chain of responsibility pattern pritiakymas

```
namespace EndlessRunner
{
    6 references | audand1, 22 hours ago | 1 author, 1 change
    abstract class HighScoreHandler
    {
        protected HighScoreHandler successor;
        1 reference | audand1, 22 hours ago | 1 author, 1 change
        public void SetSuccessor(HighScoreHandler successor)
        {
            this.successor = successor;
        }
        5 references | audand1, 22 hours ago | 1 author, 1 change
        public abstract void HandleRequest(int request);
    }
}
```

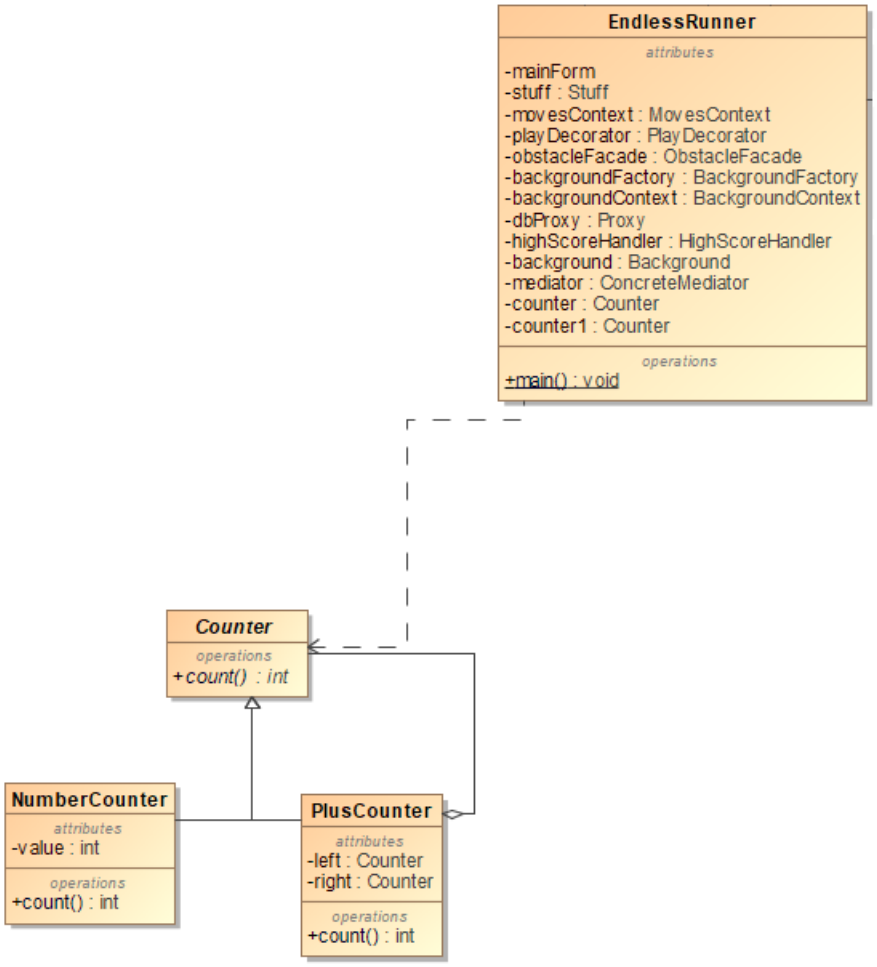
45 Pav. Chain of responsibilityt realizacija (HighscoreHandler realizacija)

```
namespace EndlessRunner
{
    1 reference | audand1, 22 hours ago | 1 author, 1 change
    class NegativeHandler : HighScoreHandler
    {
        5 references | audand1, 22 hours ago | 1 author, 1 change
        public override void HandleRequest(int request)
        {
            if (request < 0)
            {
                Console.WriteLine("{0} handled request {1}",
                    this.GetType().Name, request);
                Config.getInstance.points += request;
            }
            else if (successor != null)
            {
                successor.HandleRequest(request);
            }
        }
    }
}
```

46 Pav. Chain of responsibility realizacija (NegativeHandler realizacija)

Interpreter pattern

Šį projektavimo modelį pritaikėme „rekordų“ (highscores) skaičiavimui. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių „highscores“. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant chain of responsibility pattern realizuoti paros meto nustatymui.



47 Pav. Interpreter pattern pritaikymas

```
namespace EndlessRunner
{
    8 references | audand1, 22 hours ago | 1 author, 1 change
    abstract class Counter
    {
        5 references | audand1, 22 hours ago | 1 author, 1 change
        public abstract int count();
    }
}
```

48 Pav. Interpreter realizacija (Counter realizacija)

```
namespace EndlessRunner
{
    2 references | audand1, 22 hours ago | 1 author, 1 change
    class PlusCounter : Counter
    {
        private Counter left, right;

        1 reference | audand1, 22 hours ago | 1 author, 1 change
        public PlusCounter(Counter _left, Counter _right)
        {
            this.left = _left;
            this.right = _right;
        }

        5 references | audand1, 22 hours ago | 1 author, 1 change
        public override int count()
        {
            return left.count() + right.count();
        }
    }
}
```

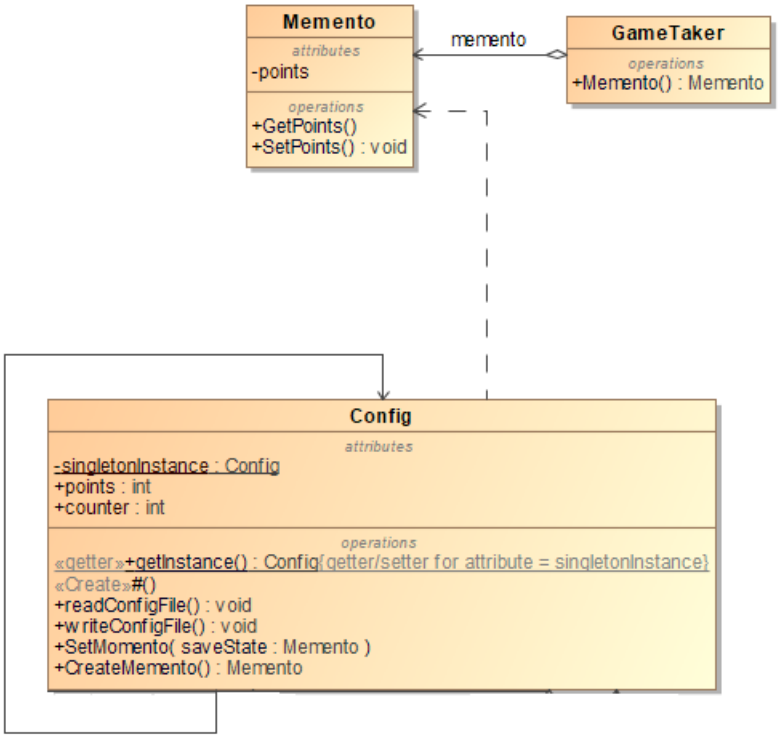
49 Pav. Interpreter realizacija (PlusCounter realizacija)

```
namespace EndlessRunner
{
    3 references | audand1, 22 hours ago | 1 author, 1 change
    class NumberCounter : Counter
    {
        private int value;
        2 references | audand1, 22 hours ago | 1 author, 1 change
        public NumberCounter(int value)
        {
            this.value = value;
        }
        5 references | audand1, 22 hours ago | 1 author, 1 change
        public override int count()
        {
            return value;
        }
    }
}
```

50 Pav. Interpreter realizacija (NumberCounter realizacija)

Memento pattern

Šį projektavimo modelį pritaikėme taškų išsaugojimui. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių mūsų užduoties realizacijai. Pagrindinis tikslas – išsaugoti turimus taškus, tam, kad žaidėjui panaudojus antrą gyvybę ar pan. būtų galima testi žaidimą toliau, o ne pradėti vėl nuo nulio. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant memento pattern realizuoti taškų kiekio išsaugojimui.



51 Pav. Memento pattern pritaikymas

```
namespace EndlessRunner
{
    2 references | audand1, 22 hours ago | 1 author, 1 change
    class GameTaker
    {
        private Memento _memento;
        2 references | audand1, 22 hours ago | 1 author, 1 change
        public Memento Memento
        {
            set { _memento = value; }
            get { return _memento; }
        }
    }
}
```

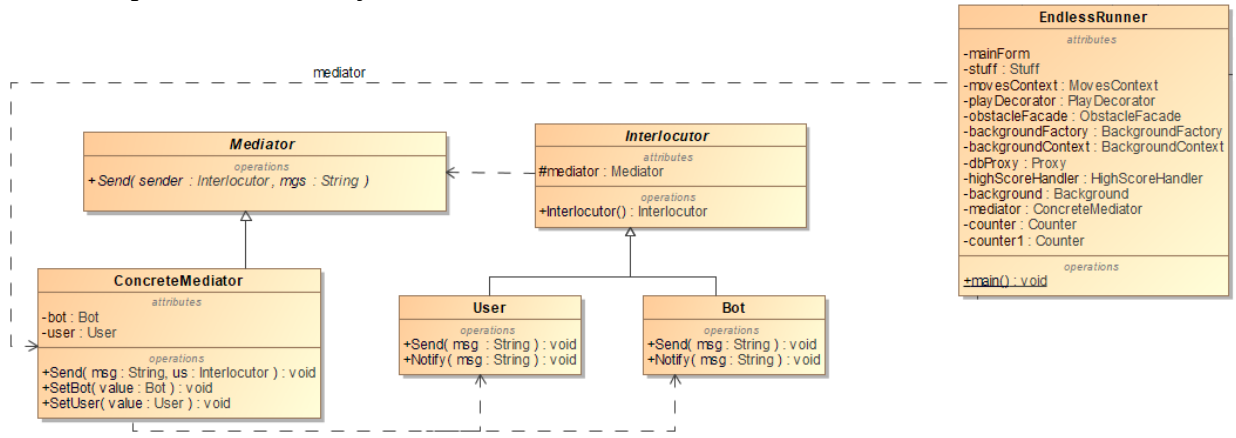
52Pav. Memento realizacija (GameTaker realizacija)

```
namespace EndlessRunner
{
    6 references | audand1, 22 hours ago | 1 author, 1 change
    class Memento
    {
        private int points;
        1 reference | audand1, 22 hours ago | 1 author, 1 change
        public Memento(int state)
        {
            this.points = state;
        }

        1 reference | audand1, 22 hours ago | 1 author, 1 change
        public int GetPoints()
        {
            return points;
        }
        0 references | audand1, 22 hours ago | 1 author, 1 change
        public void SetPoints(int state)
        {
            this.points = state;
        }
    }
}
```

53 Pav. Memento realizacija

Mediator pattern

Šį projektavimo modelį pritaikėme „rekordų“ (highscores) skaičiavimui. Pasirinkome būtent šį projektavimo modelį, kadangi, mūsų nuomone, jis tinkamiausias iš elgsenos modulių „highscores“. Žemiau pateiktame paveikslėlyje matyti kaip panaudojant chain of responsibility pattern realizuoti paros meto nustatymui.



54 Pav. Mediator pattern pritaikymas

```
namespace EndlessRunner
{
    5 references | audand1, 23 hours ago | 1 author, 1 change
    abstract class Mediator
    {
        3 references | audand1, 23 hours ago | 1 author, 1 change
        public abstract void Send(Interlocutor sender, string message);
    }
}
```

55 Pav. Mediator realizacija

```
namespace EndlessRunner
{
    5 references | audand1, 23 hours ago | 1 author, 1 change
    abstract class Interlocutor
    {
        protected Mediator mediator;
        2 references | audand1, 23 hours ago | 1 author, 1 change
        public Interlocutor(Mediator _mediator)
        {
            this.mediator = _mediator;
        }
    }
}
```

56 Pav. Mediator realizacija (Interlocutor realizacija)

```
namespace EndlessRunner
{
    2 references | audand1, 23 hours ago | 1 author, 1 change
    class ConcreteMediator : Mediator
    {
        private Bot bot;
        private User user;

        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public void SetBot(Bot value)
        {
            bot = value;
        }
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public void SetUser(User value)
        {
            user = value;
        }

        3 references | audand1, 23 hours ago | 1 author, 1 change
        public override void Send(Interlocutor sender, string message)
        {
            if (sender.Equals(user))
            {
                bot.Notify(message);
            }else if (sender.Equals(bot))
            {
                user.Notify(message);
            }
        }
    }
}
```

57 Pav. Mediator realizacija (ConcreteMediator realizacija)

```
namespace EndlessRunner
{
    5 references | audand1, 23 hours ago | 1 author, 1 change
    class User : Interlocutor
    {
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public User(Mediator mediator)
            : base(mediator)
        {

        }
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public void Send(string msg)
        {
            mediator.Send(this, msg);
        }
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public void Notify(string msg)
        {
            Console.WriteLine("User gets msg: " + msg);
        }

    }
}
```

58Pav. Mediator realizacija (User realizacija)

```
namespace EndlessRunner
{
    5 references | audand1, 23 hours ago | 1 author, 1 change
    class Bot : Interlocutor
    {
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public Bot(Mediator mediator)
            : base(mediator)
        {

        }
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public void Send(string msg)
        {
            mediator.Send(this, msg);
        }
        1 reference | audand1, 23 hours ago | 1 author, 1 change
        public void Notify(string msg)
        {
            Console.WriteLine("Bot gets msg: " + msg);
        }

    }
}
```

59 Pav. Mediator realizacija (Bot realizacija)

Išvados

Panaudoję įvairius elgsenos, kūrimo bei struktūrinius projektavimo modelius pastebėjome jų teikiamą naudą, kuomet projektas tampa lengviau suprantamas, o svarbiausia lengviau koreguojamas.