KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

T120B516 Objektinis Programų Projektavimas

Žaidimas "NEMARE"

IFF-6/11 Nerijus Dulkė
IFF-6/11 Regijus Borodinas
IFF-6/11 Marius Teleiša

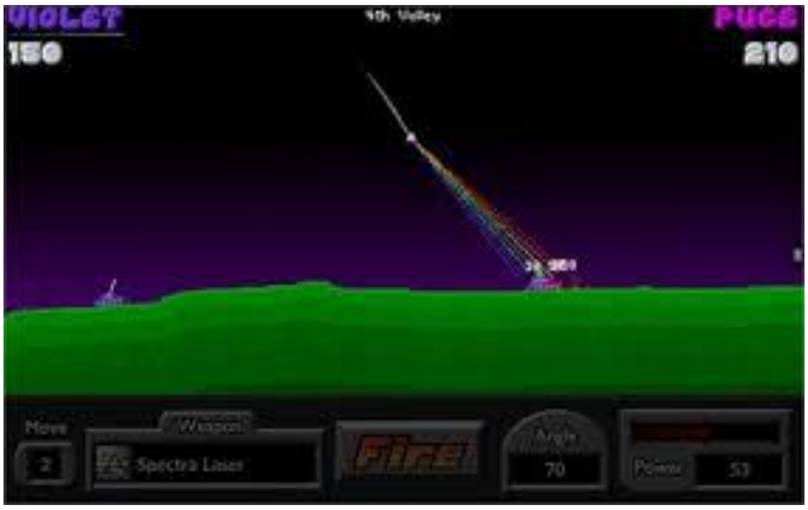
Data: 2019.11.08

KAUNAS
2019

Turinys

1. Žaidimo aprašymas

Tai yra ėjimais pagrįstas žaidimas, kurio metu du žaidėjai valdo skirtingus tankus, kurių tikslas sunaikinti priešininko tanką. Žaidime yra skirtingi šoviniai, žemėlapiai ir pastiprinimai. Žaidėjai savo ėjimo metu gali judinti savo tanką žemėlapio erdvėje ir šaudyti pasirinktus šovinius nurodyta trajektorija bei galia. Žaidimas pasibaigia kai baigiasi duotas žaidimo laikas arba vienas iš žaidėjų praranda visus gyvybės taškus. Laimi tas žaidėjas kuris žaidimo pabaigoje turi daugiau gyvybės taškų.



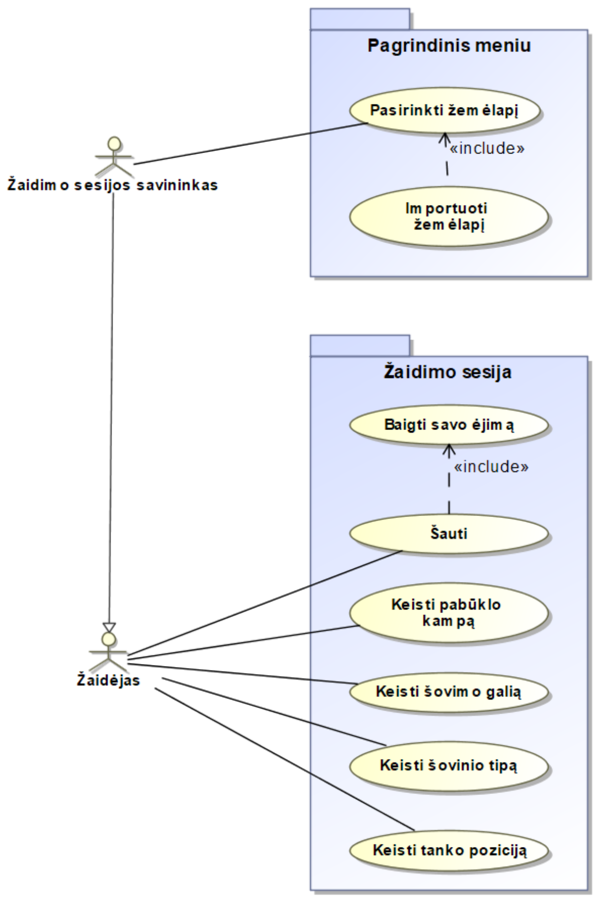
pav. 1 Žaidimo grafinis pavyzdys

2. Reikalavimai

- Lygis (žemėlapis) pasirenkamas prieš prasidedant žaidimui.
    - Galimybė pasirinkti vieną iš jau sukurtų ir žaidime esančių žemėlapių.
    - Galimybė importuoti savo sukurtą žemėlapį (PNG formatu).
        - Žemėlapis sudarytas iš juodos ir baltos spalvos (juoda spalva simbolizuoja žemę, ant kurios gali važinėti tankai, balta – orą, per kurį gali keliauti šovinys).
- Kiekvienas žaidėjas pradeda žaidimą su 100 gyvybės taškų.
- Žaidimas trunka 15 minučių.
- Žaidimas vyksta ėjimais pradedant nuo atsitiktinai pasirinkto žaidėjo.
    - Vieno ėjimo trukmė – 30 sekundžių.
    - Ėjimo metu žaidėjas gali pajudinti savo tanką ir šauti pasirinkto tipo šovinį žaidėjo nustatyta kryptimi ir stiprumu.

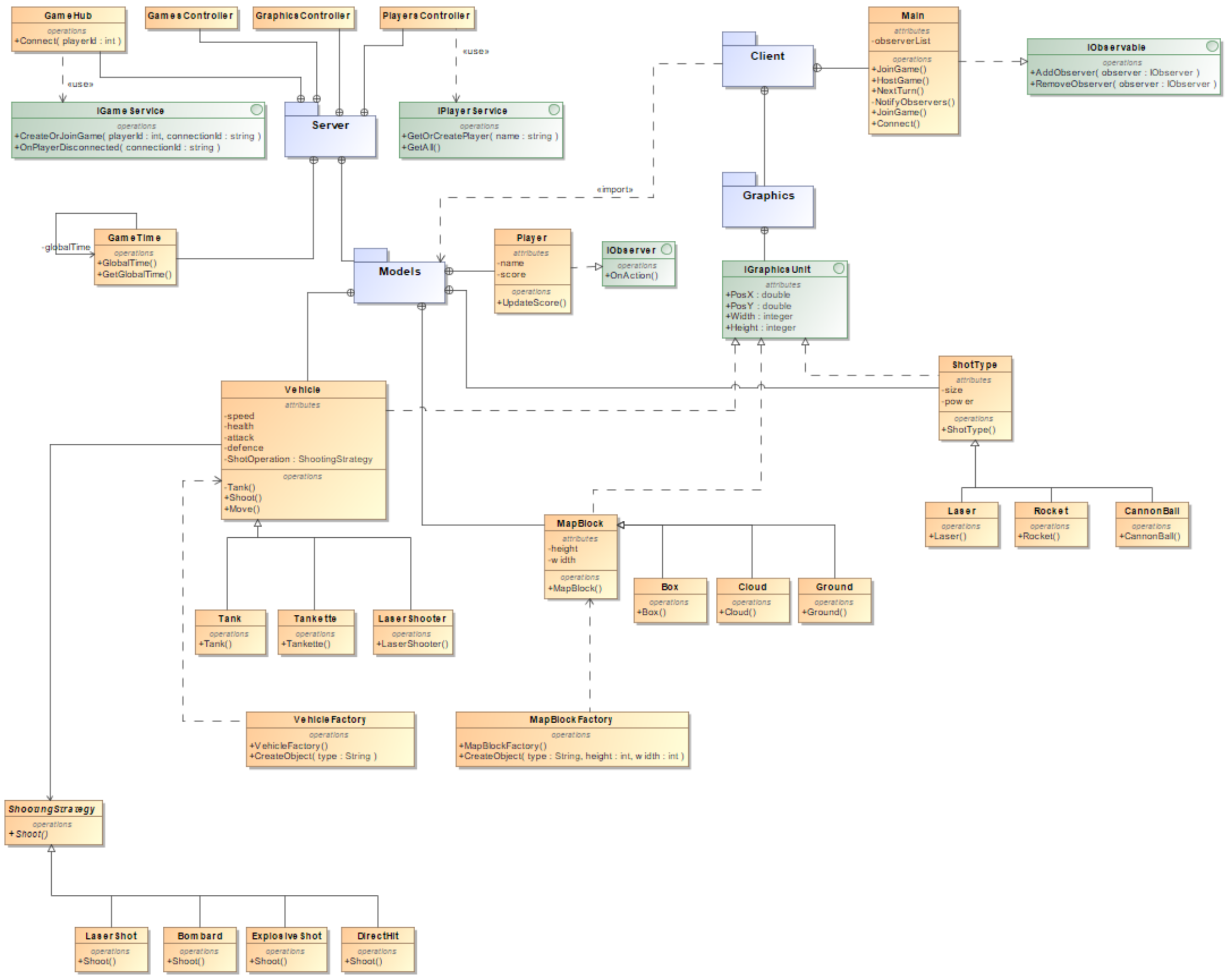
        - Jei žaidėjas pataiko į priešininko tanką, priešininkui sumažėja gyvybės taškų atitinkamai nuo šovinio tipo.
    - Jei žaidėjas nespėja baigti ėjimo per duotą laiką, prasideda kito žaidėjo ėjimas.
- Galimybė pasirinkti iš kelių skirtingų šovinių tipų.
    - Paprastas šovinys, kuris padaro žalą tik pataikius tiesiogiai į priešininko tanką.
    - Sprogstamasis šovinys, kuris nusileidęs sprogsta ir padaro žalą tam tikrame spindulyje.
    - Lazerinis šovinys, kurio trajektorija yra tiesė.
- Žaidimas pasiekia pabaigą, kai vienas žaidėjas praranda visus gyvybės taškus arba žaidimo laikas baigiasi. Laimi žaidėjas, kuriam žaidimo pabaigoje lieka daugiau gyvybės taškų.

Panaudojimo atvejų diagrama:

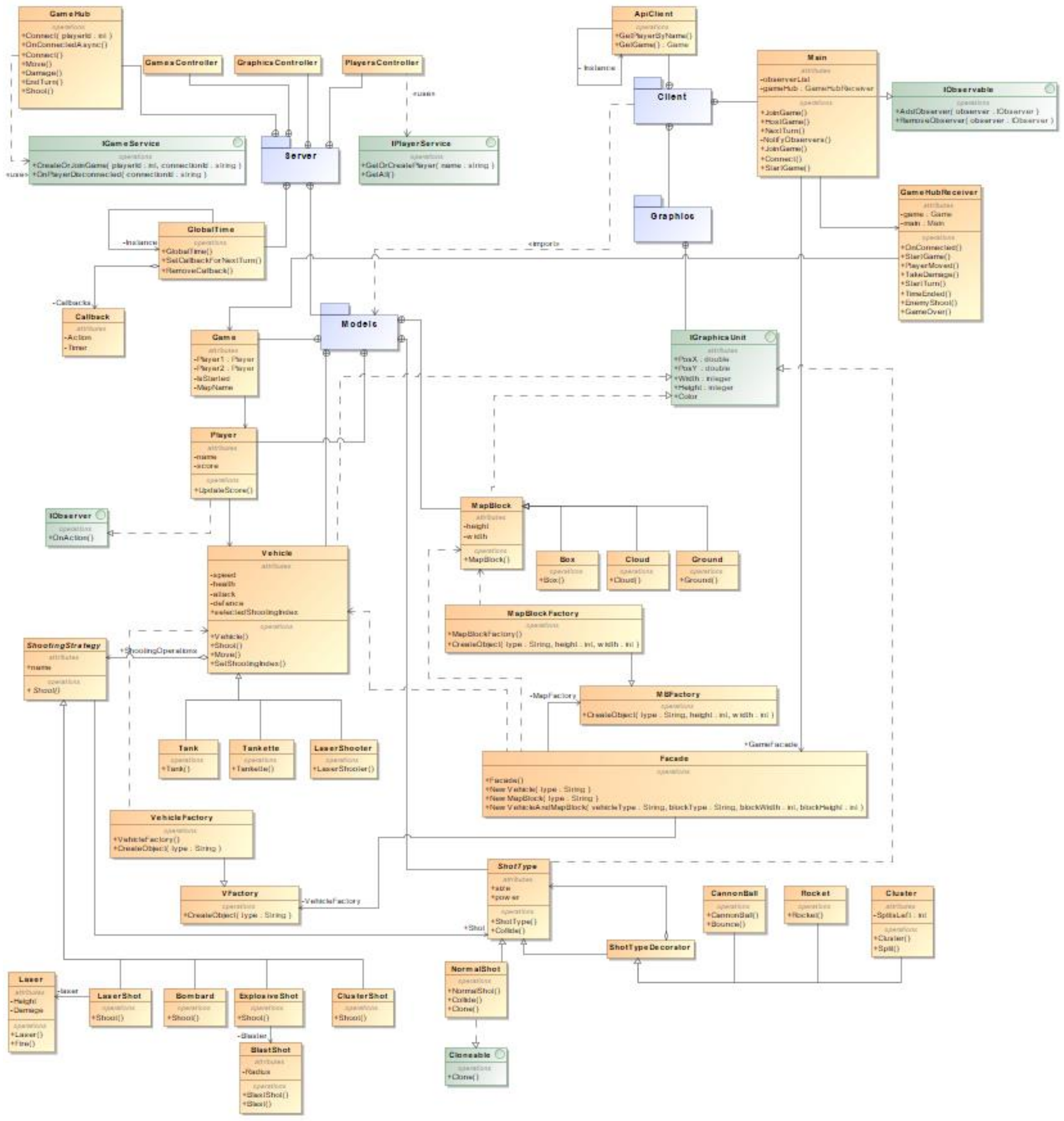


2 pav. Panaudojimo atvejų diagrama

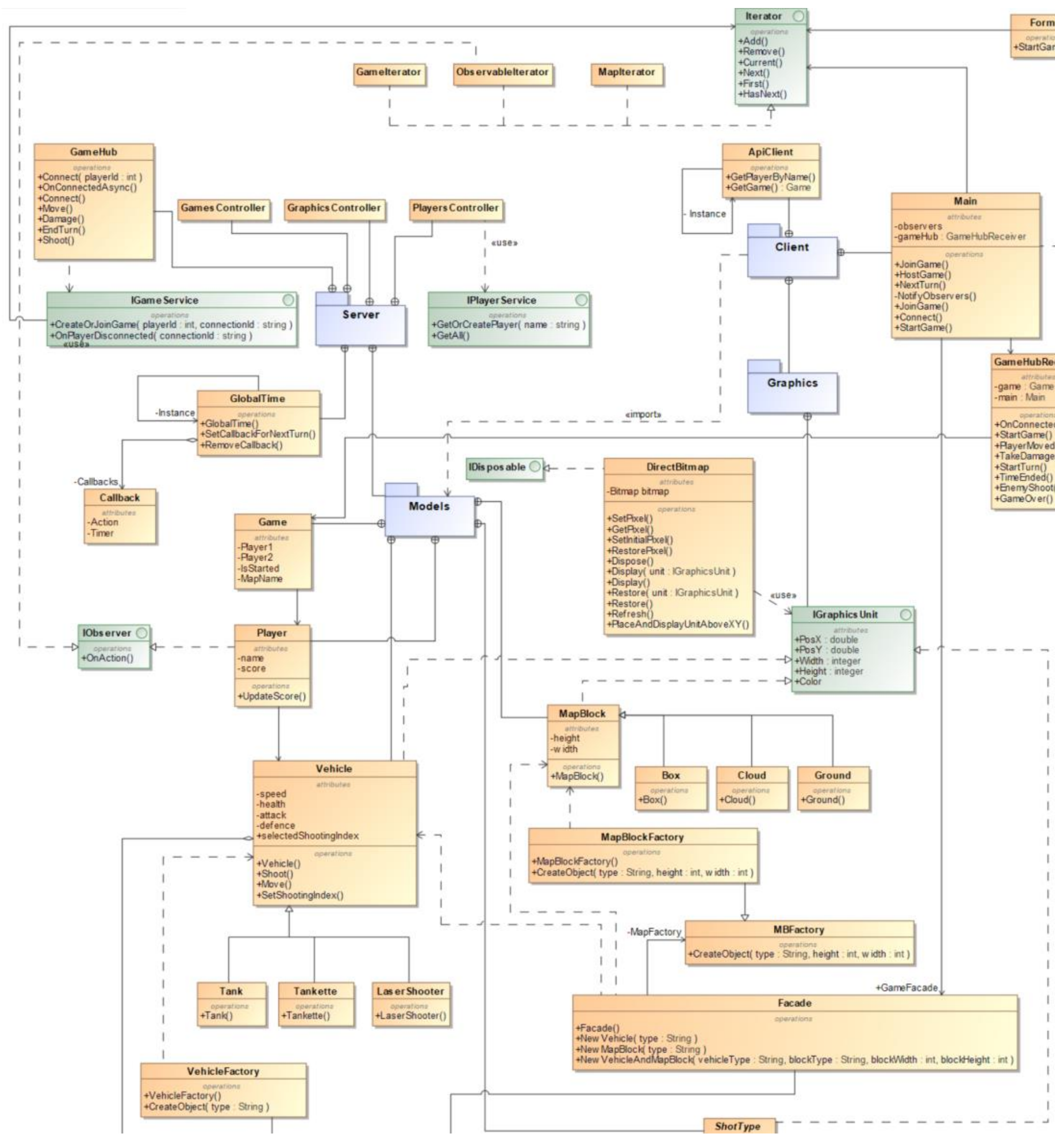
3. Klasių diagrama



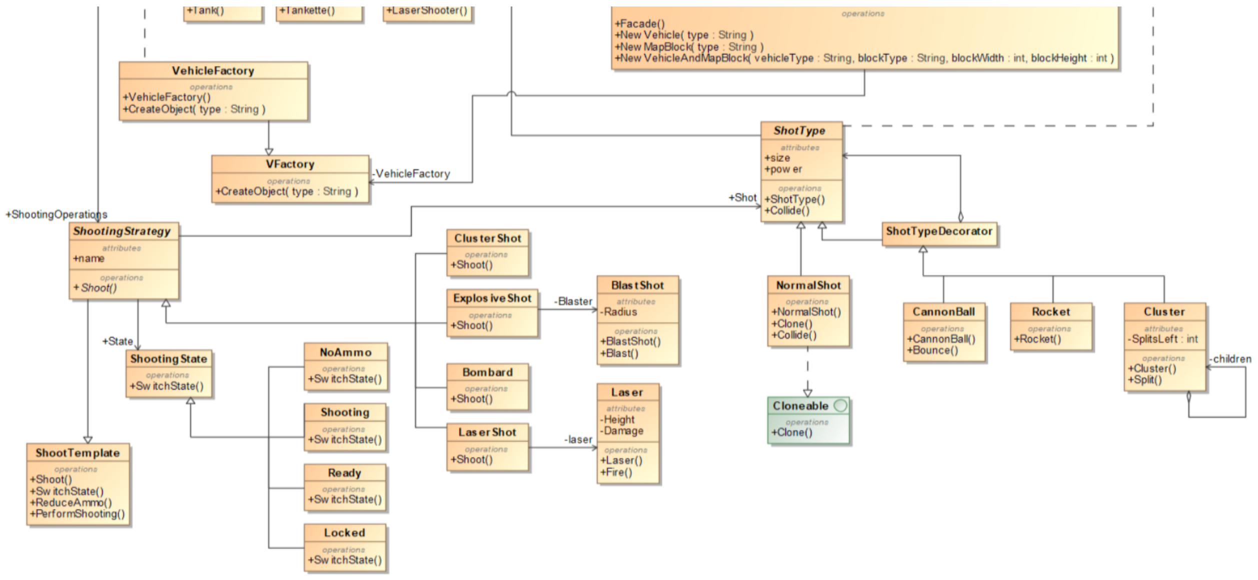
3 pav. Pradinė klasių diagrama



4 pav. Klasių diagrama po 2 žingsnio



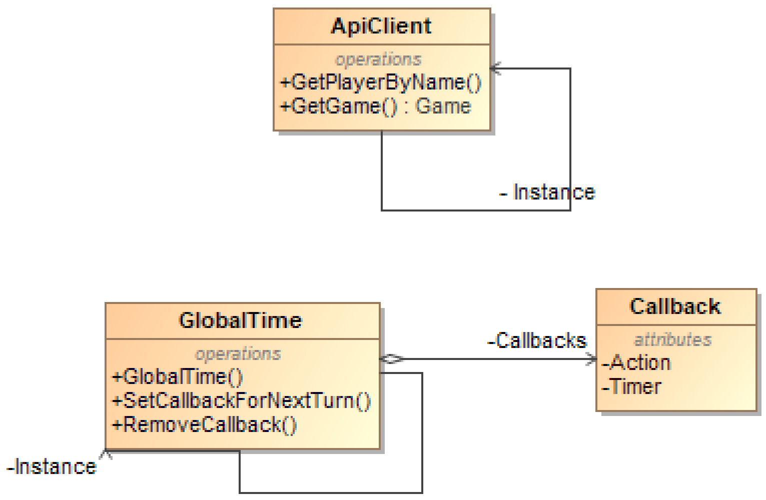
pav. 5 klasių diagrama po 3 žingsnio (1 dalis)



pav. 6 klasių diagrama po 3 žingsnio (2 dalis)

4. Realizacija

4.1. Singleton šablonas



7 pav. Singleton šablono klasių diagrama (2 realizacijos)

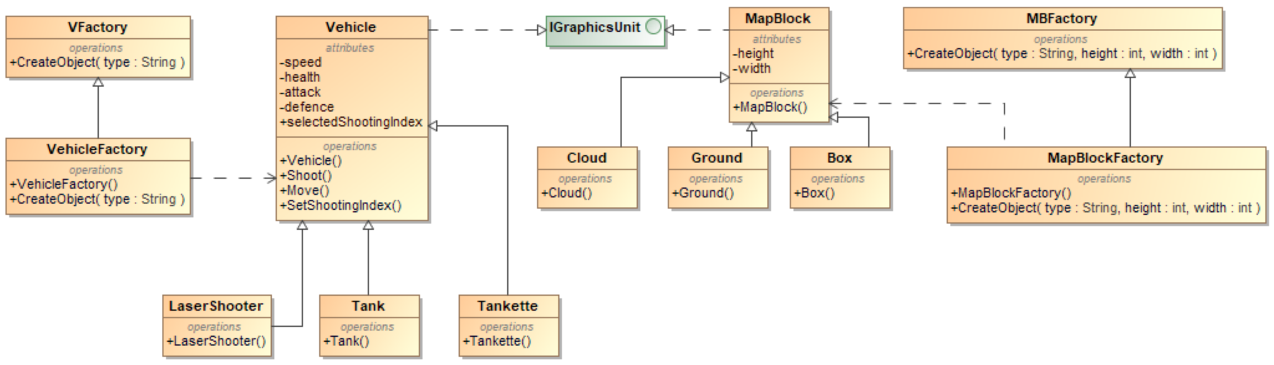
GlobalTime.cs:

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace Nemare.Backend.DataEntities
{
    public class GlobalTime
    {
        private const int TurnTimeInSeconds = 30;
        public static GlobalTime Instance { get; } = new GlobalTime();

        private  readonly  Dictionary<int,  Callback>  callbacks  =  new  Dictionary<int,
Callback>();

        public void SetCallbackForNextTurn(int id, Action callback)
        {
            callbacks.Add(id, new Callback {Action = callback});
            var timer = new Timer(state =>
            {
                if (!callbacks.ContainsKey(id))
                {
                    return;
                }
                callbacks[id].Action();
                callbacks[id].Timer.Dispose();
                callbacks.Remove(id);
            }, null, TimeSpan.FromSeconds(30), TimeSpan.FromDays(1));

            callbacks[id].Timer = timer;
        }

        public void RemoveCallback(int id)
        {
            if (!callbacks.ContainsKey(id))
            {
                return;
            }
            callbacks.Remove(id);
        }

        private class Callback
        {
            public Action Action { get; set; }
            public Timer Timer { get; set; }
        }
    }
}
```

ApiClient.cs:

```
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Nemare.Models.Entities;
using Newtonsoft.Json;

namespace NeMaRe.Logic
{
    public class ApiClient
    {
        //public const string ApiAddress = "https://localhost:5001";
        public const string ApiAddress = "https://nemare-backend.azurewebsites.net";
        private readonly HttpClient httpClient;

        private static ApiClient instance;
        public static ApiClient Instance => instance ?? (instance = new ApiClient());

        public ApiClient()
        {
            httpClient = new HttpClient
            {
                BaseAddress = new Uri(ApiAddress)
            };
            httpClient.DefaultRequestHeaders.Accept.Clear();
            httpClient.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        }

        public async Task<Player> GetPlayerByName(string name)
        {
            var result = await httpClient.GetAsync($"api/players/{name}");
            if (!result.IsSuccessStatusCode)
            {
                Console.WriteLine($"Something      went      wrong.      HTTP      response:
{result.StatusCode}");
                return null;
            }

            var content = await result.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<Player>(content);
        }
        public async Task<Game> GetGame(int gameId)
        {
            var result = await httpClient.GetAsync($"api/games/{gameId}");
            if (!result.IsSuccessStatusCode)
            {
                Console.WriteLine($"Something      went      wrong.      HTTP      response:
{result.StatusCode}");
                return null;
            }

            var content = await result.Content.ReadAsStringAsync();
            return JsonConvert.DeserializeObject<Game>(content);
        }
    }
  }
```

4.2. Factory šablonas



8 pav. Factory šablono klasių diagrama

IGraphicsUnit.cs:

```
using System;
using System.Drawing;

namespace Nemare.Models.Interfaces
{
    public interface IGraphicsUnit
    {
        double PosX { get; set; }
        double PosY { get; set; }
        int Height { get; set; }
        int Width { get; set; }
        Color Color { get; set; }
    }
}
```

Vehicle.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using Nemare.Models.Entities.ShootingStrategies;
using Nemare.Models.Interfaces;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.Vehicles
{
    public class Vehicle : IGraphicsUnit
    {
        public string Type { get; set; }
        public int Speed { get; private set; }
        public int Health { get; private set; }
        public int Defence { get; private set; }
        public int Attack { get; protected set; }
        public List<ShootingStrategy> ShootingOperations { get; set; }
        public int SelectedShootingIndex { get; protected set; }

        public double PosX { get; set; }
        public double PosY { get; set; }
        public int Height { get; set; }
        public int Width { get; set; }
        public Color Color { get; set; }

        public Vehicle()
        {

        }

        protected Vehicle(int speed, int health, int defence, int attack, int width, int
height, Color color)
        {
            Speed = speed;
            Health = health;
            Defence = defence;
            Attack = attack;

            Height = height;
            Width = width;
            Color = color;

            SelectedShootingIndex = 0;
        }

        public virtual void Shoot(double angle, int power, DirectBitmap gameMap)
        {
            double startX = this.PosX + (double)this.Width/2;
            double startY = this.PosY - this.Height;
            ShootingOperations[SelectedShootingIndex].Shoot(angle,  power,  startX,  startY,
this.Attack, gameMap);
            Console.WriteLine("Shooting with index {0} at direction {1} with power {2} and
attack {3}", SelectedShootingIndex, angle, power, Attack);
        }

        public virtual void Move(double x, double y)
        {
            Console.WriteLine("Vehicle is moving to x: {0}, y: {1}", x, y);
        }

        public void SetShootingIndex(int index)
        {
            SelectedShootingIndex = index;
        }
        public void ReceiveDamage(int dmg)
        {
            Health -= (dmg - Defence);
        }
    }
}
```

Tankette.cs:

```
using System;
using Nemare.Models.Entities.ShootingStrategies;
using System.Drawing;
using System.Collections.Generic;

namespace Nemare.Models.Entities.Vehicles
{
    public class Tankette : Vehicle
    {
        public Tankette(int speed, int health, int defence, int attack, int width, int height,
Color color) : base(speed, health, defence, attack, width, height, color)
        {
            ShootingOperations = new List<ShootingStrategy>
            {
                new Bombard(),
                new ClusterShot()
            };
        }

        public override void Move(double x, double y)
        {
            Console.WriteLine("Tankette is moving to x: {0}, y: {1}", x, y);
        }
    }
}
```

Tank.cs:

```
using Nemare.Models.Entities.ShootingStrategies;
using System.Collections.Generic;
using System.Drawing;

namespace Nemare.Models.Entities.Vehicles
{
    public class Tank : Vehicle
    {
        public Tank(int speed, int health, int defence, int attack, int height, int width,
Color color) : base(speed, health, defence, attack, height, width, color)
        {
            ShootingOperations = new List<ShootingStrategy>
            {
                new Bombard(),
                new ExplosiveShot(),
                new ClusterShot()
            };
        }
    }
}
```

LaserShooter.cs:

```
using System;
using Nemare.Models.Entities.ShootingStrategies;
using System.Drawing;
using System.Collections.Generic;

namespace Nemare.Models.Entities.Vehicles
{
    public class LaserShooter : Vehicle
    {
        public LaserShooter(int speed, int health, int defence, int attack, int width, int
height, Color color) : base(speed, health, defence, attack, width, height, color)
        {
            ShootingOperations = new List<ShootingStrategy>
            {
                new LaserShot(new Shots.Laser(6, 20, Color.Crimson)),
                new ClusterShot()
            };
        }

        public override void Move(double x, double y)
        {
            Console.WriteLine("Laser Shooter is moving to x: {0}, y: {1}", x, y);
        }
    }
}
```

VFactory.cs:

```
using Nemare.Models.Entities.Vehicles;

namespace NeMaRe.Factories
{
    abstract class VFactory
    {
        public abstract Vehicle CreateObject(string type);
    }
}
```

VehicleFactory.cs:

```
using Nemare.Models.Entities.Vehicles;
using System.Drawing;

namespace NeMaRe.Factories
{
    class VehicleFactory : VFactory
    {
        public VehicleFactory()
        {
        }

        public override Vehicle CreateObject(string type)
        {
            switch (type)
            {
                case "Tank":
                    return new Tank(40, 100, 50, 50, 6, 4, Color.Aquamarine);
                case "Tankette":
                    return new Tankette(80, 60, 40, 60, 6, 2, Color.BurlyWood);
                case "LaserShooter":
                    return new LaserShooter(60, 60, 20, 100, 4, 2, Color.Crimson);
                default:
                    return null;
            }
        }
    }
}
```

MapBlock.cs:

```
namespace Nemare.Models.Entities.Map
{
    public class MapBlock
    {
        public int Width { get; set; }
        public int Height { get; set; }

        public MapBlock(int width, int height)
        {
            Width = width;
            Height = height;
        }
    }
}
```

Box.cs:

```
namespace Nemare.Models.Entities.Map
{
    public class Box : MapBlock
    {
        public Box(int width, int height) : base(width, height)
        {
        }
    }
}
```

Ground.cs:

```
namespace Nemare.Models.Entities.Map
{
    public class Ground : MapBlock
    {
        public Ground(int width, int height) : base(width, height)
        {
        }
    }
}
```

Cloud.cs:

```
namespace Nemare.Models.Entities.Map
{
    public class Cloud : MapBlock
    {
        public Cloud(int width, int height) : base(width, height)
        {
        }
    }
}
```

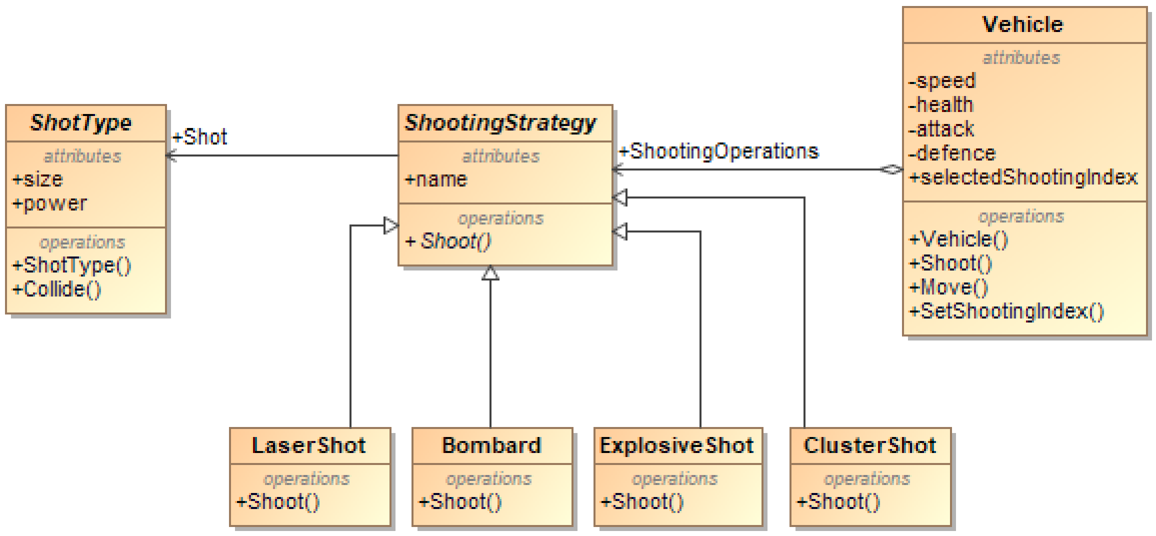
MBFactory.cs:

```
using Nemare.Models.Entities.Map;

namespace NeMaRe.Factories
{
    abstract class MBFactory
    {
        public abstract MapBlock CreateObject(string type, int width, int height);
    }
}
```

MapBlockFactory.cs:

```
using Nemare.Models.Entities.Map;

namespace NeMaRe.Factories
{
    class MapBlockFactory : MBFactory
    {
        public MapBlockFactory()
        {
        }

        public override MapBlock CreateObject(string type, int width, int height)
        {
            switch (type)
            {
                case "Box":
                    return new Box(width, height);
                case "Cloud":
                    return new Cloud(width, height);
                case "Ground":
                    return new Ground(width, height);
                default:
                    return null;
            }
        }
    }
}
```

4.3. Strategy šablonas



9 pav. Strategy šablono klasių diagrama

ShotType.cs:

```
using System.Drawing;
using Nemare.Models.Interfaces;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;

namespace Nemare.Models.Entities.Shots
{
    public abstract class ShotType : IGraphicsUnit
    {
        public int Power { get; set; }
        public double PosX { get; set; }
        public double PosY { get; set; }
        public int Height { get; set; }
        public int Width { get; set; }
        public Color Color { get; set; }

        public ShotType()
        {
        }

        public ShotType(int size, int power)
        {
            Height = Width = 2;
            Power = power;
        }

        public ShotType(int height, int width, Color color)
        {
            Height = height;
            Width = width;
            Color = color;
        }

        public abstract void Collide(DirectBitmap gameMap, double attack);
    }
}
```

ShootingStrategy.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Shots;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    public abstract class ShootingStrategy
    {
        public ShotType Shot { get; set; }
        public string Name { get; set; }

        public abstract void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap);
    }
}
```

LaserShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class LaserShot : ShootingStrategy
    {
        private Laser laser;
        public LaserShot(Laser laser)
        {
            Name = "Laser Shot";
            this.laser = laser;
        }
        public override void Shoot(double angle, int power, double startX, double startY, 
double attack, DirectBitmap gameMap)
        {
            laser.Fire(gameMap, startX, startY);
            Console.WriteLine("Shooting laser at direction " + angle);
        }
    }
}
```

Bombard.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Drawing;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using System.Windows.Forms;
using System.Diagnostics;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class Bombard : ShootingStrategy
    {
        public Bombard()
        {
            Name = "Bombard";
            Shot = new NormalShot(2, 2, Color.Black);
        }

        public override void Shoot(double angle, int power, double startX, double startY, 
double attack, DirectBitmap gameMap)
        {
            var projectile = new CannonBall((NormalShot)((NormalShot)Shot).Clone());

            var tOld = 0d;
            var xOld = -1d;
            var yOld = -1d;

            projectile.PosX = startX;
            startY -= projectile.Height;
            projectile.PosY = startY;

            var stopwatch = new Stopwatch();
            stopwatch.Start();
            double targetTime = 0 + gameMap.frameInterval;
            double currentTime = 0;
            for (double t = 0; t < 8000; t+=gameMap.frameInterval)
            {
                while (currentTime > targetTime)
                {
                    currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;
                }
                targetTime += gameMap.frameInterval;
                if (xOld > 0 && yOld > 0) // restore old position
                    gameMap.Restore(projectile);

                tOld = t;

                //calculate new projectile position
                projectile.PosX = startX - gameMap.PosXatTime(projectile, t, power, angle);
                projectile.PosY = startY - gameMap.PosYatTime(projectile, t, power, angle);
                if (projectile.PosX > 0 && projectile.PosX < gameMap.Width && projectile.PosY
> 0 && projectile.PosY < gameMap.Height)
                {
                    bool collided = gameMap.CheckUnitCollision(projectile);
                    if (collided)
                    {
                        //gameMap.Refresh();
                        projectile.Collide(gameMap, attack);
                        break;
                    }
                    else
                    {
                        gameMap.Display(projectile);
                        xOld = projectile.PosX;
                        yOld = projectile.PosY;
                        gameMap.Refresh();
                    }
                }
                else
                {
                    break;
                }
            }

            stopwatch.Stop();

            gameMap.Refresh();
        }
    }
}
```

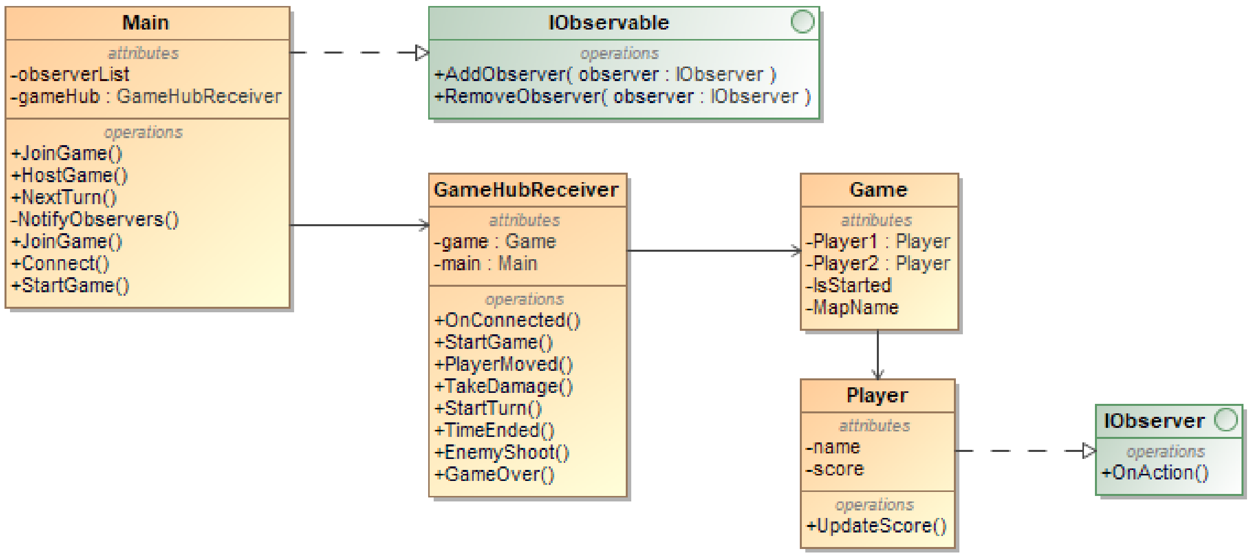
ExplosiveShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class ExplosiveShot : ShootingStrategy
    {
        private BlastShot Blaster;

        public ExplosiveShot()
        {
            Name = "Explosive Shot";
            Shot = new NormalShot(2, 2, Color.Brown);
            Blaster = new BlastShot(5);
        }
        public override void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap)
        {
            Blaster.Blast("explosive projectile", angle);
        }
    }
}
```

ClusterShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class ClusterShot : ShootingStrategy
    {
        public ClusterShot()
        {
            Name = "Direct Hit";
            Shot = new NormalShot(1, 1, Color.White);
        }

        public override void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap)
        {
            Console.WriteLine("Shooting close range projectile at direction " + angle);
        }
    }
}
```

4.4. Observer šablonas



10 pav. Observer šablono klasių diagrama

Main.cs:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.SignalR.Client;
using NeMaRe.Logic;
using Nemare.Models.Entities;
using Nemare.Models.Interfaces;

namespace NeMaRe
{
    public class Main : IObservable
    {
        private readonly List<IObserver> observerList;
        private HubConnection connection;
        private readonly GameHubReceiver gameHub;
        public Form1 Form { get; }
        public Facade GameFacade { get; }

        public Main(Form1 form)
        {
            Form = form;
            observerList = new List<IObserver>();
            gameHub = new GameHubReceiver(this);
            GameFacade = new Facade();
        }

        public void AddObserver(IObserver observer)
        {
            observerList.Add(observer);
        }

        public void RemoveObserver(IObserver observer)
        {
            observerList.Remove(observer);
        }

        public void StartGame(Player player1, Player player2, bool isPlayer1)
        {
            AddObserver(player1);
            AddObserver(player2);
            Form.StartGame(isPlayer1, player1.Vehicle.Type, player2.Vehicle.Type);
            Console.WriteLine($"Starting game with {player1.Name} and {player2.Name}");
        }

        public async void NextTurn(int posX, int posY)
        {
            observerList.ForEach(x => x.OnAction());

            await connection.InvokeCoreAsync("EndTurn", new object[]
            {
                gameHub.PlayerId,
                posX,
                posY
            });
        }

        public async void MoveVehicle(int posX, int posY)
        {
            await connection.InvokeCoreAsync("Move", new object[]
            {
                gameHub.PlayerId,
                posX,
                posY
            });
        }

        public async void Shoot(int angle, int power)
        {
            await connection.InvokeCoreAsync("Shoot", new object[] {angle, power});
        }

        public async void JoinGame(string playerName, string mapName, string tankType)
        {
            var player = await ApiClient.Instance.GetPlayerByName(playerName);

            if (connection == null)
            {
                await Connect();
            }

            await  connection.InvokeCoreAsync("Connect",  new  object[]  {player.Id,  mapName,
tankType});
        }

        private async Task Connect()
        {
            Console.WriteLine("Setting up connection");

            connection = new HubConnectionBuilder()
                .WithUrl($"{ApiClient.ApiAddress}/hubs/game")
                .Build();

            connection.On("OnConnected", (Action<int, int>)gameHub.OnConnected);
            connection.On("StartGame", gameHub.StartGame);
            connection.On("PlayerMoved", (Action<int, int, int>)gameHub.PlayerMoved);
            connection.On("TakeDamage", (Action<int, int>)gameHub.TakeDamage);
            connection.On("StartTurn", gameHub.StartTurn);
            connection.On("EnemyShoot", (Action<int, int>)gameHub.EnemyShoot);
            connection.On("TimeEnded", gameHub.TimeEnded);
            connection.On("GameOver", (Action<bool>)gameHub.GameOver);

            await connection.StartAsync();
        }

        public async void ExitGame()
        {
            await connection.StopAsync();
            await connection.DisposeAsync();
        }
    }
}
```

IObservable.cs:

```
namespace Nemare.Models.Interfaces
{
    public interface IObservable
    {
        void AddObserver(IObserver observer);

        void RemoveObserver(IObserver observer);
    }
}
```

GameHubReceiver.cs:

```
using System;
using Nemare.Models.Entities;

namespace NeMaRe.Logic
{
    public class GameHubReceiver
    {
        public int GameId { get; set; }
        public int? PlayerId { get; set; }
        private Main main;
        private Game game;

        public GameHubReceiver(Main main)
        {
            this.main = main;
        }

        public void OnConnected(int playerId, int gameId)
        {
            Console.WriteLine($"Player {playerId} joined the game {gameId}");

            if (!PlayerId.HasValue)
            {
                PlayerId = playerId;
                GameId = gameId;
            }
        }

        public async void StartGame()
        {
            game = await ApiClient.Instance.GetGame(GameId);
            main.StartGame(game.Player1, game.Player2, game.Player1.Id == PlayerId.Value);
        }

        public void PlayerMoved(int playerId, int posX, int posY)
        {
            if (playerId != PlayerId)
            {
                main.Form.UpdateTankPosition(true, posX, posY);
            }
        }

        public void TakeDamage(int playerId, int damage)
        {

        }

        public void StartTurn()
        {
            main.Form.ToggleControls(true);
        }

        public void TimeEnded()
        {
            main.Form.ToggleControls(false);
        }

        public void EnemyShoot(int angle, int power)
        {
            main.Form.Shoot(true, angle, power);
        }

        public void GameOver(bool isWinner)
        {
            main.Form.GameOver(isWinner);
        }
    }
}
```

Game.cs:

```
namespace Nemare.Models.Entities
{
    public class Game
    {
        public bool Id { get; set; }
        public bool IsStarted { get; set; }
        public Player Player1 { get; set; }
        public Player Player2 { get; set; }
        public string MapName { get; set; }
    }
}
```

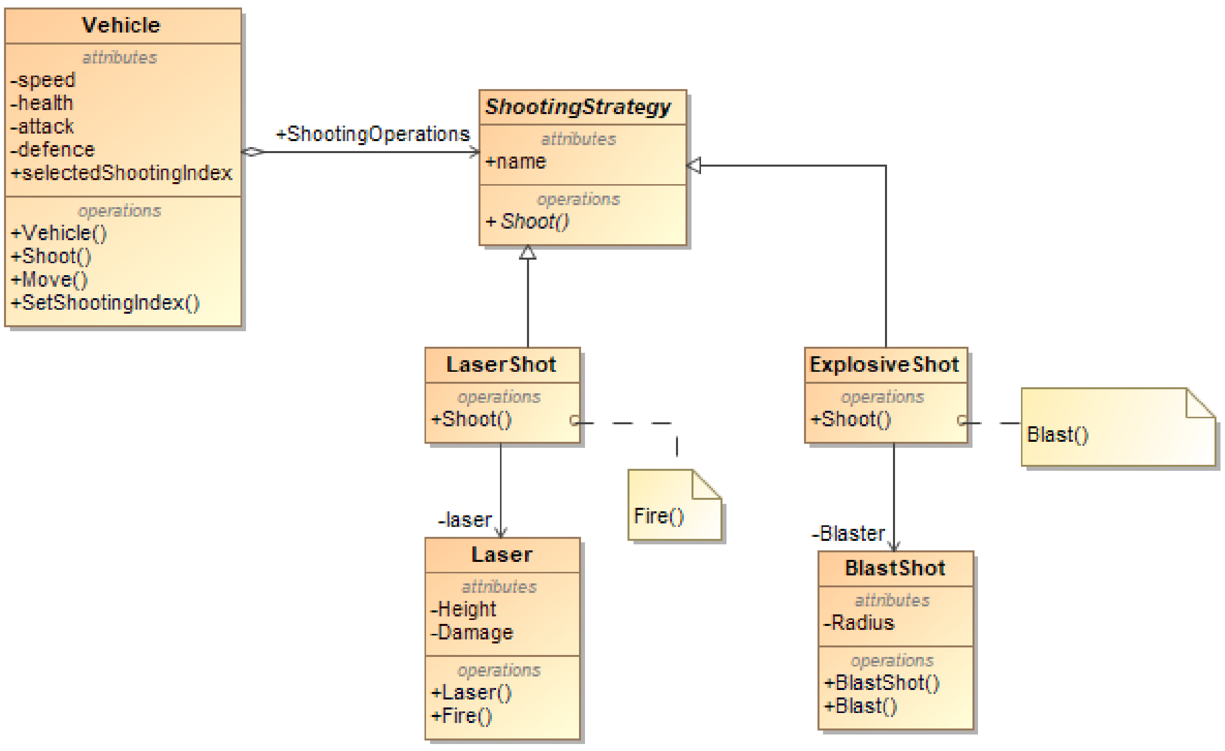
Player.cs:

```
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Interfaces;

namespace Nemare.Models.Entities
{
    public class Player : IObserver
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Score { get; set; }
        public Vehicle Vehicle { get; set; }

        public void UpdateScore()
        {

        }

        public void OnAction()
        {
            UpdateScore();
        }
    }
}
```

IObserver.cs:

```
namespace Nemare.Models.Interfaces
{
    public interface IObserver
    {
        void OnAction();
    }
}
```

4.5. Adapter šablonas



11 pav. Adapter šablono klasių diagrama (2 realizacijos)

ShootingStrategy.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Shots;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    public abstract class ShootingStrategy
    {
        public ShotType Shot { get; set; }
        public string Name { get; set; }

        public abstract void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap);
    }
}
```

LaserShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class LaserShot : ShootingStrategy
    {
        private Laser laser;
        public LaserShot(Laser laser)
        {
            Name = "Laser Shot";
            this.laser = laser;
        }

        public override void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap)
        {
            laser.Fire(gameMap, startX, startY);
            Console.WriteLine("Shooting laser at direction " + angle);
        }
    }
}
```

Laser.cs:

```
using System;
using System.Diagnostics;
using System.Drawing;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Interfaces;

namespace Nemare.Models.Entities.Shots
{
    class Laser
    {
        int Height;
        int Damage;
        Color Color;
        public Laser() { }

        public Laser(int height, int damage, Color color)
        {
            Height = height;
            Damage = damage;
            Color = color;
        }

        public void Fire(DirectBitmap gameMap, double startX, double startY)
        {
            int x;
            var stopwatch = new Stopwatch();
            NormalShot unit = new NormalShot(1,20);
            unit.Color = Color;
            unit.Height = Height;
            unit.PosY = startY - unit.Height / 2;

            stopwatch.Start();
            double targetTime = 0 + gameMap.frameInterval;
            double currentTime = 0;
            for (x = (int)startX; x < gameMap.Width-1; x++)
            {
                unit.PosX = x;
                bool collided = gameMap.CheckUnitCollision(unit);
                if (collided)
                {
                    unit.Collide(gameMap, Damage);
                    x++;
                    break;
                }
                else
                {
                    gameMap.Display(unit);
                }
                currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;
                if (currentTime > targetTime)
                {
                    gameMap.Refresh();

                    targetTime += gameMap.frameInterval;
                }
            }
            gameMap.Refresh();
            targetTime += 38 * gameMap.frameInterval; // wait n more frames until restoring
            while (currentTime < targetTime) // wait for remaining frames
                currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;

            stopwatch.Stop();

            for (x--; x >= (int)startX; x--)
            {
                unit.PosX = x;
                gameMap.Restore(unit);
            }
            gameMap.Refresh();
        }
    }
}
```

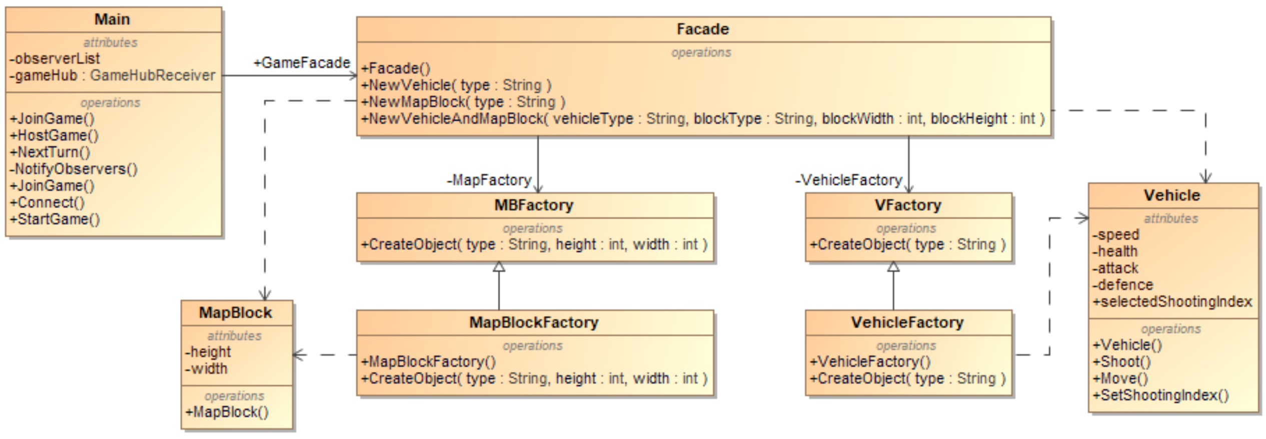
ExplosiveShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class ExplosiveShot : ShootingStrategy
    {
        private BlastShot Blaster;

        public ExplosiveShot()
        {
            Name = "Explosive Shot";
            Shot = new NormalShot(2, 2, Color.Brown);
            Blaster = new BlastShot(5);
        }

        public override void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap)
        {
            Blaster.Blast("explosive projectile", angle);
        }
    }
}
```

BlastShot.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Nemare.Models.Entities.Shots
{
    class BlastShot
    {
        public int Radius;

        public BlastShot(int radius)
        {
            Radius = radius;
        }

        public void Blast(string type, double angle)
        {
            Console.WriteLine("Shooting {0} at angle {1}. It blasts in {2} block radius.",
type, angle, Radius);
        }
    }
}
```

4.6. Façade šablonas

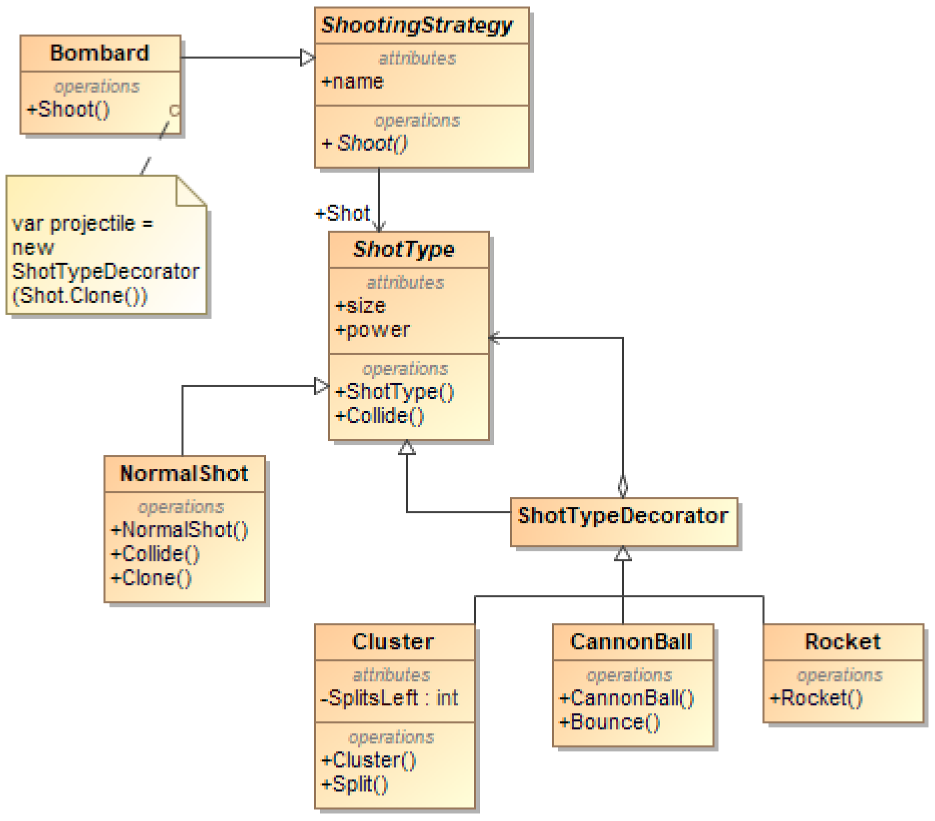


12 pav. Façade šablono klasių diagrama

Façade.cs:

```
using System;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using NeMaRe.Factories;

namespace NeMaRe
{
    public class Facade
    {
        private MBFactory MapFactory;
        private VFactory VehicleFactory;

        public Facade()
        {
            MapFactory = new MapBlockFactory();
            VehicleFactory = new VehicleFactory();
        }

        public Vehicle NewVehicle(string type)
        {
            Console.WriteLine("Creating vehicle of type {0}", type);
            return VehicleFactory.CreateObject(type);
        }

        public MapBlock NewMapBlock(string type, int width, int height)
        {
            Console.WriteLine("Creating map block of type {0}, width {1}, height {2}", type,
width, height);
            return MapFactory.CreateObject(type, width, height);
        }

        public   (Vehicle,   MapBlock)   NewVehicleAndMapBlock(string   vehicleType,   string
blockType, int blockWidth, int blockHeight)
        {
            Vehicle TempVehicle = NewVehicle(vehicleType);
            MapBlock TempMapBlock = NewMapBlock(blockType, blockWidth, blockHeight);

            return (TempVehicle, TempMapBlock);
        }
    }
}
```

4.7. Decorator šablonas



13 pav. Decorator šablono klasių diagrama

ShotType.cs:

```
using System.Drawing;
using Nemare.Models.Interfaces;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;

namespace Nemare.Models.Entities.Shots
{
    public abstract class ShotType : IGraphicsUnit
    {
        public int Power { get; set; }
        public double PosX { get; set; }
        public double PosY { get; set; }
        public int Height { get; set; }
        public int Width { get; set; }
        public Color Color { get; set; }

        public ShotType()
        {
        }
        public ShotType(int size, int power)
        {
            Height = Width = 2;
            Power = power;
        }

        public ShotType(int height, int width, Color color)
        {
            Height = height;
            Width = width;
            Color = color;
        }

        public abstract void Collide(DirectBitmap gameMap, double attack);
    }
}
```

NormalShot.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Interfaces;

namespace Nemare.Models.Entities.Shots
{
    class NormalShot : ShotType, Cloneable
    {
        public NormalShot() : base() { }

        public NormalShot(int size, int power) : base(size, power) { }

        public NormalShot(int height, int width, Color color) : base(height, width, color)
{ }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            Console.WriteLine("<Collision>");
            gameMap.inflictDamage(this, attack);
        }

        public object Clone()
        {
            NormalShot clone = (NormalShot)this.MemberwiseClone();
            clone.Color = Color.FromArgb(this.Color.ToArgb());
            return clone;
        }
    }
}
```

ShotTypeDecorator.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;

namespace Nemare.Models.Entities.Shots
{
    class ShotTypeDecorator : ShotType
    {
        public ShotType shot { get; private set; }
        public ShotTypeDecorator(ShotType shot)
        {
            this.shot = shot;
            Power = shot.Power;
            PosX = shot.PosX;
            PosY = shot.PosY;
            Height = shot.Height;
            Width = shot.Width;
            Color = shot.Color;
        }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            shot.Collide(gameMap, attack);
        }
    }
}
```

Cluster.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Interfaces;

namespace Nemare.Models.Entities.Shots
{
    class Cluster : ShotTypeDecorator
    {

        public Cluster(ShotType shot) : base(shot) { }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            base.Collide(gameMap, attack);
            Console.WriteLine("<" + this.GetType().ToString() + ">");
        }

        private void Split()
        {
            Console.WriteLine("<split>");
        }
    }
}
```

CannonBall.cs:

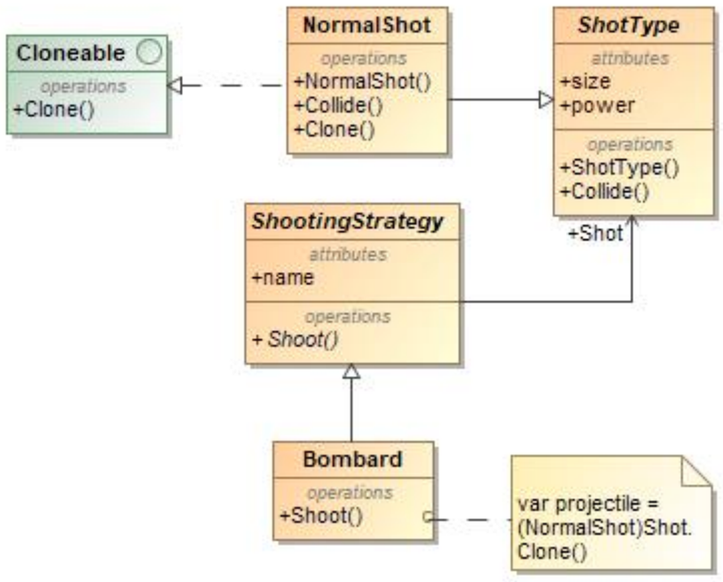
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;

namespace Nemare.Models.Entities.Shots
{
    class CannonBall : ShotTypeDecorator
    {
        public CannonBall(ShotType shot) : base(shot) { }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            base.Collide(gameMap, attack);
            Console.WriteLine("<" + this.GetType().ToString() + ">");
            Bounce(gameMap, attack);
        }

        public void Bounce(DirectBitmap gameMap, double attack)
        {
            Random rand = new Random();
            int power = rand.Next(10, 25);
            int angle = rand.Next(45, 135);

            var projectile = shot;

            var tOld = 0d;
            var xOld = -1d;
            var yOld = -1d;

            double startY;

            projectile.PosX = PosX;
            //projectile.PosY = PosY - 5;
            projectile.PosY = PosY;
            gameMap.PlaceAndDisplayUnitAboveXY(projectile,                  projectile.PosX,
projectile.PosY);
            startY = projectile.PosY;

            var stopwatch = new Stopwatch();
            stopwatch.Start();
            double targetTime = 0 + gameMap.frameInterval;
            double currentTime = 0;
            for (double t = 0; t < 8000; t += gameMap.frameInterval)
            {
                while (currentTime > targetTime)
                {
                    currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;
                }
                targetTime += gameMap.frameInterval;
                if (xOld > 0 && yOld > 0)
```

```
                    gameMap.Restore(projectile);

                tOld = t;

                projectile.PosX = PosX - gameMap.PosXatTime(projectile, t, power, angle);
                projectile.PosY = startY - gameMap.PosYatTime(projectile, t, power, angle);
                if (projectile.PosX > 0 && projectile.PosX < gameMap.Width && projectile.PosY
> 0 && projectile.PosY < gameMap.Height)
                {
                    bool collided = gameMap.CheckUnitCollision(projectile);
                    if (collided)
                    {
                        projectile.Collide(gameMap, attack);
                        //gameMap.Restore(projectile);
                        break;
                    }
                    else
                    {
                        gameMap.Display(projectile);
                        xOld = projectile.PosX;
                        yOld = projectile.PosY;
                        gameMap.Refresh();
                    }
                }
                else
                {
                    break;
                }
            }

            stopwatch.Stop();

            gameMap.Refresh();
        }
    }
}
```

Rocket.cs:

```
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Nemare.Models.Entities.Shots
{
    class Rocket : ShotTypeDecorator
    {

        public Rocket(ShotType shot) : base(shot) { }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            base.Collide(gameMap, attack);
            Console.WriteLine("<" + this.GetType().ToString() + ">");
        }
        private void Split()
        {
            Console.WriteLine("<split>");
        }
    }
}
```

4.8. Prototype šablonas



14 pav. Prototype šablono klasių diagrama

Cloneable.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Nemare.Models.Interfaces
{
    public interface Cloneable
    {
        object Clone();
    }
}
```

NormalShot.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Interfaces;

namespace Nemare.Models.Entities.Shots
{
    class NormalShot : ShotType, Cloneable
    {
        public NormalShot() : base() { }

        public NormalShot(int size, int power) : base(size, power) { }

        public NormalShot(int height, int width, Color color) : base(height, width, color)
{ }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            Console.WriteLine("<Collision>");
            gameMap.inflictDamage(this, attack);
        }

        public object Clone()
        {
            NormalShot clone = (NormalShot)this.MemberwiseClone();
            clone.Color = Color.FromArgb(this.Color.ToArgb());
            return clone;
        }
    }
}
```

ShotType.cs:

```
using System.Drawing;
using Nemare.Models.Interfaces;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;

namespace Nemare.Models.Entities.Shots
{
    public abstract class ShotType : IGraphicsUnit
    {
        public int Power { get; set; }
        public double PosX { get; set; }
        public double PosY { get; set; }
        public int Height { get; set; }
        public int Width { get; set; }
        public Color Color { get; set; }

        public ShotType()
        {
        }

        public ShotType(int size, int power)
        {
            Height = Width = 2;
            Power = power;
        }

        public ShotType(int height, int width, Color color)
        {
            Height = height;
            Width = width;
            Color = color;
        }

        public abstract void Collide(DirectBitmap gameMap, double attack);
    }
}
```

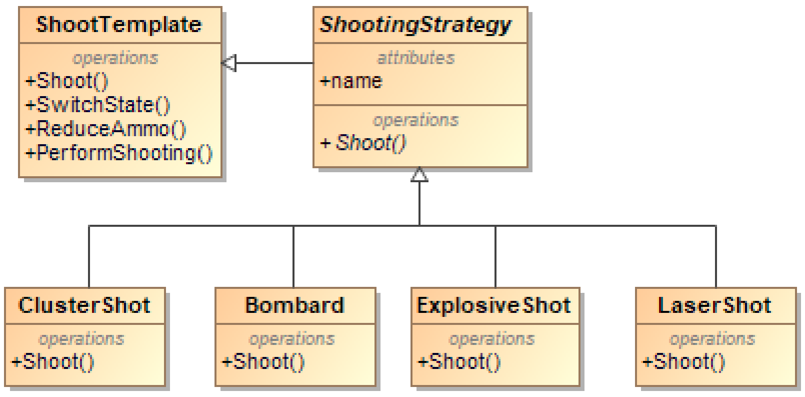
ShootingStrategy.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Shots;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.ShootingStrategies
{
    public abstract class ShootingStrategy
    {
        public ShotType Shot { get; set; }
        public string Name { get; set; }

        public abstract void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap);
    }
}
```

Bombard.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Drawing;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using System.Windows.Forms;
using System.Diagnostics;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class Bombard : ShootingStrategy
    {
        public Bombard()
        {
            Name = "Bombard";
            Shot = new NormalShot(2, 2, Color.Black);
        }

        public override void Shoot(double angle, int power, double startX, double startY,
double attack, DirectBitmap gameMap)
        {
            var projectile = new CannonBall((NormalShot)((NormalShot)Shot).Clone());
            var tOld = 0d;
            var xOld = -1d;
            var yOld = -1d;

            projectile.PosX = startX;
            startY -= projectile.Height;
            projectile.PosY = startY;

            var stopwatch = new Stopwatch();
            stopwatch.Start();
            double targetTime = 0 + gameMap.frameInterval;
            double currentTime = 0;
            for (double t = 0; t < 8000; t+=gameMap.frameInterval)
            {
                while (currentTime > targetTime)
                {
                    currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;
                }
                targetTime += gameMap.frameInterval;
                if (xOld > 0 && yOld > 0) // restore old position
                    gameMap.Restore(projectile);

                tOld = t;

                //calculate new projectile position
                projectile.PosX = startX - gameMap.PosXatTime(projectile, t, power, angle);
                projectile.PosY = startY - gameMap.PosYatTime(projectile, t, power, angle);
                if (projectile.PosX > 0 && projectile.PosX < gameMap.Width && projectile.PosY
> 0 && projectile.PosY < gameMap.Height)
                {
                    bool collided = gameMap.CheckUnitCollision(projectile);
                    if (collided)
                    {
                        //gameMap.Refresh();
                        projectile.Collide(gameMap, attack);
                        break;
                    }
                    else
                    {
                        gameMap.Display(projectile);
                        xOld = projectile.PosX;
                        yOld = projectile.PosY;
                        gameMap.Refresh();
                    }
                }
                else
                {
                    break;
                }
            }

            stopwatch.Stop();

            gameMap.Refresh();
        }
    }
}
```

4.9. Template šablonas



15 pav. Template šablono klasių diagrama

ShootTemplate.cs:

```
using Nemare.Models.Entities.Map;

namespace Nemare.Models.Entities.Templates
{
    public abstract class ShootTemplate
    {
        public abstract void SwitchState();
        public abstract void ReduceAmmo();
        public abstract void PerformShooting(double angle, int power, double startX, double
startY, double attack, DirectBitmap gameMap);

        public void Shoot(double angle, int power, double startX, double startY, double
attack, DirectBitmap gameMap)
        {
            SwitchState();
            PerformShooting(angle, power, startX, startY, attack, gameMap);
            ReduceAmmo();
            SwitchState();
        }
    }
}
```

ShootingStrategy.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Shots;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.ShootingStrategies.States;
using Nemare.Models.Entities.Templates;

namespace Nemare.Models.Entities.ShootingStrategies
{
    public abstract class ShootingStrategy : ShootTemplate
    {
        public ShootingState State { get; set; }
        public ShotType Shot { get; set; }
        public string Name { get; set; }
        public int Ammo { get; set; }

        public override void SwitchState()
        {
            State.SwitchState(this);
        }

        public override void ReduceAmmo()
        {
            Ammo--;
        }

        public void SetLockedState()
        {
            State = new Locked();
        }
    }
}
```

ClusterShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.ShootingStrategies.States;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class ClusterShot : ShootingStrategy
    {
        public ClusterShot()
        {
            Name = "Direct Hit";
            Shot = new NormalShot(1, 1, Color.White);
            Ammo = 5;
            State = new Locked();
        }

        public override void PerformShooting(double angle, int power, double startX, double
startY, double attack, DirectBitmap gameMap)
        {
            Console.WriteLine("Shooting close range projectile at direction " + angle);
        }
    }
}
```

Bombard.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Drawing;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using System.Windows.Forms;
using System.Diagnostics;
using Nemare.Models.Entities.ShootingStrategies.States;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class Bombard : ShootingStrategy
    {
        public Bombard()
        {
            Name = "Bombard";
            Shot = new NormalShot(2, 2, Color.Black);
            Ammo = 10;
            State = new Locked();
        }

        public override void PerformShooting(double angle, int power, double startX, double
startY, double attack, DirectBitmap gameMap)
        {
            var projectile = new CannonBall((NormalShot)((NormalShot)Shot).Clone());

            var tOld = 0d;
            var xOld = -1d;
            var yOld = -1d;

            projectile.PosX = startX;
            startY -= projectile.Height;
            projectile.PosY = startY;

            var stopwatch = new Stopwatch();
            stopwatch.Start();
            var targetTime = 0 + gameMap.frameInterval;
            double currentTime = 0;
            for (double t = 0; t < 8000; t+=gameMap.frameInterval)
            {
                while (currentTime > targetTime)
                {
                    currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;
                }
                targetTime += gameMap.frameInterval;
                if (xOld > 0 && yOld > 0) // restore old position
                    gameMap.Restore(projectile);

                tOld = t;

                //calculate new projectile position
                projectile.PosX = startX - gameMap.PosXatTime(projectile, t, power, angle);
                projectile.PosY = startY - gameMap.PosYatTime(projectile, t, power, angle);
                if (projectile.PosX > 0 && projectile.PosX < gameMap.Width &&
projectile.PosY > 0 && projectile.PosY < gameMap.Height)
                {
                    var collided = gameMap.CheckUnitCollision(projectile);
                    if (collided)
                    {
                        //gameMap.Refresh();
                        projectile.Collide(gameMap, attack);
                        break;
                    }
                    else
                    {
                        gameMap.Display(projectile);
                        xOld = projectile.PosX;
                        yOld = projectile.PosY;
                        gameMap.Refresh();
                    }
                }
                else
                {
                    break;
                }
            }

            stopwatch.Stop();

            gameMap.Refresh();
        }
    }
}
```

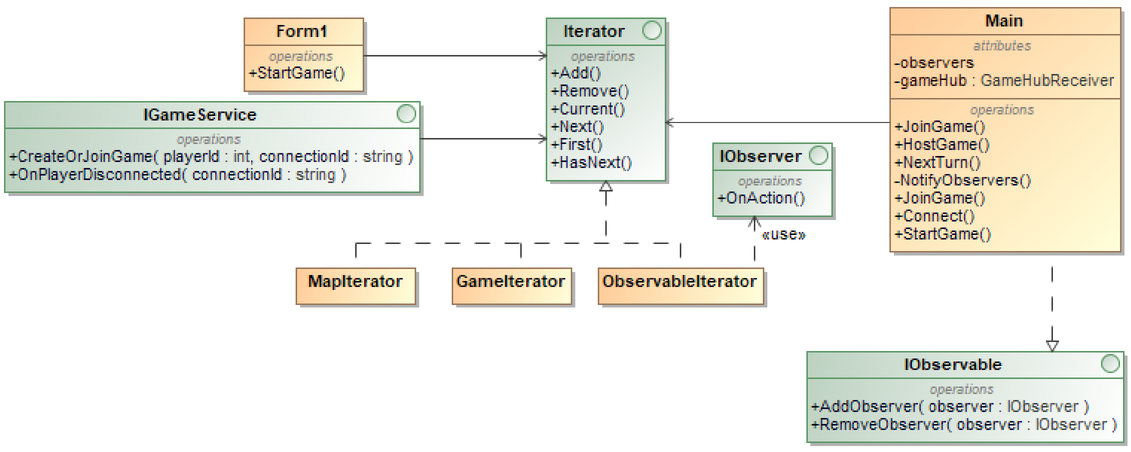
ExplosiveShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.ShootingStrategies.States;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class ExplosiveShot : ShootingStrategy
    {
        private BlastShot Blaster;

        public ExplosiveShot()
        {
            Name = "Explosive Shot";
            Shot = new NormalShot(2, 2, Color.Brown);
            Blaster = new BlastShot(5);
            Ammo = 5;
            State = new Locked();
        }

        public override void PerformShooting(double angle, int power, double startX, double
startY, double attack, DirectBitmap gameMap)
        {
            Blaster.Blast("explosive projectile", angle);
        }
    }
}
```

LaserShot.cs:

```
using Nemare.Models.Entities.Shots;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.ShootingStrategies.States;

namespace Nemare.Models.Entities.ShootingStrategies
{
    class LaserShot : ShootingStrategy
    {
        private Laser laser;
        public LaserShot(Laser laser)
        {
            Name = "Laser Shot";
            this.laser = laser;
            Ammo = 5;
            State = new Locked();
        }

        public override void PerformShooting(double angle, int power, double startX, double
startY, double attack, DirectBitmap gameMap)
        {
            laser.Fire(gameMap, startX, startY);
            Console.WriteLine("Shooting laser at direction " + angle);
        }
    }
}
```

4.10. Iterator šablonas



16 pav. Iterator šablono klasių diagrama

Iterator.cs

```
public interface Iterator<T>
{
    T Next();
    T Current();
    bool HasNext();
    T First();
    void Remove(T Item);
    void Add(T item);
}
```

MapIterator.cs

```
public class MapIterator : Iterator<MapIterator.Pixel>
{
    private readonly Bitmap map;
    private int x;
    private int y;
    private bool isFinished;

    public MapIterator(Bitmap map)
    {
        this.map = map;
        x = 0;
        y = 0;
        isFinished = false;
    }

    public Pixel Next()
    {
        var val = Current();
        x += 1;
        if (x >= map.Width && y < map.Height)
        {
            x = 0;
            y += 1;

            if (y >= map.Height)
            {
                isFinished = true;
            }
        }

        return val;
    }

    public Pixel First()
    {
        x = 0;
        y = 0;
        return Next();
    }

    public void Remove(Pixel Item)
    {
        throw new System.NotImplementedException();
    }

    public void Add(Pixel item)
    {
        throw new System.NotImplementedException();
    }

    public bool HasNext()
    {
        return !isFinished;
    }

    public Pixel Current()
    {
        return new Pixel
        {
            X = x,
            Y = y,
            Color = map.GetPixel(x, y)
        };
    }

    public class Pixel
    {
        public int X { get; set; }
        public int Y { get; set; }
        public Color Color { get; set; }
    }
}
```

GameIterator.cs

```
public class GameIterator : Iterator<Game>
{
    private readonly List<Game> games;
    private int i;

    public GameIterator(List<Game> games)
    {
        this.games = games;
    }

    public Game Next()
    {
        var val = games[i];
        i += 1;
        return val;
    }

    public Game Current()
    {
        return games[i];
    }

    public bool HasNext()
    {
        return i < games.Count;
    }

    public Game First()
    {
        i = 0;
        return Current();
    }
    public void Remove(Game Item)
    {
        games.Remove(Item);
    }

    public void Add(Game item)
    {
        games.Add(item);
    }
}
```

ObserverIterator.cs

```
public class ObserverIterator : Iterator<IObserver>
{
    private readonly List<IObserver> observers;
    private int i;

    public ObserverIterator()
    {
        observers = new List<IObserver>();
        i = 0;
    }

    public IObserver Next()
    {
        var val = observers[i];
        i += 1;
        return val;
    }

    public IObserver Current()
    {
        return observers[i];
    }

    public bool HasNext()
    {
        return i < observers.Count;
    }

    public IObserver First()
    {
        i = 0;
        return Next();
    }

    public void Remove(IObserver Item)
    {
        observers.Remove(Item);
    }

    public void Add(IObserver item)
    {
        observers.Add(item);
    }
}
```

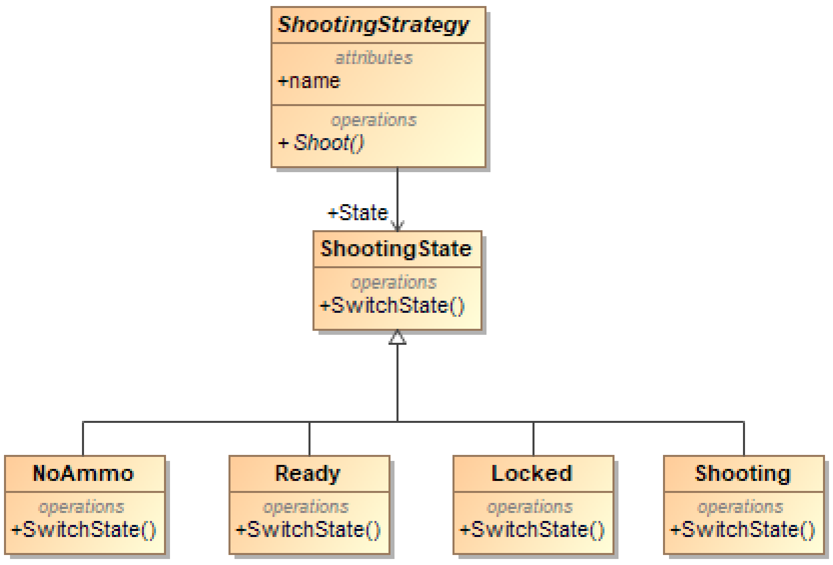
Form1.cs/StartGame(...)

```
public void StartGame(bool isPlayer1, string vehicleType1, string vehicleType2)
{
    var origmap = new Bitmap($"maps/{mapName}.png"); //take image
    var resizeMult = 3;
    var map = ResizeImage(origmap, origmap.Width * resizeMult, origmap.Height * resizeMult);
// resize it
    gameMap = new DirectBitmap(map.Width, map.Height, pictureBox1); // create bitmap with
resized size
    var mapIterator = new MapIterator(map);
    for (var pixel = mapIterator.First(); mapIterator.HasNext(); pixel = mapIterator.Next())
    {
        gameMap.SetInitialPixel(pixel.X, pixel.Y, pixel.Color); // push pixels into
DirectBitmap
    }

    Invoke((Action)(() =>
    {
        pictureBox1.Image = gameMap.Bitmap; // gameMap.Display image
        textBoxAngle.Text = "0";
        textBoxVelocity.Text = "0";
        textBoxSpeedmult.Text = "1";
        Console.WriteLine("Map loaded");
    }));

    if (!isPlayer1)
    {
        ToggleControls(false);
    }

    var placeOffset = 50;

    vehicle1 = main.GameFacade.NewVehicle(isPlayer1 ? vehicleType1 : vehicleType2);
    vehicle2 = main.GameFacade.NewVehicle(!isPlayer1 ? vehicleType1 : vehicleType2);

    vehicle1.PosX = isPlayer1 ? placeOffset : gameMap.Width - placeOffset;
    vehicle2.PosX = !isPlayer1 ? placeOffset : gameMap.Width - placeOffset;

    vehicle1.Color = Color.Red;
    vehicle2.Color = Color.Blue;

    Invoke((Action)(() =>
    {
        SetStrategies();
    }));

    gameMap.PlaceAndDisplayUnitAboveXY(vehicle1, vehicle1.PosX, gameMap.Height - 1);
    gameMap.PlaceAndDisplayUnitAboveXY(vehicle2, vehicle2.PosX, gameMap.Height - 1);

    Invoke((Action)(() =>
    {
        pictureBox1.Refresh();
    }));
}
```

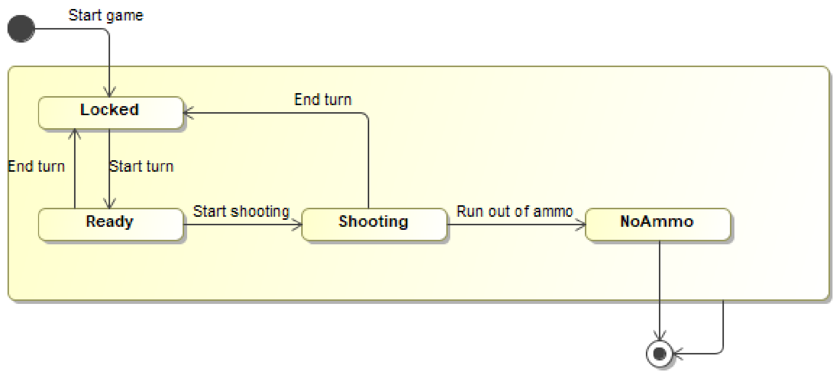
Main.cs(ObserverIterator panaudojimas)

```
private readonly ObserverIterator observers;
private HubConnection connection;
private readonly GameHubReceiver gameHub;
public Form1 Form { get; }
public Facade GameFacade { get; }

public Main(Form1 form)
{
    Form = form;
    observers = new ObserverIterator();
    gameHub = new GameHubReceiver(this);
    GameFacade = new Facade();
}

public void AddObserver(IObserver observer)
{
    observers.Add(observer);
}

public void RemoveObserver(IObserver observer)
{
    observers.Remove(observer);
}

public void StartGame(Player player1, Player player2, bool isPlayer1)
{
    AddObserver(player1);
    AddObserver(player2);
    Form.StartGame(isPlayer1, player1.Vehicle.Type, player2.Vehicle.Type);
    Console.WriteLine($"Starting game with {player1.Name} and {player2.Name}");
}

public async void NextTurn(int posX, int posY)
{
    for (var observer = observers.First(); observers.HasNext(); observer = observers.Next())
    {
        observer.OnAction();
    }

    await connection.InvokeCoreAsync("EndTurn", new object[]
    {
        gameHub.PlayerId,
        posX,
        posY
    });
}
```

4.11. State šablonas



17 pav. State šablono klasių diagrama



18 pav. State šablono būsenų diagrama

ShootingStrategy.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Shots;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.ShootingStrategies.States;
using Nemare.Models.Entities.Templates;

namespace Nemare.Models.Entities.ShootingStrategies
{
    public abstract class ShootingStrategy : ShootTemplate
    {
        public ShootingState State { get; set; }
        public ShotType Shot { get; set; }
        public string Name { get; set; }
        public int Ammo { get; set; }
        public override void SwitchState()
        {
            State.SwitchState(this);
        }

        public override void ReduceAmmo()
        {
            Ammo--;
        }

        public void SetLockedState()
        {
            State = new Locked();
        }
    }
}
```

ShootingState.cs:

```
namespace Nemare.Models.Entities.ShootingStrategies.States
{
    public interface ShootingState
    {
        void SwitchState(ShootingStrategy strategy);
    }
}
```

Locked.cs:

```
namespace Nemare.Models.Entities.ShootingStrategies.States
{
    public class Locked : ShootingState
    {
        public void SwitchState(ShootingStrategy strategy)
        {
            strategy.State = new Ready();
        }
    }
}
```

Ready.cs:

```
namespace Nemare.Models.Entities.ShootingStrategies.States
{
    public class Ready : ShootingState
    {
        public void SwitchState(ShootingStrategy strategy)
        {
            strategy.State = new Shooting();
        }
    }
}
```

Shooting.cs:

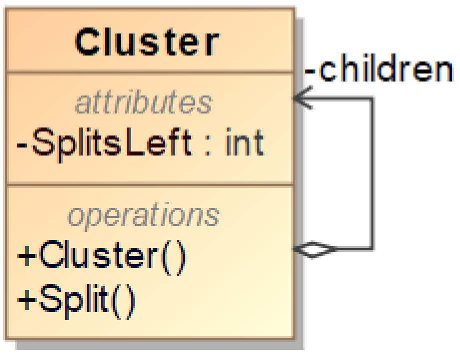
```
namespace Nemare.Models.Entities.ShootingStrategies.States
```

```
{
    public class Shooting : ShootingState
    {
        public void SwitchState(ShootingStrategy strategy)
        {
            if (strategy.Ammo == 0)
            {
                strategy.State = new NoAmmo();
            }
            else
            {
                strategy.State = new Locked();
            }
        }
    }
}
```

NoAmmo.cs:

```
namespace Nemare.Models.Entities.ShootingStrategies.States
{
    public class NoAmmo : ShootingState
    {
        public void SwitchState(ShootingStrategy strategy)
        {
            strategy.State = new NoAmmo(); // Might change to "Ready" if we implement ammo
pickups.
        }
    }
}
```

4.12. Composite šablonas



pav. 19 composite šablono klasių diagrama

Cluster.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Nemare.Models.Entities.Map;
using Nemare.Models.Entities.Vehicles;
using Nemare.Models.Interfaces;

namespace Nemare.Models.Entities.Shots
{
    class Cluster : ShotTypeDecorator
    {
        private List<Cluster> children;
        public Cluster(ShotType shot) : base(shot) {
            children = new List<Cluster>();
        }

        public override void Collide(DirectBitmap gameMap, double attack)
        {
            base.Collide(gameMap, attack);
            Console.WriteLine("child count: " + children.Count);
            if (children.Count > 0)
            {
                Split(gameMap, attack);
            }
        }

        public void AddChild(Cluster child)
        {
            children.Add(child);
        }

        public void RemoveChild(Cluster child)
        {
            for (int i = 0; i < children.Count; i++)
            {
                if (children[i].Equals(child)) {
                    children.Remove(child);
                    break;
                }
            }
        }

        public Cluster GetChild(int index)
        {
            if (index < 0 || index >= children.Count)
                throw new Exception("index out of bounds");
            else
                return children[index];
        }

        private void Split(DirectBitmap gameMap, double attack)
        {
            var power = 15;
            var projectile = shot;
            List<double> angles = new List<double>();
            var firstDisplay = true;
            double startY;
            var angleStep = 180 / (children.Count + 1);
            List<int> childrenDone = new List<int>();

            for (int i = 0; i < children.Count; i++)
            {
                children[i].PosX = PosX;
                children[i].PosY = PosY;
                angles.Add((i + 1) * angleStep);
                gameMap.PlaceAndDisplayUnitAboveXY(children[i], children[i].PosX,
children[i].PosY);
            }
            startY = children[0].PosY;

            var stopwatch = new Stopwatch();
            stopwatch.Start();
            var targetTime = 0 + gameMap.frameInterval;
            double currentTime = 0;
            for (double t = 0; t < 8000; t += gameMap.frameInterval)
            {
                while (currentTime > targetTime)
                {
                    currentTime = (double)stopwatch.ElapsedMilliseconds / 1000;
                }
                targetTime += gameMap.frameInterval;

                for (int i = 0; i < children.Count; i++)
                {
                    if (childrenDone.Contains(i)) // if projectile is not active
                    {
                        continue;
                    }

                    projectile = children[i];
                    var angle = angles[i];


                    if (!firstDisplay)
                        gameMap.Restore(projectile);

                    projectile.PosX = PosX - gameMap.PosXatTime(projectile, t, power,
angle);
                    projectile.PosY = startY - gameMap.PosYatTime(projectile, t, power,
angle);
                    if (projectile.PosX > 0 && projectile.PosX < gameMap.Width &&
projectile.PosY > 0 && projectile.PosY < gameMap.Height)
                    {
                        var collided = gameMap.CheckUnitCollision(projectile);
                        if (collided)
                        {
                            projectile.Collide(gameMap, attack);
                            //gameMap.Restore(projectile);
                            childrenDone.Add(i);
                        }
                        else
                        {
                            gameMap.Display(projectile);
                            gameMap.Refresh();
                        }
                    }
                    else
                    {
                        childrenDone.Add(i);
                    }

                    if(childrenDone.Count == children.Count)
                    {
                        break;
                    }
                }
                firstDisplay = false;
            }

            stopwatch.Stop();

            gameMap.Refresh();
        }
    }
}
```