KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Objektinis programų projektavimas (T120B516)

Projektinio darbo ataskaita

Atliko:

Mildaras Karvelis IFF 1/4

Ugnė Petrauskaitė IFF 1/5

Priėmė:

doc. prakt. VALINČIUS Kęstutis

KAUNAS 2024

TURINYS

1. Pirmoji projekto dalis

1.1. Žaidimas – Tower Defence Game

„Tower Defense“ žaidime žaidėjas stato bokštus, kad apsigintų nuo priešiškų būtybių, kurios juda link žaidėjo bazės būriais arba „waves“. Kiekvienas būrys tampa vis sudetingesnis – priešai tampa stipresni ir greitesni, tad žaidėjas turi nuolat tobulinti gynybą.

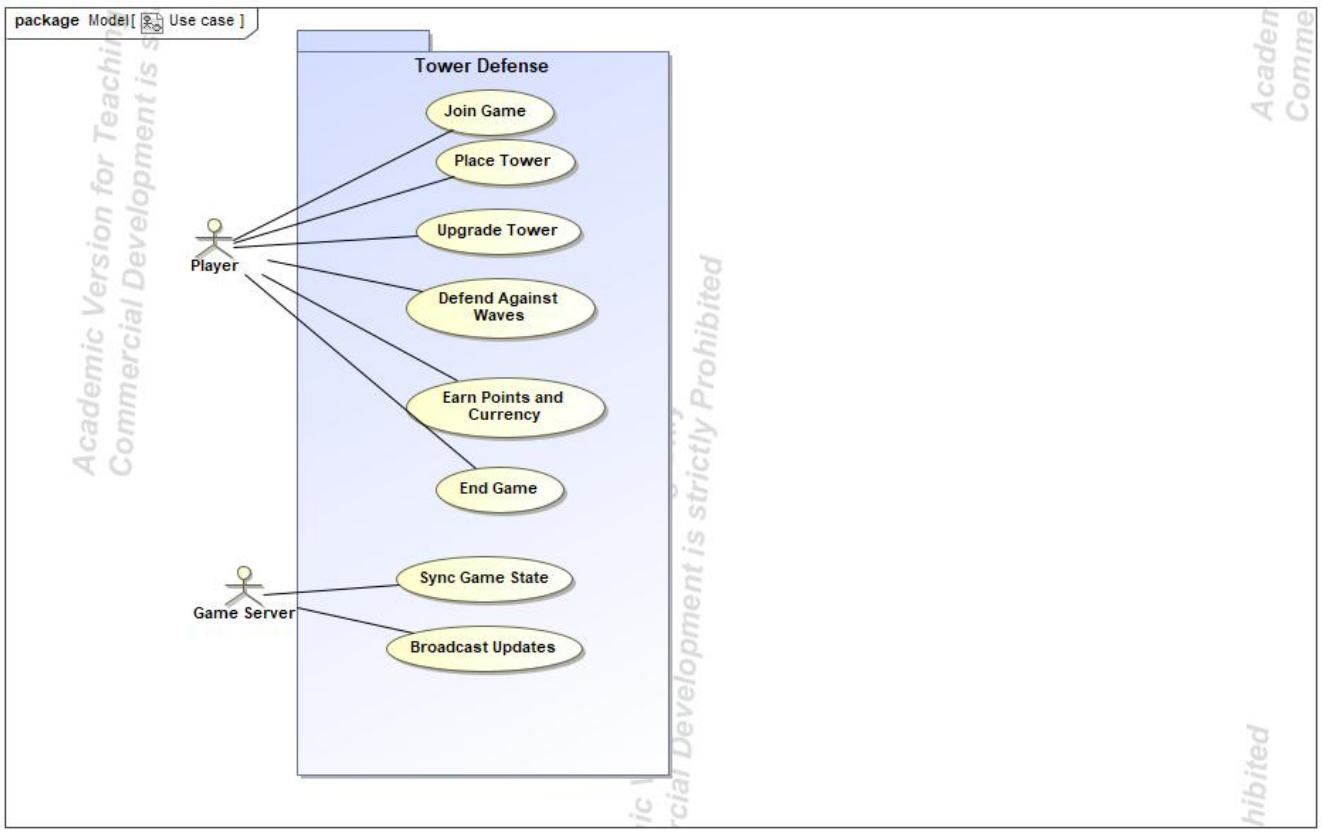
Žaidimo pradžioje žaidėjas turi šiek tiek aukso, kurį gali išleisti statydamas pirmuosius bokštus. Bokštus galima gerinti, kad darytų daugiau žalos.

Kiekvieną kartą, kai bokštas sunaikina priešą, žaidėjas gauna šiek tiek aukso, kurį gali panaudoti naujų bokštų statymui arba esamų atnaujinimui. Kuo daugiau aukso žaidėjas surenka, tuo galingesnę gynybą gali sukurti, kas leidžia išgyventi vis didesnį skaičių priešų bangų. Taip žaidimas tampa strategine užduotimi, kur žaidėjas turi nuolat priimti sprendimus, kur ir kokius bokštus statyti ar atnaujinti, kad išlaikytų tvirtą gynybos liniją prieš nenutrūkstančias priešų bangas.

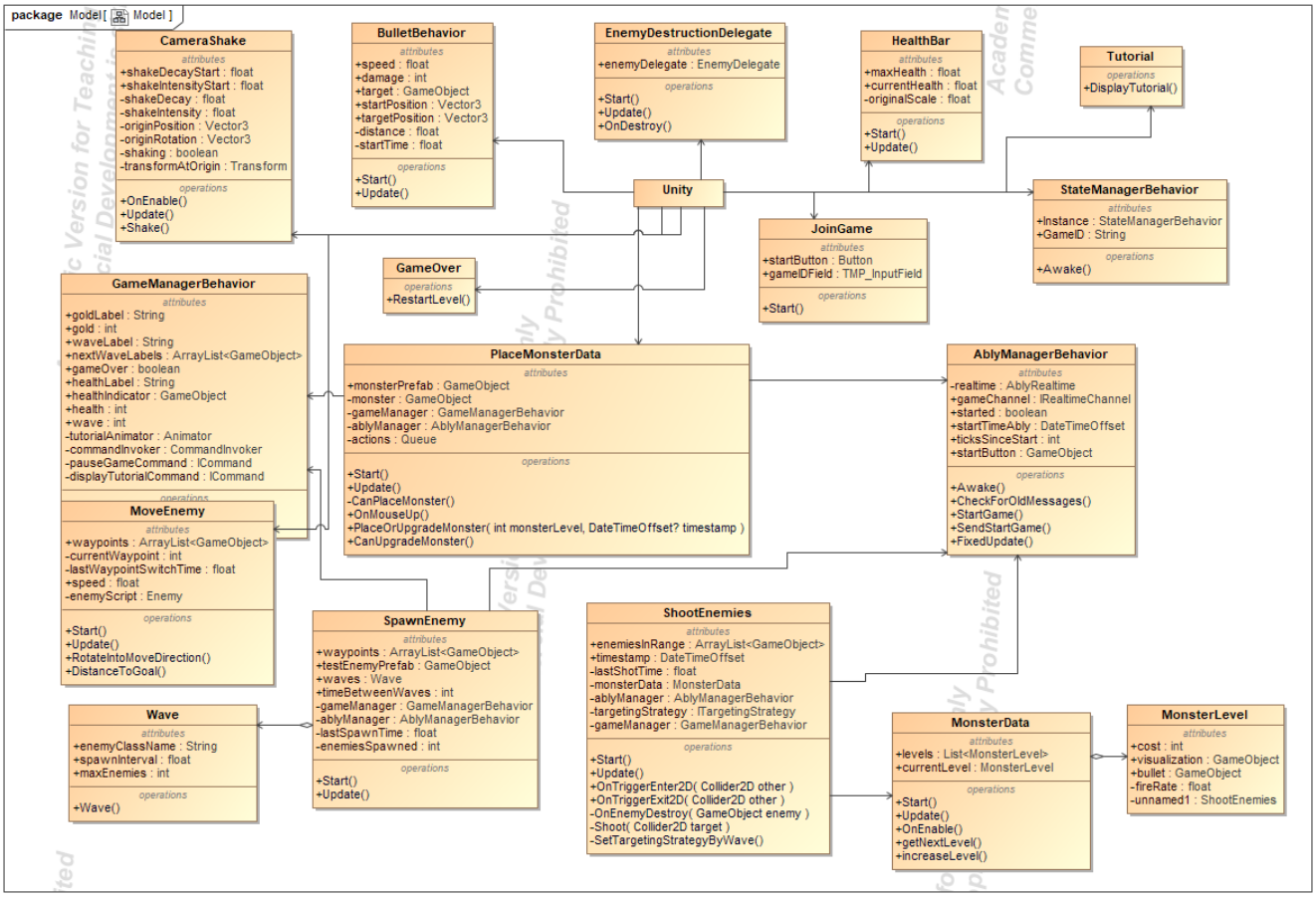
1.2. Multiplayer dalis

Abu žaidėjai, prisijungę prie to pačio kambario, gali valdyti žaidimą: delioti bokštus, startuoti žaidimą, dalinasi ta pačia pinigine ir kovoja prieš tuos pačius priešus.

1.3. Use Case diagrama

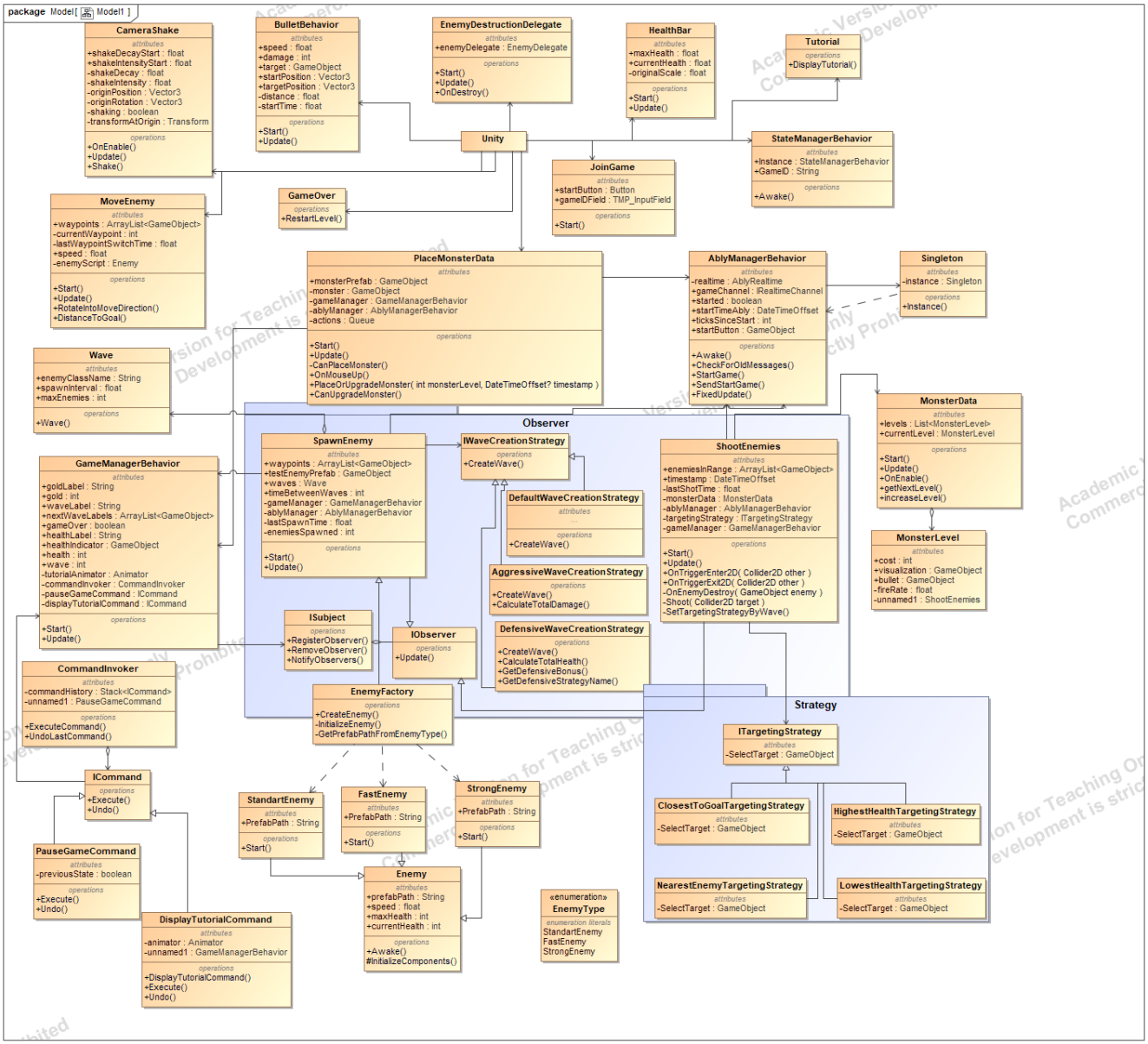


1.4. UML diagrama prieš pakeitimus



1 pav. Žaidimo UML diagrama be „design patterns“

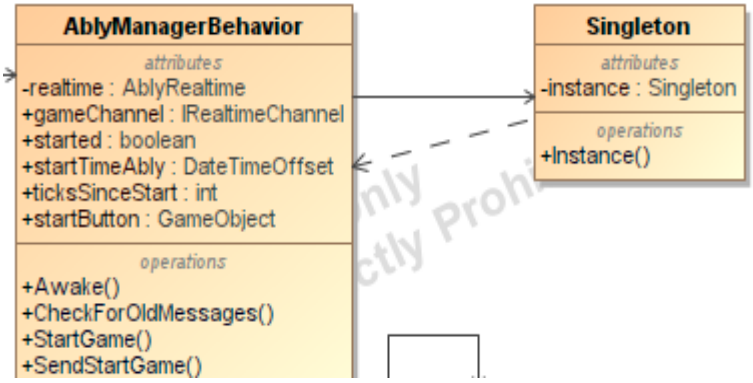
1.5. UML diagrama su pakeitimais



2 pav. Žaidimo UML diagrama su realizuotais „design patterns“

1.6. Realizuoti šablonai

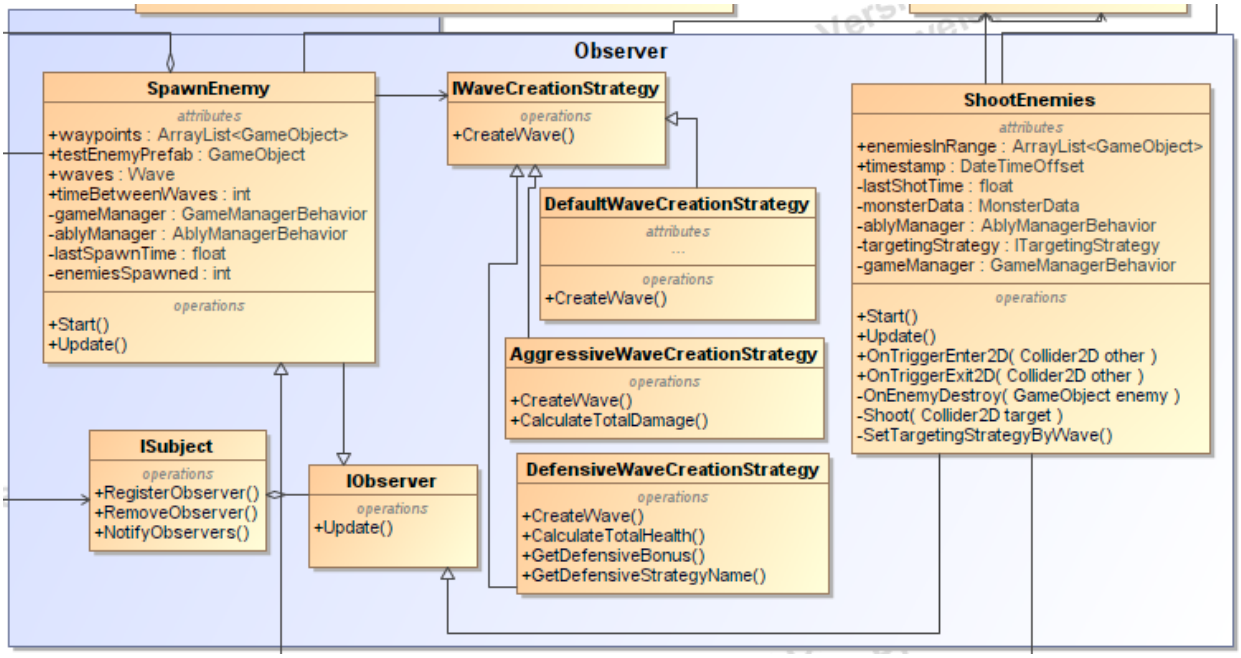
1.6.1 Singleton



```
public class Singleton<T> where T : class
{
    // Private static variable to hold the single instance
    private static T instance;

    // Public static property to provide access to the instance
    public static T Instance
    {
        get
        {
            return instance;
        }
        set
        {
            if (instance == null)
            {
                instance = value;
            }
        }
    }
}
```

```
  void Start()
  {
      if (Singleton<AblyManagerBehavior>.Instance == null)
      {
          Singleton<AblyManagerBehavior>.Instance = this;
          DontDestroyOnLoad(gameObject);
          InitializeAbly();
      }
      else
      {
          Destroy(gameObject);
      }
  }
    You, 6 days ago • Kazkiek multiplayerio ir singletronas
```

1.6.2 Observer

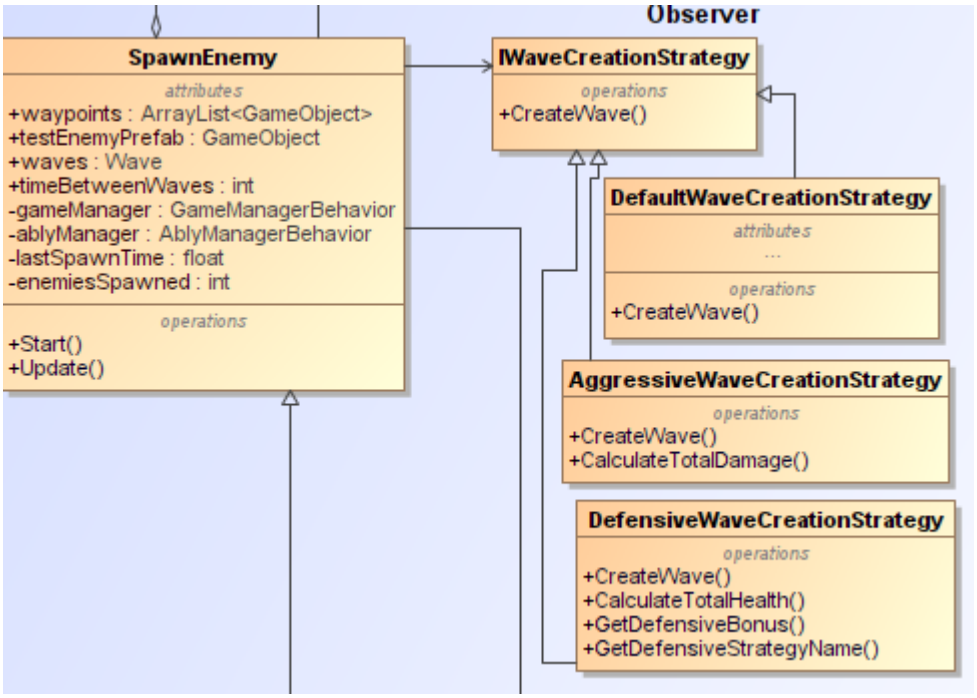



```
public interface IObserver
{
    void Update();
}
```

```
public interface ISubject
{
    void RegisterObserver(IObserver observer);
    void RemoveObserver(IObserver observer);
    void NotifyObservers();
}
```

```
private List<IObserver> observers = new List<IObserver>();

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

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

public void NotifyObservers()
{
    foreach (IObserver observer in observers)
    {
        observer.Update();
    }
}
```

1.6.3 Adapter




```
public interface IWaveCreationStrategy
{
    Wave CreateWave(int waveNumber);
}
```

```
public class AggressiveWaveCreationStrategy : IWaveCreationStrategy
{
    public Wave CreateWave(int waveNumber)
    {
        float spawnInterval = Mathf.Max(0.3f, 2.0f - waveNumber * 0.15f);
        int maxEnemies = Mathf.CeilToInt(15 * Mathf.Pow(1.3f, waveNumber));
        return new Wave("", spawnInterval, maxEnemies);
    }

    public int CalculateTotalDamage(int waveNumber)
    {
        int baseDamage = 10;
        return baseDamage * waveNumber * 2;
    }
}
```

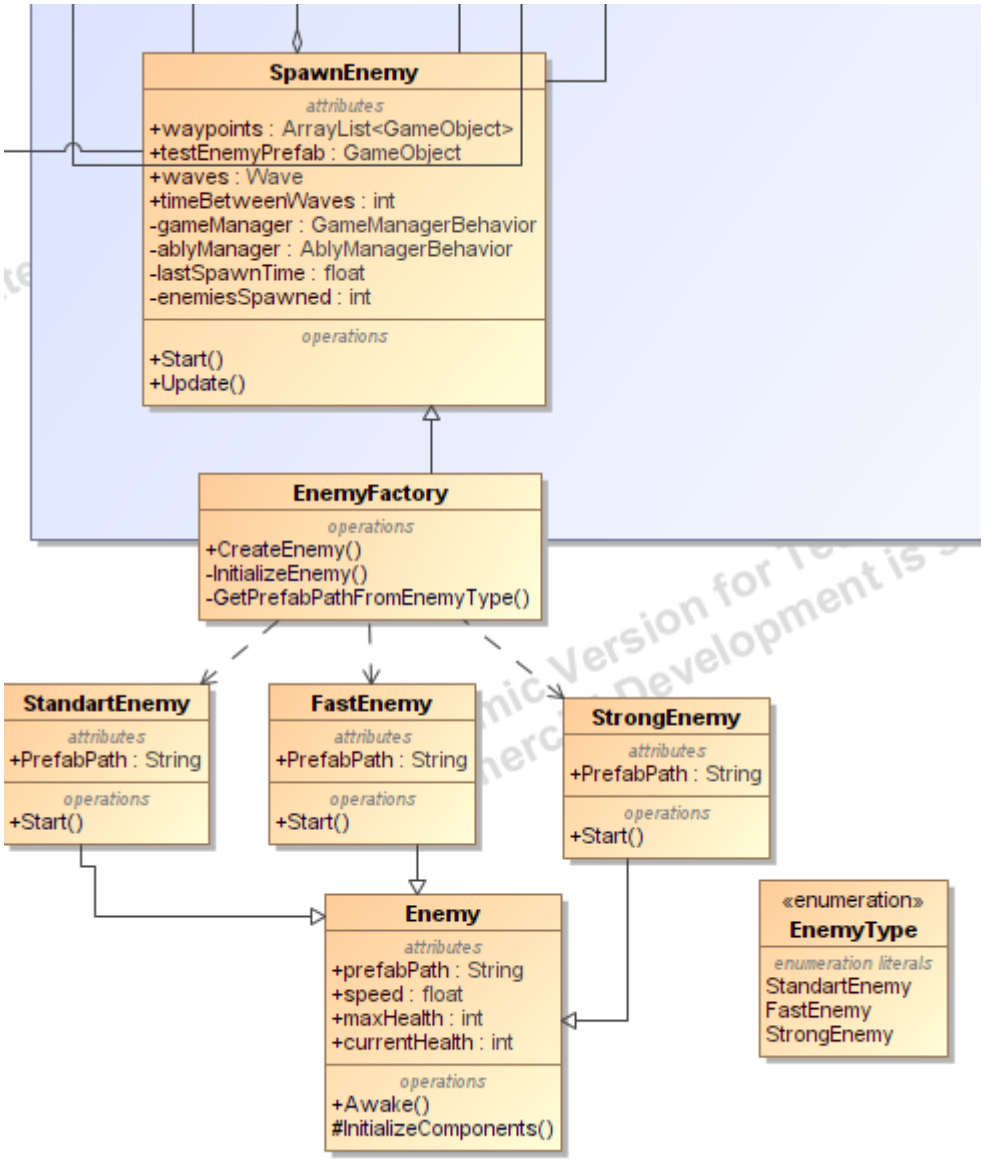
```
public class DefaultWaveCreationStrategy : IWaveCreationStrategy
{
    public Wave CreateWave(int waveNumber)
    {
        float spawnInterval = Mathf.Max(0.5f, 2.5f - waveNumber * 0.1f);
        int maxEnemies = Mathf.CeilToInt(10 * Mathf.Pow(1.2f, waveNumber));
        return new Wave("", spawnInterval, maxEnemies);
    }
}
```

```
public class DefensiveWaveCreationStrategy : IWaveCreationStrategy
{
    public Wave CreateWave(int waveNumber)
    {
        float spawnInterval = Mathf.Max(0.7f, 3.0f - waveNumber * 0.05f);
        int maxEnemies = Mathf.CeilToInt(8 * Mathf.Pow(1.1f, waveNumber));
        return new Wave("", spawnInterval, maxEnemies);
    }

    public int CalculateTotalHealth(int waveNumber)
    {
        int baseHealth = 100;
        return baseHealth * waveNumber * 3;
    }

    public float GetDefensiveBonus(int waveNumber)
    {
        return 1.0f + (waveNumber * 0.05f);
    }

    public string GetDefensiveStrategyName()
    {
        return "Defensive Strategy";
    }
}
```

```
private void SetWaveCreationStrategy(int waveNumber)
{
    if (waveNumber % 3 == 0)
    {
        waveCreationStrategy = new AggressiveWaveCreationStrategy();
        Debug.Log("Total Damage: " + ((AggressiveWaveCreationStrategy)waveCreationStrategy).CalculateTotalDamage(waveNumber));
    }
    else if (waveNumber % 3 == 1)
    {
        waveCreationStrategy = new DefensiveWaveCreationStrategy();
        Debug.Log("Total Health: " + ((DefensiveWaveCreationStrategy)waveCreationStrategy).CalculateTotalHealth(waveNumber));
        Debug.Log("Defensive Bonus: " + ((DefensiveWaveCreationStrategy)waveCreationStrategy).GetDefensiveBonus(waveNumber));
        Debug.Log("Strategy Name: " + ((DefensiveWaveCreationStrategy)waveCreationStrategy).GetDefensiveStrategyName());
    }
    else
    {
        waveCreationStrategy = new DefaultWaveCreationStrategy();
    }
}
```

```
private IWaveCreationStrategy waveCreationStrategy;
```

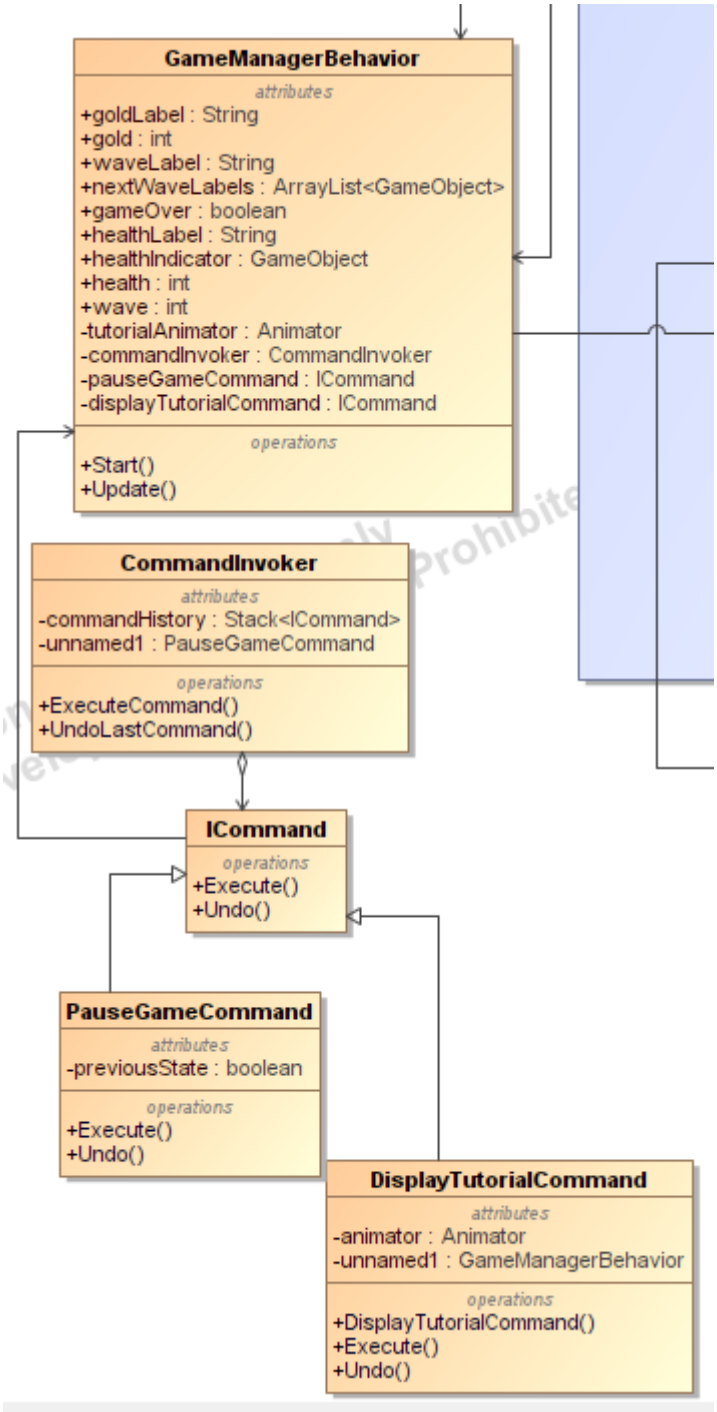
1.6.4 Factory



public static class EnemyFactory

```
{
    public static GameObject CreateEnemy(Type enemyType, Vector3 position, Quaternion rotation, GameObject[] waypoints)
    {
        string prefabPath = GetPrefabPathFromEnemyType(enemyType);
        if (string.IsNullOrEmpty(prefabPath))
        {
            return null;
        }

        GameObject enemyPrefab = Resources.Load<GameObject>(prefabPath);
        if (enemyPrefab == null)
        {
            return null;
        }

        GameObject newEnemy = GameObject.Instantiate(enemyPrefab, position, rotation);
        newEnemy.AddComponent(enemyType);

        InitializeEnemy(newEnemy, waypoints);
        return newEnemy;
    }

    private static string GetPrefabPathFromEnemyType(Type enemyType)
    {
        var propertyInfo = enemyType.GetProperty("PrefabPath", BindingFlags.Public | BindingFlags.Instance | BindingFlags.FlattenHierarchy);

        if (propertyInfo != null)
        {
            Enemy enemyInstance = Activator.CreateInstance(enemyType) as Enemy;
            return propertyInfo.GetValue(enemyInstance) as string;
        }
        else
        {
            return null;
        }
    }

    private static void InitializeEnemy(GameObject enemy, GameObject[] waypoints)
    {
        MoveEnemy moveEnemy = enemy.GetComponent<MoveEnemy>();
        if (moveEnemy != null)
        {
            moveEnemy.waypoints = waypoints;
        }
    }
}
```

1.6.5 Command

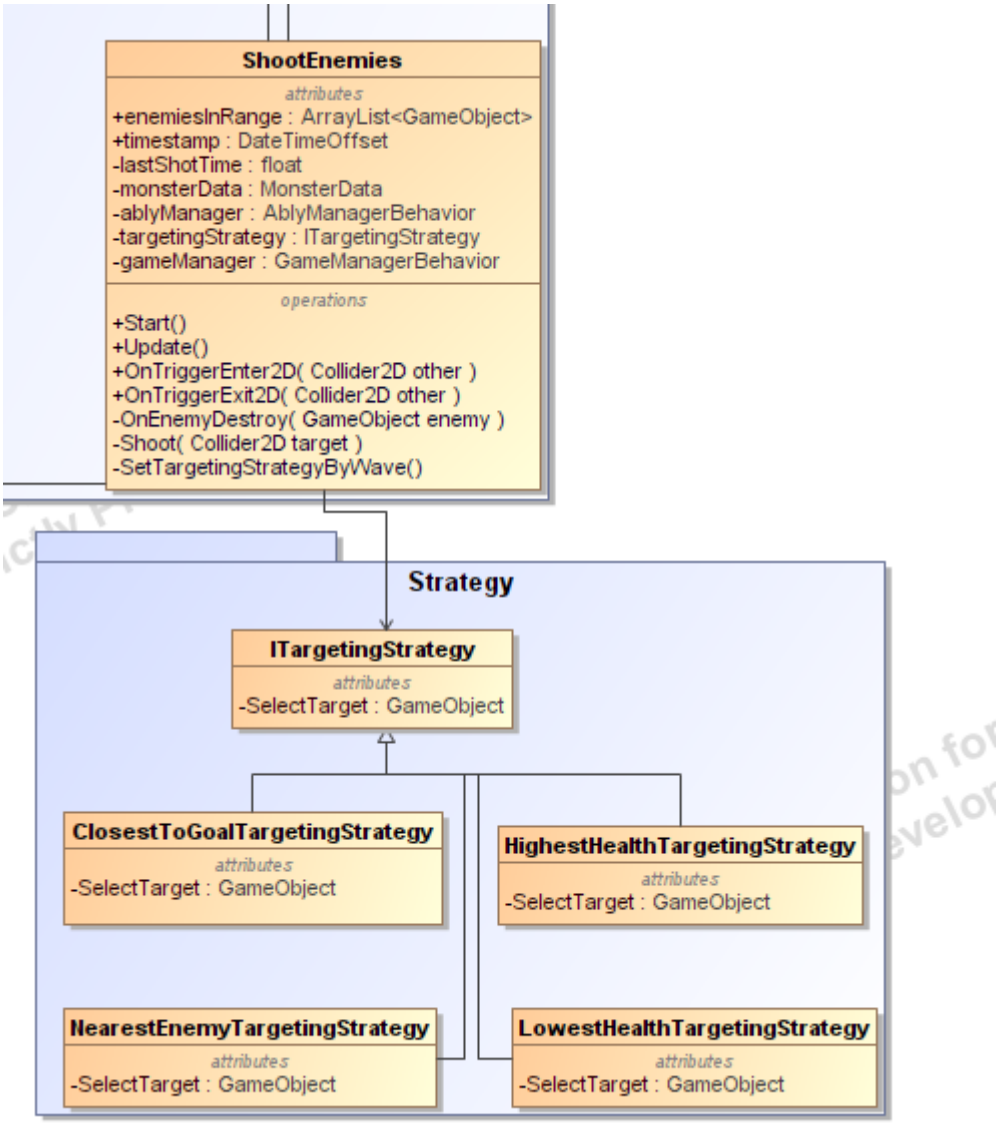



```
public interface ICommand
{
    void Execute();
    void Undo();
}
```

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.T))
    {
        commandInvoker.ExecuteCommand(displayTutorialCommand);
    }

    if (Input.GetKeyDown(KeyCode.P))
    {
        commandInvoker.ExecuteCommand(pauseGameCommand);
    }

    if (Input.GetKeyDown(KeyCode.Z))
    {
        commandInvoker.UndoLastCommand();
    }
}
```

```
public class CommandInvoker
{
    private Stack<ICommand> commandHistory = new Stack<ICommand>();

    public void ExecuteCommand(ICommand command)
    {
        command.Execute();
        commandHistory.Push(command);
    }

    public void UndoLastCommand()
    {
        if (commandHistory.Count > 0)
        {
            ICommand lastCommand = commandHistory.Pop();
            lastCommand.Undo();
        }
    }
}
```

1.6.6 Strategy




```
public interface ITargetingStrategy
{
    GameObject SelectTarget(List<GameObject> enemiesInRange);
}
```

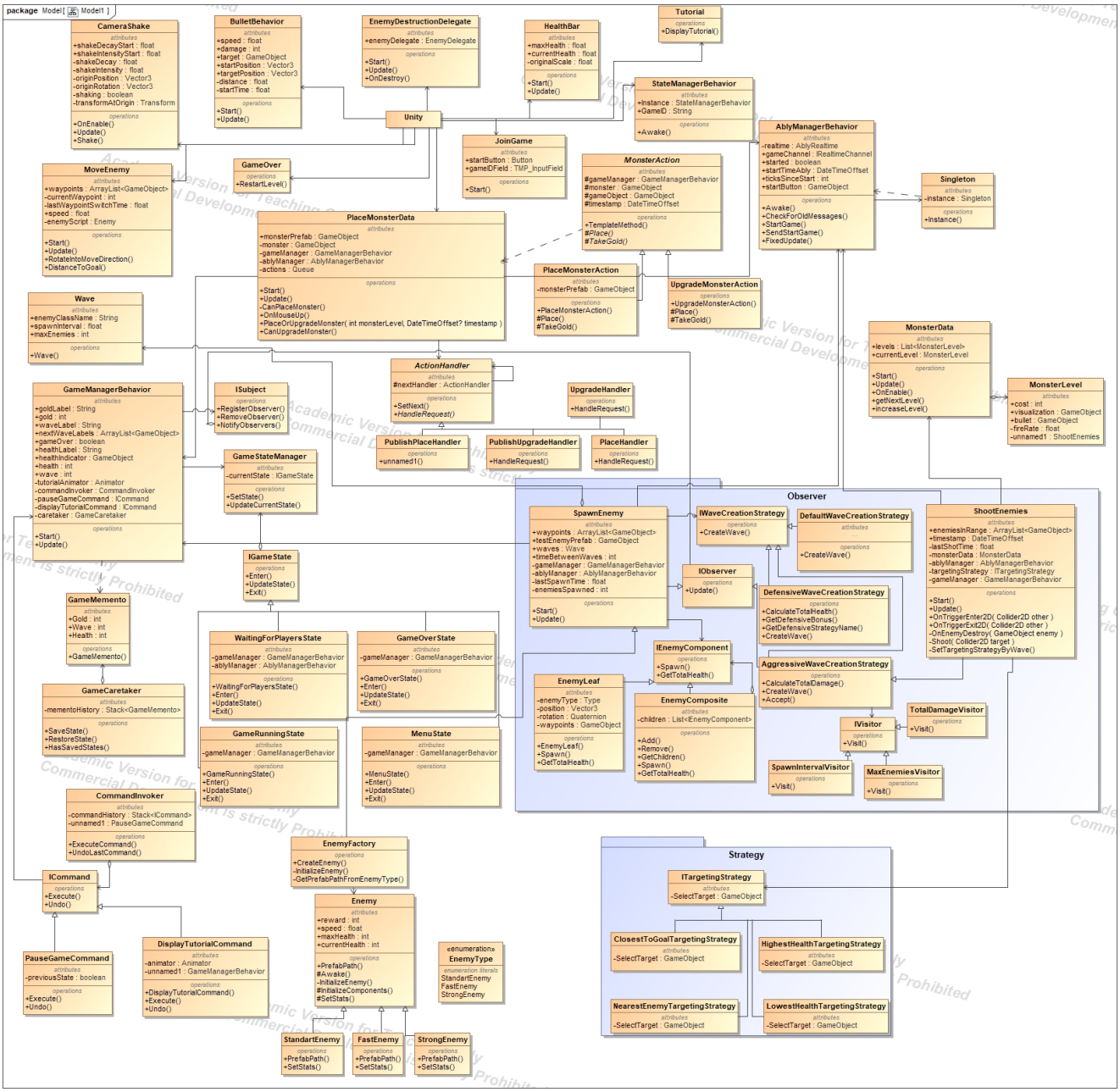
```
public class NearestEnemyTargetingStrategy : ITargetingStrategy
{
    private Transform towerTransform;

    public NearestEnemyTargetingStrategy(Transform towerTransform)
    {
        this.towerTransform = towerTransform;
    }

    public GameObject SelectTarget(List<GameObject> enemiesInRange)
    {
        GameObject target = null;
        float minimalDistance = float.MaxValue;

        foreach (GameObject enemy in enemiesInRange)
        {
            float distance = Vector3.Distance(enemy.transform.position, towerTransform.position);
            if (distance < minimalDistance)
            {
                target = enemy;
                minimalDistance = distance;
            }
        }

        return target;
    }
}
```

```
public class ClosestToGoalTargetingStrategy : ITargetingStrategy
{
    public GameObject SelectTarget(List<GameObject> enemiesInRange)
    {
        GameObject target = null;
        float minimalDistance = float.MaxValue;

        foreach (GameObject enemy in enemiesInRange)
        {
            float distanceToGoal = enemy.GetComponent<MoveEnemy>().DistanceToGoal();
            if (distanceToGoal < minimalDistance)
            {
                target = enemy;
                minimalDistance = distanceToGoal;
            }
        }

        return target;
    }
}
```

```
public class HighestHealthTargetingStrategy : ITargetingStrategy
{
    public GameObject SelectTarget(List<GameObject> enemiesInRange)
    {
        GameObject target = null;
        float highestHealth = float.MinValue;

        foreach (GameObject enemy in enemiesInRange)
        {
            Transform healthBarTransform = enemy.transform.Find("HealthBar");
            if (healthBarTransform != null)
            {
                HealthBar healthBar = healthBarTransform.GetComponent<HealthBar>();
                if (healthBar != null)
                {
                    float enemyHealth = healthBar.currentHealth;
                    if (enemyHealth > highestHealth)
                    {
                        target = enemy;
                        highestHealth = enemyHealth;
                    }
                }
            }
        }

        return target;
    }
}
```

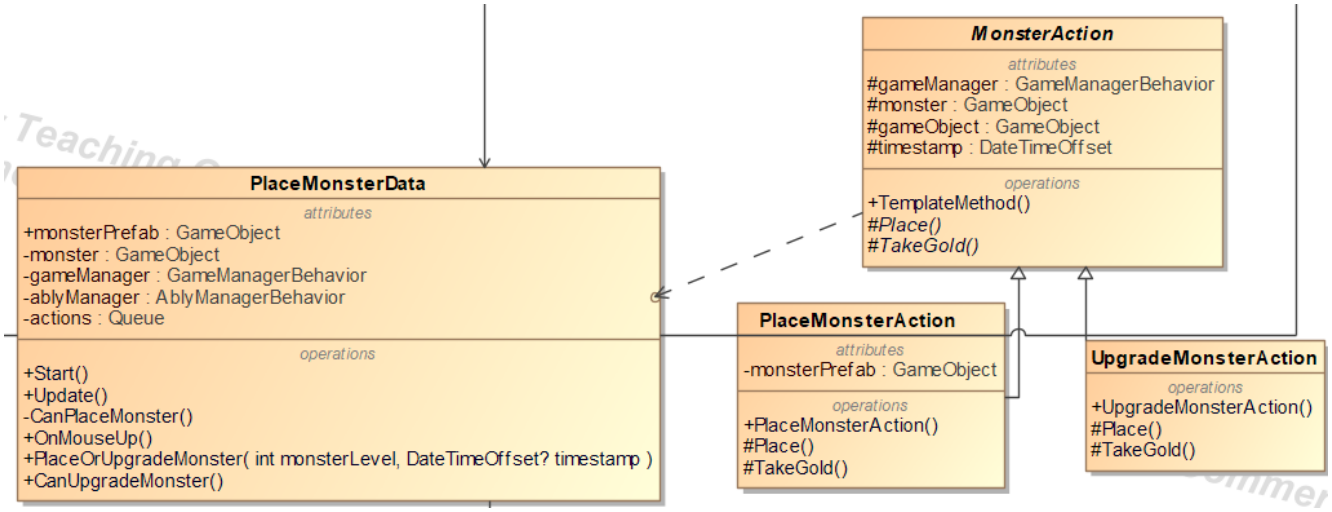
2. Antroji projekto dalis

2.1. UML diagrama po naujų šablonų pridėjimo



2.2. Realizuoti šablonai

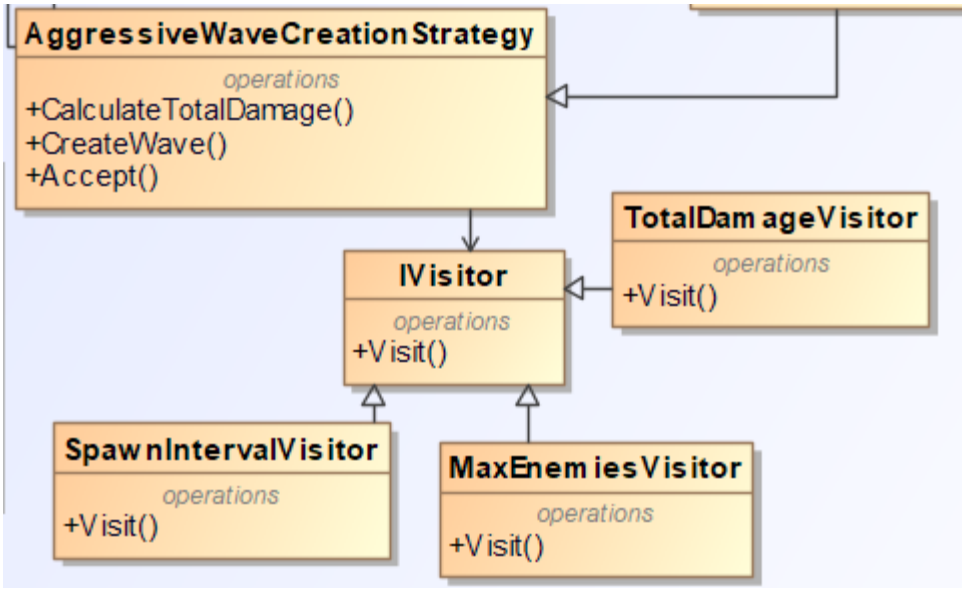
2.2.1 Template Method




```
public abstract class MonsterAction
{
    protected GameManagerBehavior gameManager;
    protected GameObject monster;
    protected GameObject gameObject;
    protected DateTimeOffset? timestamp;

    // Template Method
    public void TemplateMethod()
    {
        Place();    // Place or upgrade
        TakeGold(); // Deduct gold
    }

    // Abstract methods to be implemented by concrete classes
    protected abstract void Place();
    protected abstract void TakeGold();
}

public class PlaceMonsterAction : MonsterAction
{
    private GameObject monsterPrefab;

    public PlaceMonsterAction(GameObject monsterPrefab, GameManagerBehavior gameManager,
GameObject gameObject, DateTimeOffset? timestamp)
    {
        this.monsterPrefab = monsterPrefab;
        this.gameManager = gameManager;
        this.gameObject = gameObject;
        this.timestamp = timestamp;
    }

    protected override void Place()
    {
        monster = GameObject.Instantiate(monsterPrefab, gameObject.transform.position,
Quaternion.identity);
```

```
        monster.GetComponent<ShootEnemies>().timestamp = timestamp;

        AudioSource audioSource = gameObject.GetComponent<AudioSource>();
        audioSource.PlayOneShot(audioSource.clip);
    }

    protected override void TakeGold()
    {
        gameManager.Gold -= monster.GetComponent<MonsterData>().CurrentLevel.cost;
    }
}

public class UpgradeMonsterAction : MonsterAction
{
    public UpgradeMonsterAction(GameObject monster, GameManagerBehavior gameManager,
DateTimeOffset? timestamp)
    {
        this.monster = monster;
        this.gameManager = gameManager;
        this.timestamp = timestamp;
    }

    protected override void Place()
    {
        MonsterData monsterData = monster.GetComponent<MonsterData>();
        monsterData.increaseLevel();
        monster.GetComponent<ShootEnemies>().timestamp = timestamp;

        AudioSource audioSource = monster.GetComponent<AudioSource>();
        audioSource.PlayOneShot(audioSource.clip);
    }

    protected override void TakeGold()
    {
        gameManager.Gold -= monster.GetComponent<MonsterData>().CurrentLevel.cost;
    }
}
```

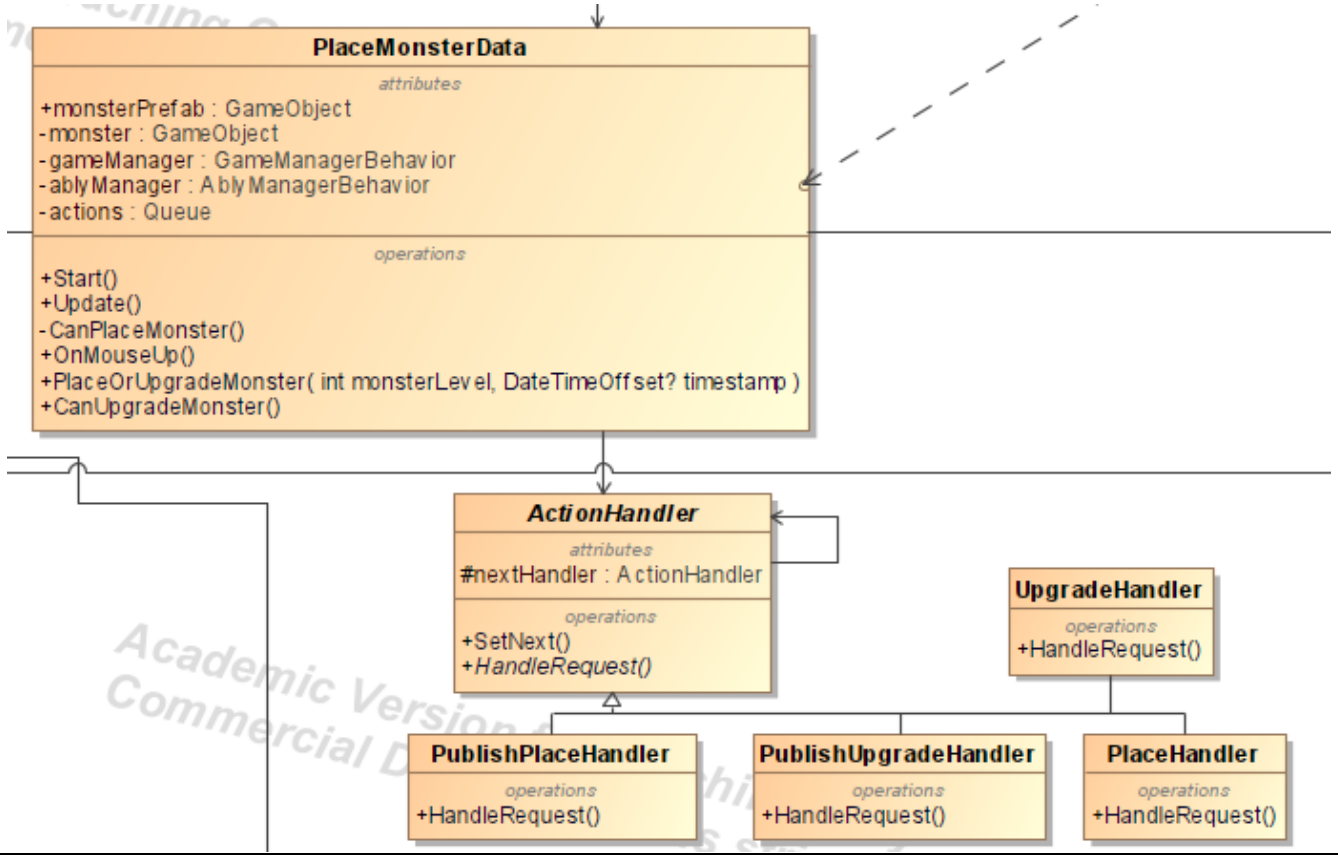
2.2.2 Visitor




```
public class AggressiveWaveCreationStrategy : IWaveCreationStrategy
{
    public Wave CreateWave(int waveNumber)
    {
        float spawnInterval = Mathf.Max(0.3f, 2.0f - waveNumber * 0.15f);
        int maxEnemies = Mathf.CeilToInt(15 * Mathf.Pow(1.3f, waveNumber));
        return new Wave("", spawnInterval, maxEnemies);
    }

    public int CalculateTotalDamage(int waveNumber)
    {
        int baseDamage = 10;
        return baseDamage * waveNumber * 2;
    }

    public void Accept(IVisitor visitor)
    {
        visitor.Visit(this);
    }
}

public interface IVisitor
{
    void Visit(AggressiveWaveCreationStrategy strategy, int waveNumber);
}

public class SpawnIntervalVisitor : IVisitor
{
    public void Visit(AggressiveWaveCreationStrategy strategy, int waveNumber)
    {
        Debug.Log("Spawn Interval: " + strategy.CreateWave(waveNumber).spawnInterval);
    }
}

public class MaxEnemiesVisitor : IVisitor
```

```
{
    public void Visit(AggressiveWaveCreationStrategy strategy, int waveNumber)
    {
        Debug.Log("Max Enemies: " + strategy.CreateWave(waveNumber).maxEnemies);
    }
}

public class TotalDamageVisitor : IVisitor
{
    public void Visit(AggressiveWaveCreationStrategy strategy, int waveNumber)
    {
        Debug.Log("Total Damage: " + strategy.CalculateTotalDamage(waveNumber));
    }
}
```

2.2.3 Chain of Responsibility

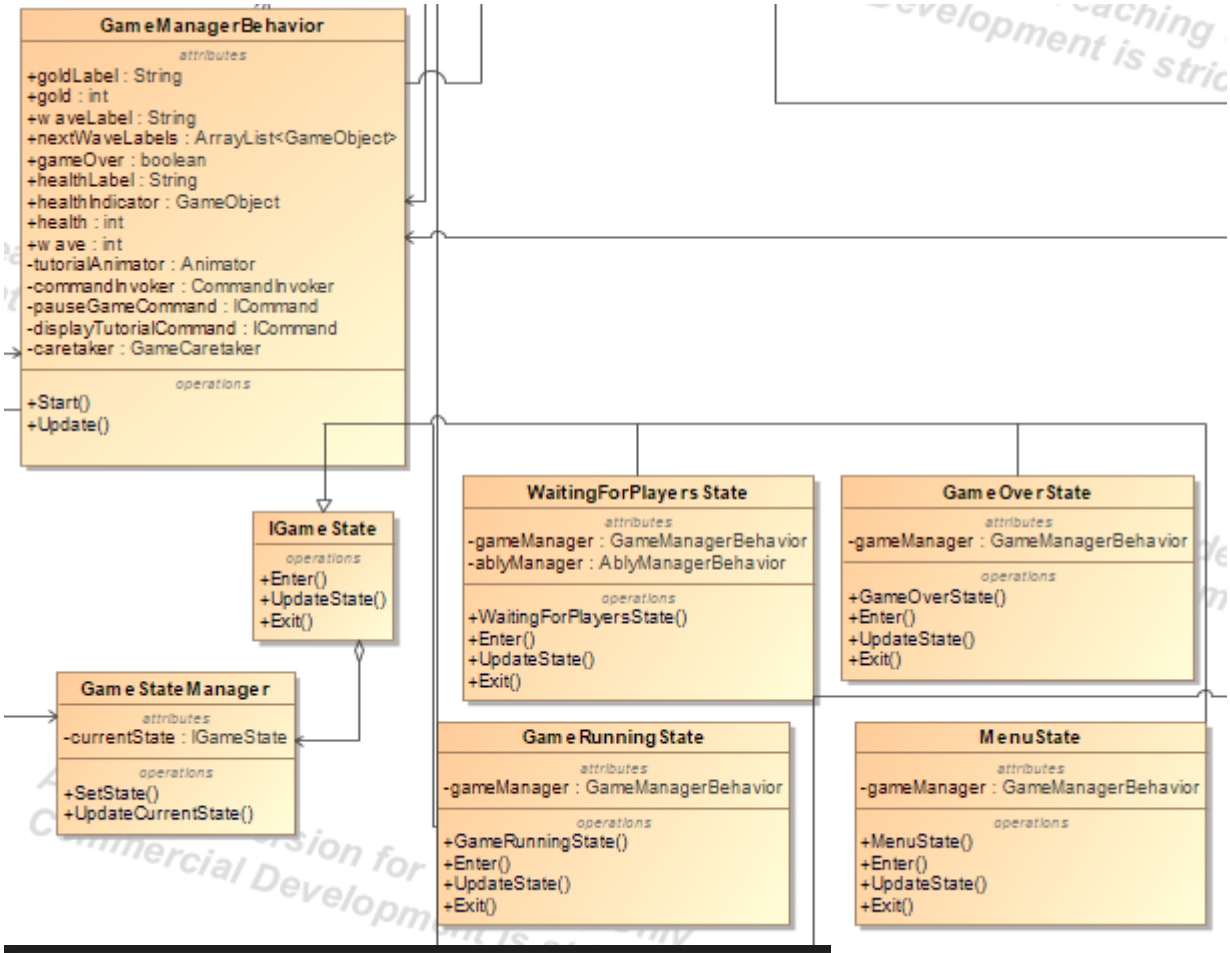



```
public abstract class ActionHandler
{
    protected ActionHandler nextHandler;

    public void SetNext(ActionHandler nextHandler)
    {
        this.nextHandler = nextHandler;
    }

    public abstract void HandleRequest(string action, PlaceMonster placeMonster, PlaceMonsterData
data);
```

```
}



public class PublishPlaceHandler : ActionHandler
{
    public override void HandleRequest(string action, PlaceMonster placeMonster, PlaceMonsterData
data)
    {
        if (action == "publish_place")
        {
            placeMonster.actions.Enqueue(data);
            Debug.Log("Publishing place action...");
        }
        else if (nextHandler != null)
        {
            nextHandler.HandleRequest(action, placeMonster, data);
        }
    }
}


public class PublishUpgradeHandler : ActionHandler
{
    public override void HandleRequest(string action, PlaceMonster placeMonster, PlaceMonsterData
data)
    {
        if (action == "publish_upgrade")
        {
            placeMonster.actions.Enqueue(data);
            Debug.Log("Publishing upgrade action...");
        }
        else if (nextHandler != null)
        {
            nextHandler.HandleRequest(action, placeMonster, data);
        }
    }
}


public class PlaceHandler : ActionHandler
{
    public override void HandleRequest(string action, PlaceMonster placeMonster, PlaceMonsterData
data)
    {
        if (action == "place")
        {
            var monsterAction = new PlaceMonsterAction(placeMonster.monsterPrefab,
PlaceMonster.gameManager, placeMonster.gameObject, data.timestamp);
            monsterAction.TemplateMethod(); // Execute place action
            Debug.Log("Placing monster...");
        }
```

```
        else if (nextHandler != null)
        {
            nextHandler.HandleRequest(action, placeMonster, data);
        }
    }
}


public class UpgradeHandler : ActionHandler
{
    public override void HandleRequest(string action, PlaceMonster placeMonster, PlaceMonsterData
data)
    {
        if (action == "upgrade")
        {
            var monsterAction = new UpgradeMonsterAction(placeMonster.monster,
PlaceMonster.gameManager, data.timestamp);
            monsterAction.TemplateMethod(); // Execute upgrade action
            Debug.Log("Upgrading monster...");
        }
        else if (nextHandler != null)
        {
            nextHandler.HandleRequest(action, placeMonster, data);
        }
    }
}
private ActionHandler BuildActionHandlerChain()
    {
        // Instantiate handlers
        ActionHandler publishPlaceHandler = new PublishPlaceHandler();
        ActionHandler publishUpgradeHandler = new PublishUpgradeHandler();
        ActionHandler placeHandler = new PlaceHandler();
        ActionHandler upgradeHandler = new UpgradeHandler();

        // Link the handlers
        publishPlaceHandler.SetNext(publishUpgradeHandler);
        publishUpgradeHandler.SetNext(placeHandler);
        placeHandler.SetNext(upgradeHandler);

        return publishPlaceHandler; // Return the first handler in the chain
    }
}
```

2.2.4 State




```
public interface IGameState
{
    5 references
    void Enter();
    5 references
    void UpdateState();
    5 references
    void Exit();
}
```

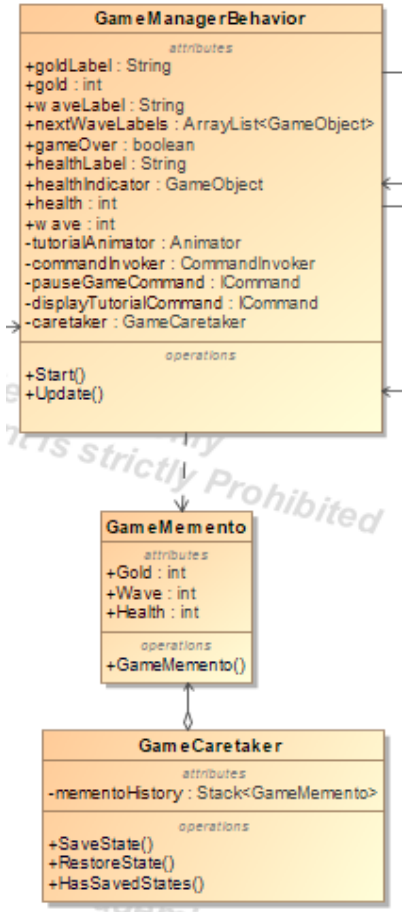
```
2 references
public class GameStateManager
{
    private IGameState currentState;

    4 references
    public void SetState(IGameState newState)
    {
        currentState?.Exit();
        currentState = newState;
        currentState?.Enter();
    }

    1 reference
    public void UpdateCurrentState()
    {
        currentState?.UpdateState();
    }
}
```

```
using UnityEngine;

2 references
public class WaitingForPlayersState : IGameState
{
    private GameManagerBehavior gameManager;
    private AblyManagerBehavior ablyManager;

    1 reference
    public WaitingForPlayersState(GameManagerBehavior gm)
    {
        this.gameManager = gm;
        this.ablyManager = GameObject.Find("AblyManager").GetComponent<AblyManagerBehavior>();
    }

    2 references
    public void Enter()
    {
        ablyManager.started = false;
    }

    2 references
    public void UpdateState()
    {
        if (ablyManager.started && ablyManager.startTimeAbly != null)
        {
            gameManager.StartGameRunning();
        }
    }

    2 references
    public void Exit()
    {
    }
}
```

```
using UnityEngine;

2 references
public class MenuState : IGameState
{
    private GameManagerBehavior gameManager;

    1 reference
    public MenuState(GameManagerBehavior gm)
    {
        this.gameManager = gm;
    }

    2 references
    public void Enter()
    {
    }

    2 references
    public void UpdateState()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            gameManager.StartWaitingForPlayers();
        }
    }

    2 references
    public void Exit()
    {

    }
}
```

```
using UnityEngine;

2 references
public class GameOverState : IGameState
{
    private GameManagerBehavior gameManager;

    1 reference
    public GameOverState(GameManagerBehavior gm)
    {
        this.gameManager = gm;
    }

    2 references
    public void Enter()
    {
    }

    2 references
    public void UpdateState()
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            gameManager.StartGameRunning();
        }
    }

    2 references
    public void Exit()
    {
    }
}
public class GameRunningState : IGameState
{
    private GameManagerBehavior gameManager;

    1 reference
    public GameRunningState(GameManagerBehavior gm)
    {
        this.gameManager = gm;
    }

    2 references
    public void Enter()
    {
    }

    2 references
    public void UpdateState()
    {
        if (gameManager.Health <= 0)
        {
            gameManager.GameOver();
        }
    }

    2 references
    public void Exit()
    {
    }
}
```

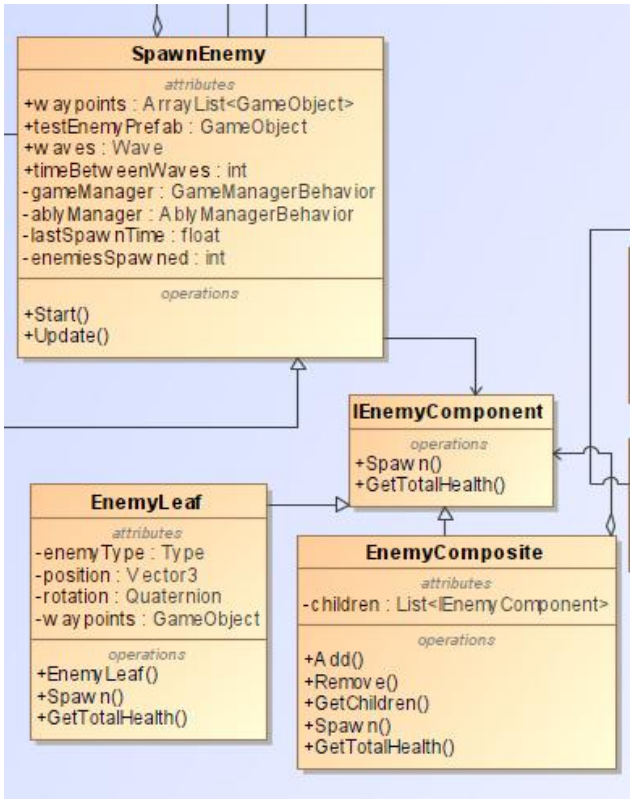
2.2.5 Memento




```
public class GameMemento
{
    2 references
    public int Gold { get; private set; }
    2 references
    public int Wave { get; private set; }
    2 references
    public int Health { get; private set; }

    1 reference
    public GameMemento(int gold, int wave, int health)
    {
        Gold = gold;
        Wave = wave;
        Health = health;
    }
}
```

```
using System.Collections.Generic;

2 references
public class GameCaretaker
{
    private Stack<GameMemento> mementoHistory = new Stack<GameMemento>();

    1 reference
    public void SaveState(GameMemento memento)
    {
        mementoHistory.Push(memento);
    }

    1 reference
    public GameMemento RestoreState()
    {
        if (mementoHistory.Count > 0)
        {
            return mementoHistory.Pop();
        }
        return null;
    }

    1 reference
    public bool HasSavedStates()
    {
        return mementoHistory.Count > 0;
    }
}
```

2.2.6 Composite



```
using System.Collections.Generic;

5 references
public class EnemyComposite : IEnemyComponent
{
    private List<IEnemyComponent> children = new List<IEnemyComponent>();

    1 reference
    public void Add(IEnemyComponent component)
    {
        children.Add(component);
    }

    0 references
    public void Remove(IEnemyComponent component)
    {
        children.Remove(component);
    }

    1 reference
    public List<IEnemyComponent> GetChildren()
    {
        return children;
    }

    2 references
    public void Spawn()
    {
        foreach (var child in children)
        {
            child.Spawn();
        }
    }

    2 references
    public int GetTotalHealth()
    {
        int total = 0;
        foreach (var child in children)
        {
            total += child.GetTotalHealth();
        }
        return total;
    }
}
```

```
public interface IEnemyComponent
{
    4 references
    void Spawn();
    3 references
    int GetTotalHealth();
}
```

```
using UnityEngine;

7 references
public class EnemyLeaf : IEnemyComponent
{
    private System.Type enemyType;
    private Vector3 position;
    private Quaternion rotation;
    private GameObject[] waypoints;

    1 reference
    public EnemyLeaf(System.Type enemyType, Vector3 position, Quaternion rotation, GameObject[] waypoints)
    {
        this.enemyType = enemyType;
        this.position = position;
        this.rotation = rotation;
        this.waypoints = waypoints;
    }

    3 references
    public void Spawn()
    {
        EnemyFactory.CreateEnemy(enemyType, position, rotation, waypoints);
    }

    2 references
    public int GetTotalHealth()
    {
        return 100;
    }
}
```