Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

Высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель С. М. Старолетов\_

(подпись) (и. о.,фамилия)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

дата

Комплексный отчет

по дисциплине

«Архитектурное проектирование и паттерны программирования»

ЛР 09.03.04.9.000О

Студент группы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ПИ-42\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С.А. Киреков

(и. о., фамилия)

Преподаватель\_\_\_\_\_\_\_доцент, к-т. ф-м. н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С. М. Старолетов

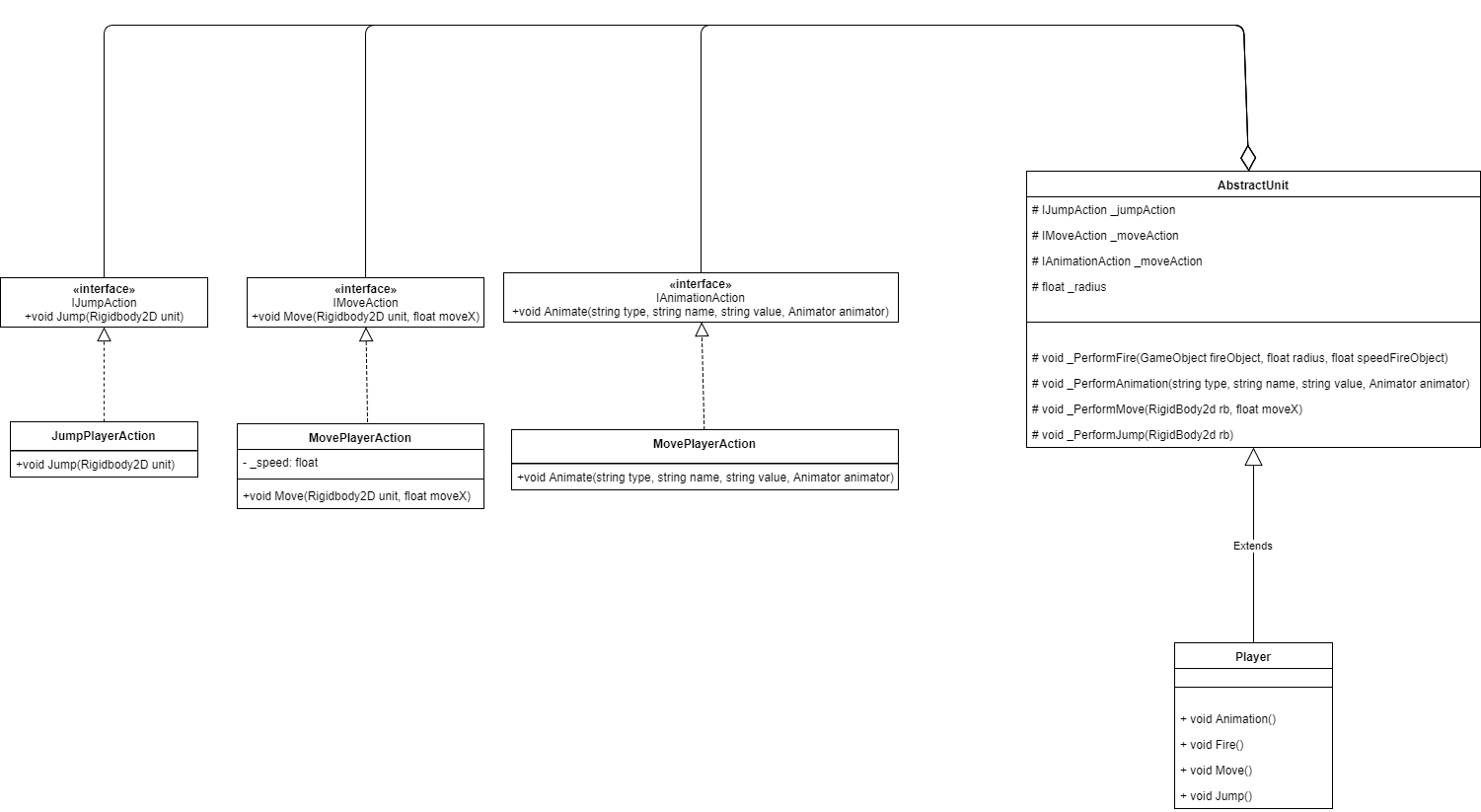
должность, ученое звание (и. о., фамилия)

Барнаул 2017

В качестве проекта для реализации паттернов проектирования была выбрана разработка игры в жанре «2D-платформер» на базе движка Unity-3D. Язык разработки – C#. Исходный код программы прилагается на диске вместе с отчетом.

**Делегирование, GRASP.**

Диаграмма классов:

****

Код:

Player.cs

using System.Collections;

using System.Collections.Generic;

using System;

using UnityEngine;

public class Player : AbstractUnit

{

private Rigidbody2D \_rigidBody;

private Animator \_anim; // контроллер анимации

private string \_currentFacing; // в какую сторону смотрит персонаж

public GameObject shotObject; // объект снаряда, которым стреляем

Player()

{

\_jumpAction = new JumpAction(); // делегат, реализующий прыжок

\_moveAction = new MoveAction(); // делегат, реализующий движение персонажа

\_animationAction = new AnimationAction(); // делегат, реализующий анимацию персонажа

}

private void Start()

{

\_rigidBody = GetComponent<Rigidbody2D>();

\_anim = GetComponent<Animator>();

\_currentFacing = "right";

}

// отзеркалить по оси y спрайт персонажа

private void FlipSprite()

{

var vector = transform.localScale;

vector.x \*= -1;

transform.localScale = vector;

switch (\_currentFacing)

{

case "right":

\_currentFacing = "left";

break;

case "left":

\_currentFacing = "right";

break;

default:

\_currentFacing = "right";

Debug.Log("Ошибка поворота персонажа");

break;

}

}

void Update()

{

Jump(); // если нажали пробел, прыгаем

Fire();

}

// физика персонажа

private void FixedUpdate()

{

Move(); // если нажали кнопки движения, двигаемся

}

// движение персонажа

private void Move()

{

float moveSpeed = Input.GetAxis("Horizontal");

\_PerformMove(\_rigidBody, moveSpeed);

// в зависимости от направления движения разворачиваем спрайт

if (moveSpeed > 0 && \_currentFacing != "right" ||

moveSpeed < 0 && \_currentFacing != "left")

{

FlipSprite();

}

// анимация движения

Animation("float", "Speed", Mathf.Abs(moveSpeed).ToString());

}

private void Fire()

{

// если нажали левую кнопку мыши

if (Input.GetMouseButtonDown(0))

{

float radius = 1f;

// если персонаж смотрит влево, умножим радиус на -1, чтобы снаряд появился слева от него

if (\_currentFacing == "left")

{

radius \*= -1;

}

\_PerformFire(shotObject, radius);

}

}

// прыжок

private void Jump()

{

if (Input.GetKeyDown(KeyCode.Space))

{

\_PerformJump(\_rigidBody);

}

Animation("float", "Vertical Speed", \_rigidBody.velocity.y.ToString());

}

// анимация персонажа

private void Animation(string type, string name, string value)

{

try

{

\_PerformAnimation(type, name, value, \_anim);

}

catch (Exception e)

{

Debug.Log(e.Message.ToString());

}

}

}

AbstractUnit.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AbstractUnit : MonoBehaviour

{

protected IJumpAction \_jumpAction;

protected IMoveAction \_moveAction;

protected IAnimationAction \_animationAction;

protected float \_radius;

// прыжок

protected void \_PerformJump(Rigidbody2D rigidBody)

{

\_jumpAction.Jump(rigidBody);

}

// движение

protected void \_PerformMove(Rigidbody2D rigidBody, float moveX)

{

\_moveAction.Move(rigidBody, moveX);

}

// анимация

protected void \_PerformAnimation(string type, string name, string value, Animator animator)

{

\_animationAction.Animate(type, name, value, animator);

}

// огонь

protected void \_PerformFire(GameObject fireObject, float radius, float speedFireObject = 25f)

{

// высчитываем позицию, где должен появиться снаряд

var position = transform.position;

position.x += radius;

// обозначаем направление, куда должен двигаться объект при выстреле

int direction = radius >= 0 ? 1 : -1;

// инстанцируем объект на сцену и задаем ему вектор движения

GameObject fireObjInst = Instantiate(fireObject, position, transform.rotation, transform.parent) as GameObject;

fireObjInst.GetComponent<Rigidbody2D>().AddForce(new Vector2(speedFireObject \* direction, 0));

// если за 10 секунд снаряд не долетел цели, уничтожаем его

Destroy(fireObjInst, 10f);

}

}

IJumpAction.cs

// интерфейс прыжка

public interface IJumpAction

{

void Jump(Rigidbody2D rigidBody);

}

IMoveAction.cs

// интерфейс движения

public interface IMoveAction

{

void Move(Rigidbody2D rigidBody, float moveX);

}

IAnimationAction.cs

// интерфейс анимации

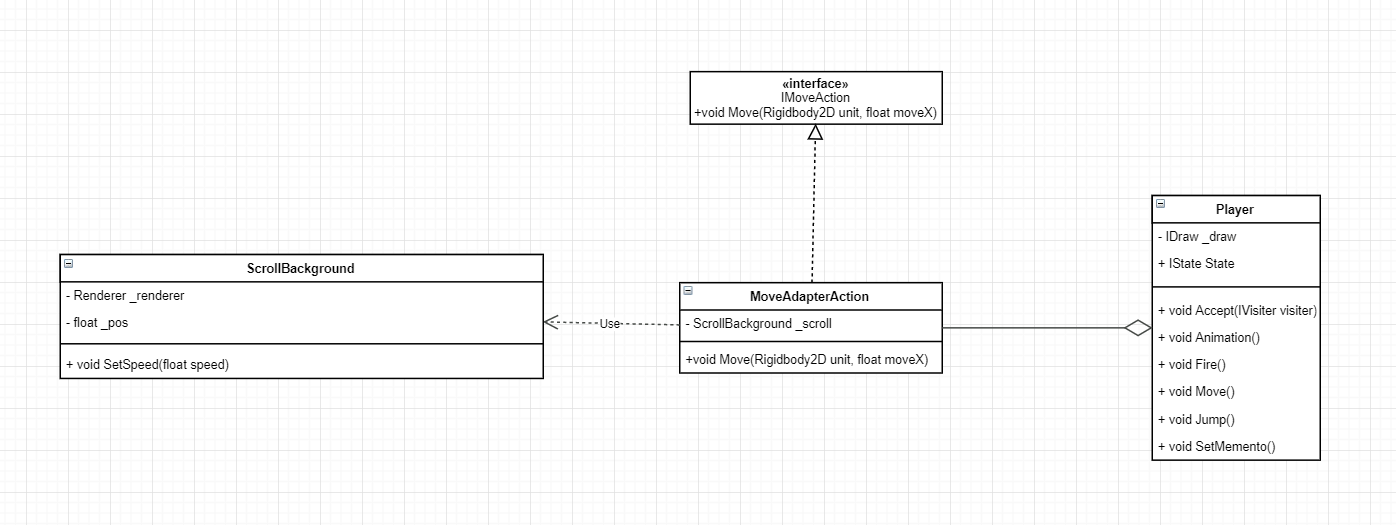
public interface IAnimationAction

{

void Animate(string type, string name, string value, Animator animator);

}

**Адаптер**



Player:

// движение персонажа

private void Move()

{

float moveSpeed = Input.GetAxis("Horizontal");

\_PerformMove(\_rigidBody, moveSpeed \* \_speedCoef);

\_adapter.Move(\_rigidBody, moveSpeed / 500);

// в зависимости от направления движения разворачиваем спрайт

if (moveSpeed > 0 && \_currentFacing != "right" ||

moveSpeed < 0 && \_currentFacing != "left")

{

FlipSprite();

}

// анимация движения

Animation("float", "Speed", Mathf.Abs(moveSpeed).ToString());

}

MoveAdapterAction:

public class MoveAdapterAction : MonoBehaviour, IMoveAction

{

public ScrollBackground \_scroll = null;

public void Move(Rigidbody2D rigidbody, float moveX)

{

if (\_scroll != null)

\_scroll.SetSpeed(moveX);

}

}

ScrollBackground:

public void SetSpeed(float speed)

{

\_pos += speed;

if (\_pos > 1)

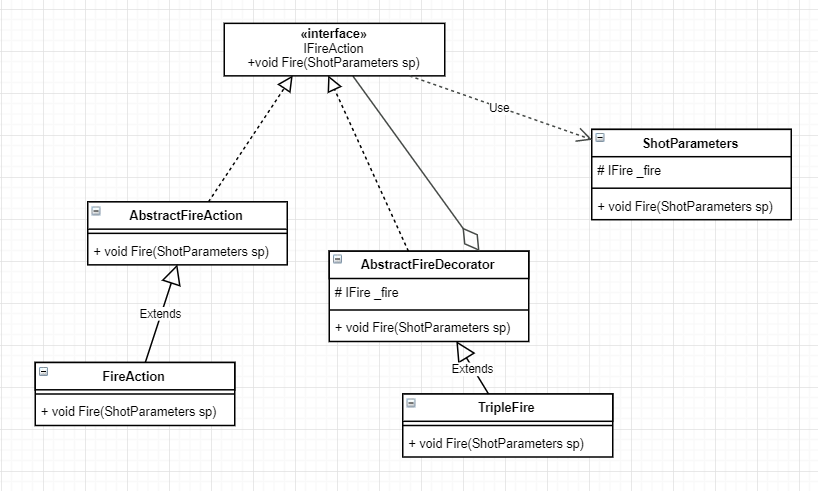
{

\_pos -= 1;

}

\_renderer.material.mainTextureOffset = new Vector2(\_pos, 0);}

**Декоратор**



ShotParameters:

public class ShotParameters

{

public GameObject FireObject { private set; get; } // снаряд

public Vector2 Direction { private set; get; } // Вектор направления стр.

public Vector3 Position { private set; get; } // точка, откуда стреляем

public float Speed { private set; get; }

public ShotParameters(GameObject fireObject, Vector2 direction,

Vector3 position, float speed)

{

FireObject = fireObject;

Direction = direction.normalized;

Position = position;

Speed = speed;

}

}

FireAction:

public override void Fire(ShotParameters sp)

{

// инстанцируем объект на сцену и задаем ему вектор движения

GameObject fireObjInst = Instantiate(sp.FireObject, sp.Position, Quaternion.Euler(0, 0, 0));

Rigidbody2D rb;

try

{

rb = fireObjInst.GetComponent<Rigidbody2D>();

}

catch (NullReferenceException)

{

Debug.Log("Ошибка в FireAction->Fire(ShorParameters sp).\n" +

"sp.FireObject должен содержать компонент RigidBody2D");

Destroy(fireObjInst);

return;

}

rb.AddForce(sp.Direction \* sp.Speed);

// если за 10 секунд снаряд не долетел цели, уничтожаем его

Destroy(fireObjInst, 10f);

}

AbstractFireDecorator:

public abstract class AbstractFireDecorator : IFireAction

{

protected IFireAction \_fire;

public abstract void Fire(ShotParameters sp);

}

TripleFire:

public override void Fire(ShotParameters sp)

{

// рассчитаем ширину объекта, которым стреляем

float width = sp.FireObject.GetComponent<SpriteRenderer>().size.x;

// угол между направлением движения объекта и осью X

float angle = Mathf.Atan2(sp.Direction.y, sp.Direction.x);

// теперь будем стрелять одновременно тремя зарядами так, чтобы они были на равном расстоянии друг от друга

Vector2 pos1 = sp.Position;

Vector2 pos2 = new Vector2(

(float)(sp.Position.x + Mathf.Cos(angle) \* width \* 1.5),

(float)(sp.Position.y + Math.Sin(angle) \* width \* 1.5)

);

Vector2 pos3 = new Vector2(

(float)(pos2.x + Mathf.Cos(angle) \* width \* 1.5),

(float)(pos2.y + Math.Sin(angle) \* width \* 1.5)

);

ShotParameters sp2 = new ShotParameters(sp.FireObject, sp.Direction, pos2, sp.Speed);

ShotParameters sp3 = new ShotParameters(sp.FireObject, sp.Direction, pos3, sp.Speed);

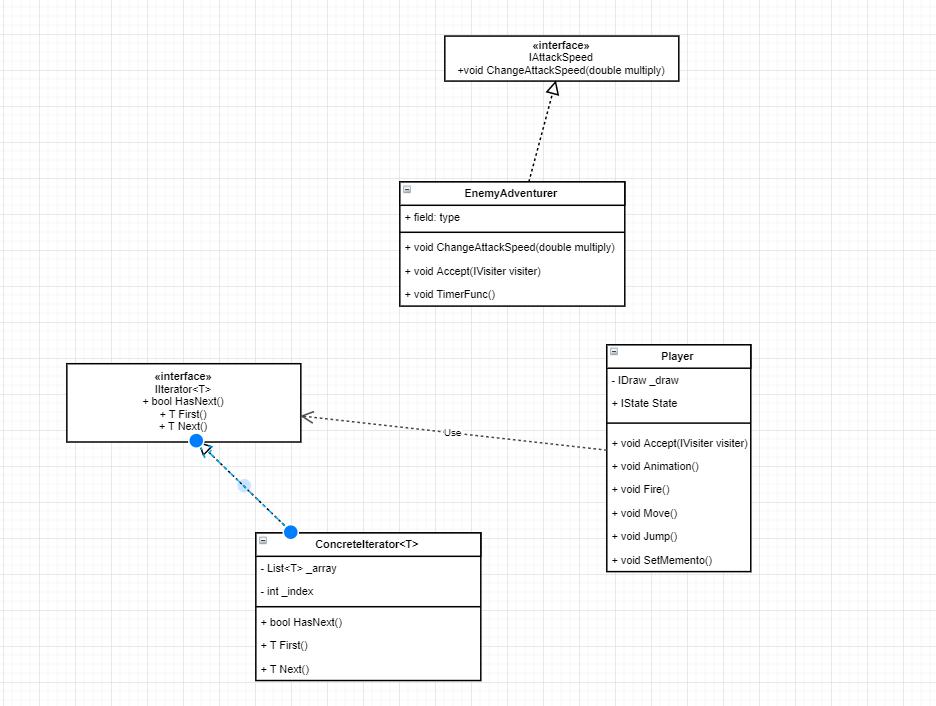
\_fire.Fire(sp);

\_fire.Fire(sp2);

\_fire.Fire(sp3);

}

**Итератор**



ConcreteIterator:

public class ConcreteIterator<T> : IIterator<T>

{

private IList<T> \_array;

private int \_index;

public ConcreteIterator(IList<T> array)

{

\_array = array;

\_index = 0;

}

public bool HasNext()

{

return \_index < \_array.Count;

}

public T Next()

{

return \_array[\_index++];

}

public T First()

{

\_index = 0;

return \_array[\_index];

}

}

Player:

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.gameObject.tag == "DecreaseAttackSpeed")

{

Destroy(collision.gameObject);

List<IAttackSpeed> list = new List<IAttackSpeed>(rootEnemies.GetComponentsInChildren<IAttackSpeed>());

IIterator<IAttackSpeed> iterator = new ConcreteIterator<IAttackSpeed>(list);

while (iterator.HasNext())

{

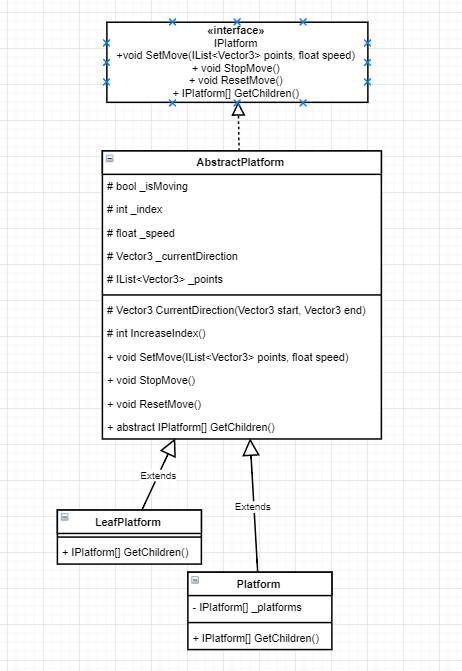
iterator.Next().ChangeAttackSpeed(3);

}

}

}

**Композит**



AbstractPlatform:

public abstract class AbstractPlatform : MonoBehaviour, IPlatform

{

protected bool \_isMoving = false;

protected IList<Vector3> \_points = null;

protected float \_speed;

protected int \_index;

protected Vector3 \_currentDirection;

protected Vector3 CalculateDirection(Vector3 start, Vector3 end)

{

return (end - start).normalized;

}

protected void IncreaseIndex()

{

\_index++;

if (\_index >= \_points.Count)

{

\_index = 0;

}

}

// скорость движения платформы

public void SetMove(IList<Vector3> points, float speed)

{

\_isMoving = true;

\_points = points;

\_speed = speed;

\_index = 0;

\_currentDirection = CalculateDirection(transform.position, points[0]);

}

// приостановить движение

public void StopMove()

{

\_isMoving = false;

}

// возобновить движение

public void ResetMove()

{

if (\_points != null)

{

\_isMoving = true;

}

else

{

return;

}

}

public abstract IPlatform[] GetChildren();

protected void Update()

{

if (!\_isMoving)

{

return;

}

transform.Translate(\_currentDirection \* \_speed \* Time.deltaTime, Space.World);

if ((transform.position - \_points[\_index]).magnitude < 0.1)

{

Vector3 start = \_points[\_index];

IncreaseIndex();

\_currentDirection = CalculateDirection(start, \_points[\_index]);

}

}

}

Platform:

public class Platform : AbstractPlatform

{

private IPlatform[] \_children;

private void Start()

{

\_children = GetComponentsInChildren<IPlatform>();

}

public override IPlatform[] GetChildren()

{

return \_children;

}

}

LeafPlatform:

public class LeafPlatform : AbstractPlatform

{

public override IPlatform[] GetChildren()

{

throw new NotImplementedException();

}

}

**Мост**



ConsoleDraw:

public class ConsoleDraw : AbstractDraw, IDraw

{

public ConsoleDraw(GameObject obj)

{

player = obj;

sr = player.GetComponent<SpriteRenderer>();

oldColor = sr.color;

newColor = new Color(oldColor.r, oldColor.g, oldColor.b, 0);

}

public void DrawJump()

{

Reset();

DebugConsole.Log("Игрок прыгает");

}

public void DrawFire()

{

Reset();

DebugConsole.Log("Игрок стреляет");

}

public void DrawMove(float speed)

{

if (Mathf.Abs(speed) < 0.01)

return;

Reset();

string direction = (speed > 0) ? "вправо" : "влево";

DebugConsole.Log("Игрок двигается " + direction);

}

public override void Reset()

{

sr.color = newColor;

DebugConsole.Clear();

}

}

RegularDraw:

public class RegularDraw : AbstractDraw, IDraw

{

public RegularDraw(GameObject player)

{

this.player = player;

sr = player.GetComponent<SpriteRenderer>();

oldColor = sr.color;

newColor = new Color(oldColor.r, oldColor.g, oldColor.b, 1);

}

public void DrawJump()

{

Reset();

}

public void DrawFire()

{

Reset();

}

public void DrawMove(float speed)

{

Reset();

}

public override void Reset()

{

sr.color = newColor;

}

}

Player:

IDraw GetDraw(bool isConsoleDraw)

{

if (isConsoleDraw)

return \_drawConsole;

else

return \_basicDraw;

}

void Update()

{

if (Input.GetKeyDown(KeyCode.U))

{

isConsoleDraw = !isConsoleDraw;

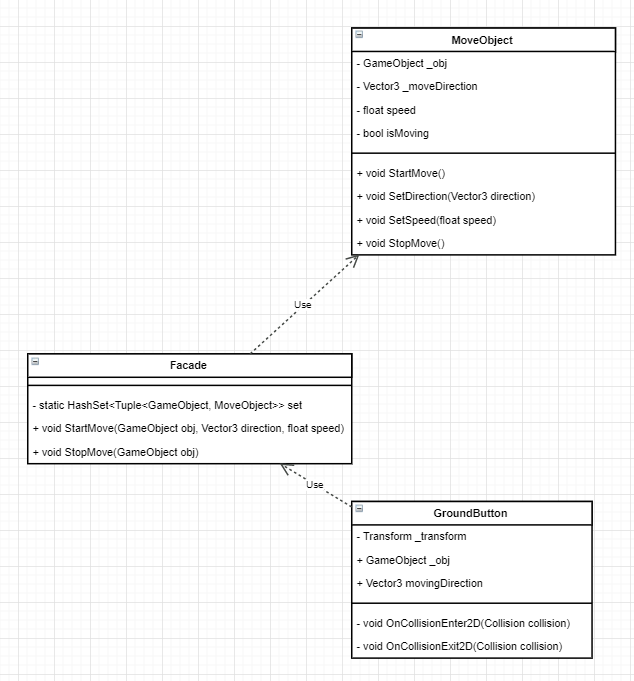
}

Jump(); // если нажали пробел, прыгаем

Fire();

}

**Фасад**



GroundButton:

public class GroundButton : MonoBehaviour

{

private Transform \_transform;

public GameObject movingPlatform;

public Vector3 moveDirection;

void Start()

{

\_transform = transform;

}

void Update()

{

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag != "Player")

return;

Facade.StartMove(movingPlatform, moveDirection, 1);

}

private void OnCollisionExit2D(Collision2D collision)

{

if (collision.gameObject.tag != "Player")

return;

Facade.StopMove(movingPlatform);

}

}

MoveObject:

public class MoveObject : MonoBehaviour

{

public GameObject \_obj;

public Vector3 \_moveDirection;

public float \_speed;

private bool \_isMoving = false;

public MoveObject()

{

}

public MoveObject(GameObject obj, Vector3 moveDirection, float speed)

{

SetDirection(moveDirection, speed);

\_obj = obj;

\_isMoving = false;

}

public void SetDirection(Vector3 direction, float speed)

{

\_moveDirection = direction.normalized;

SetSpeed(speed);

}

public void SetSpeed(float speed)

{

\_speed = speed;

}

public void StartMove()

{

\_isMoving = true;

}

public void StopMove()

{

\_isMoving = false;

}

private void Update()

{

if (!\_isMoving)

return;

\_obj.transform.Translate(\_moveDirection \* \_speed \* Time.deltaTime, Space.World);

}

}

Facade:

public class Facade

{

static HashSet<Tuple<GameObject, MoveObject>> set = new HashSet<Tuple<GameObject, MoveObject>>();

public static void StartMove(GameObject obj, Vector3 direction, float speed)

{

GameObject gm = new GameObject();

var move = gm.AddComponent<MoveObject>();

move.\_obj = obj;

move.\_moveDirection = direction.normalized;

move.\_speed = speed;

move.StartMove();

move.StartMove();

set.Add(new Tuple<GameObject, MoveObject>(obj, move));

}

public static void StopMove(GameObject obj)

{

Tuple<GameObject, MoveObject> removeItem = null;

foreach (var item in set)

{

if (item.Item1 == obj)

{

removeItem = item;

removeItem.Item2.StopMove();

break;

}

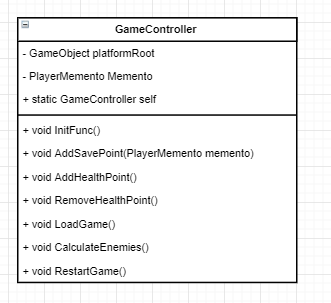
}

set.Remove(removeItem);

}

}

**Информационный эксперт**



GameController:

public class GameController : MonoBehaviour

{

public GameObject platformRoot;

public GameObject root;

// Use this for initialization

void Start()

{

InitPlatforms();

}

private void InitPlatforms()

{

var platforms = platformRoot.GetComponentsInChildren<AbstractPlatform>();

foreach (var platform in platforms)

{

Vector3 position = platform.transform.position;

platform.SetMove(

new Vector3[]

{

new Vector3(position.x, position.y + (Random.value + 1), position.z),

new Vector3(position.x, position.y - (Random.value + 1), position.z)

},

(Random.value + 1) \* 1

);

}

}

void Update()

{

var curTimeScale = Time.timeScale;

if (Input.GetAxis("Mouse ScrollWheel") > 0f)

{

curTimeScale += 0.05f;

}

else if (Input.GetAxis("Mouse ScrollWheel") < 0f)

{

curTimeScale -= 0.05f;

}

SetTimeScale(curTimeScale);

// подсчитать количество врагов

if (Input.GetKeyDown(KeyCode.I))

{

Debug.Log("------------");

int units = root.GetComponentsInChildren<AbstractUnit>().Length;

int ea = root.GetComponentsInChildren<EnemyAdventurer>().Length;

int es = root.GetComponentsInChildren<EnemySoldier>().Length;

Debug.Log("Количество юнитов (вместе с игроком): " + units);

Debug.Log("Количество врагов EnemyAdventurer: " + ea);

Debug.Log("Количество врагов EnemySoldier: " + es);

Debug.Log("------------");

}

}

void SetTimeScale(float timescale)

{

if (timescale < 0)

{

timescale = 0;

}

if (Mathf.Abs(timescale - Time.timeScale) > 0.01f)

{

DebugConsole.Clear();

DebugConsole.Log(timescale.ToString());

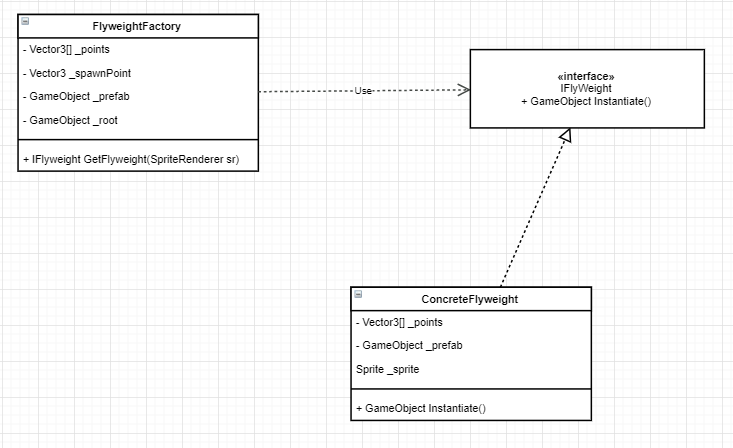
}

Time.timeScale = timescale;

}

}

**Приспособленец**



FlyweightFactory:

public class FlyweightFactory : MonoBehaviour

{

private Vector3[] \_points;

private Vector3 \_spawnPoint;

private GameObject \_prefab;

private GameObject \_root;

public FlyweightFactory(Vector3[] points, Vector3 spawnPoint, GameObject prefab, GameObject root)

{

\_points = points;

\_spawnPoint = spawnPoint;

\_prefab = prefab;

\_root = root;

}

public IFlyweight GetFlyweight(SpriteRenderer sprite)

{

return new ConcreteFlyweight(sprite, \_points, Instantiate(\_prefab,\_spawnPoint, Quaternion.EulerAngles(0, 0, 0), \_root.transform));

}

}

ConcreteFlyweight:

public class ConcreteFlyweight : IFlyweight

{

Vector3[] \_points;

GameObject \_prefab;

Sprite \_sprite;

public ConcreteFlyweight(SpriteRenderer sprite, Vector3[] points, GameObject prefab)

{

\_points = points;

\_prefab = prefab;

\_sprite = sprite.sprite;

}

public GameObject Instantiate()

{

\_prefab.GetComponent<SpriteRenderer>().sprite = \_sprite;

\_prefab.AddComponent<FlyweightGameObject>();

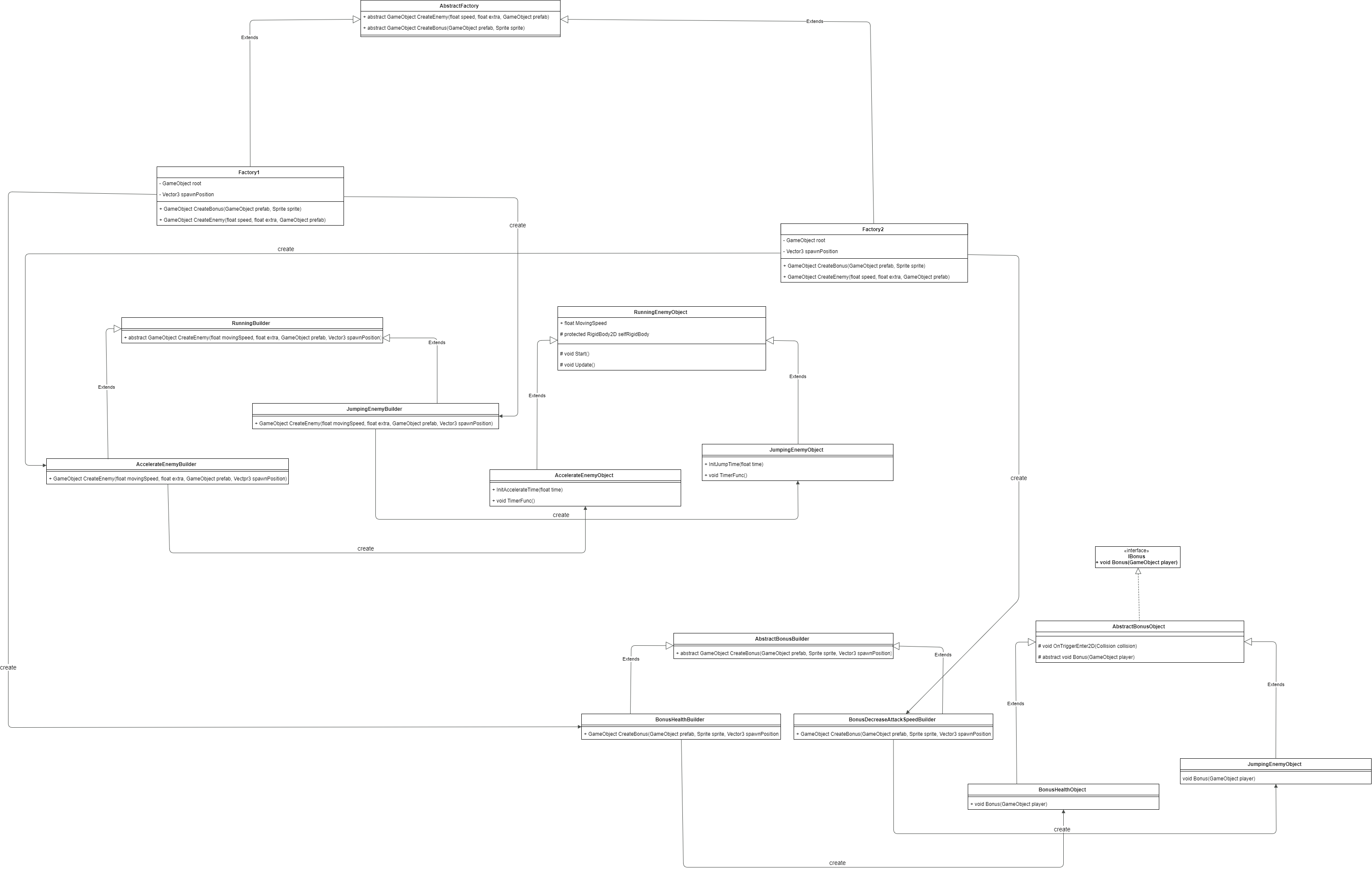
\_prefab.GetComponent<FlyweightGameObject>().position = \_points;

return \_prefab;

}

}

**Абстрактная фабрика, фабричный метод и строитель**



Factory1:

public class Factory1 : AbstractFactory

{

GameObject root;

Vector3 spawnPosition;

public Factory1(GameObject root, Vector3 spawnPosition)

{

this.root = root;

this.spawnPosition = spawnPosition;

}

public override GameObject CreateEnemy(float speed, float extra, GameObject prefab)

{

var jump = new JumpingEnemyBuilder();

return jump.CreateEnemy(speed, extra, prefab, spawnPosition);

}

public override GameObject CreateBonus(GameObject prefab, Sprite sprite)

{

var bonus = new BonusHealthBuilder();

return bonus.CreateBonus(prefab, sprite, spawnPosition);

}

}

Factory2:

public class Factory2 : AbstractFactory

{

private GameObject root;

private Vector3 spawnPosition;

public Factory2(GameObject root, Vector3 spawnPosition)

{

this.root = root;

this.spawnPosition = spawnPosition;

}

public override GameObject CreateBonus(GameObject prefab, Sprite sprite)

{

var bonus = new BonusDecreaseAttackBuilder();

return bonus.CreateBonus(prefab, sprite, spawnPosition);

}

public override GameObject CreateEnemy(float speed, float extra, GameObject prefab)

{

var accelerate = new AccelerationEnemyBuilder();

return accelerate.CreateEnemy(speed, extra, prefab, spawnPosition);

}

}

AccelerationEnemyBuilder:

public class AccelerationEnemyBuilder : RunningEnemyBuilder

{

public override GameObject CreateEnemy(float movingSpeed, float extra, GameObject prefab, Vector3 spawnPosition)

{

prefab.GetComponent<Transform>().position = spawnPosition;

AccelerationEnemyObject enemy = prefab.AddComponent<AccelerationEnemyObject>();

enemy.MovingSpeed = movingSpeed;

enemy.InitAccelerateTime(extra);

return prefab;

}

}

JumpingEnemyBuilder:

public class JumpingEnemyBuilder : RunningEnemyBuilder

{

public override GameObject CreateEnemy(float movingSpeed, float extra, GameObject prefab, Vector3 spawnPosition)

{

prefab.GetComponent<Transform>().position = spawnPosition;

JumpingEnemyObject enemy = prefab.AddComponent<JumpingEnemyObject>();

enemy.MovingSpeed = movingSpeed;

enemy.InitJumpTime(extra);

return prefab;

}

}

BonusHealthBuilder:

public class BonusHealthBuilder : AbstractBonusBuilder

{

public override GameObject CreateBonus(GameObject prefab, Sprite sprite, Vector3 spawnPosition)

{

var bonus = prefab.AddComponent<BonusHealthObject>();

prefab.GetComponent<SpriteRenderer>().sprite = sprite;

prefab.GetComponent<Transform>().position = spawnPosition;

return prefab;

}

}

BonusDecreaseAttackBuilder:

public class BonusDecreaseAttackBuilder : AbstractBonusBuilder

{

public override GameObject CreateBonus(GameObject prefab, Sprite sprite, Vector3 spawnPosition)

{

var bonus = prefab.AddComponent<BonusDecreaseAttackSpeedObject>();

prefab.GetComponent<SpriteRenderer>().sprite = sprite;

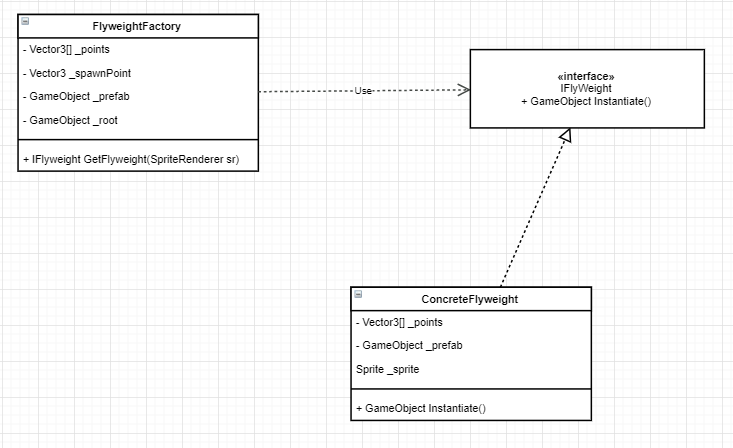
prefab.GetComponent<Transform>().position = spawnPosition;

return prefab;

}

}

**Фабрика**



public class FlyweightFactory : MonoBehaviour

{

private Vector3[] \_points;

private Vector3 \_spawnPoint;

private GameObject \_prefab;

private GameObject \_root;

public FlyweightFactory(Vector3[] points, Vector3 spawnPoint, GameObject prefab, GameObject root)

{

\_points = points;

\_spawnPoint = spawnPoint;

\_prefab = prefab;

\_root = root;

}

public IFlyweight GetFlyweight(SpriteRenderer sprite)

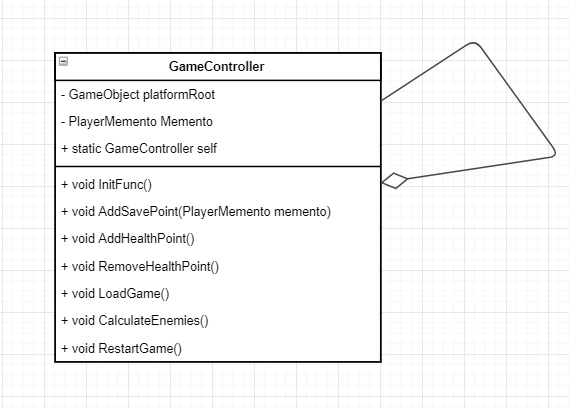
{

return new ConcreteFlyweight(sprite, \_points, Instantiate(\_prefab,\_spawnPoint, Quaternion.EulerAngles(0, 0, 0), \_root.transform));

}

}

**Синглтон**



GameController:

public static GameController self { private set; get; }

private void Awake()

{

Timer = root.GetComponentInChildren<TimerAction>();

Memento = null;

if (self != null)

{

Debug.Log("Ошибка! Может быть только один экземпляр GameController");

Destroy(gameObject);

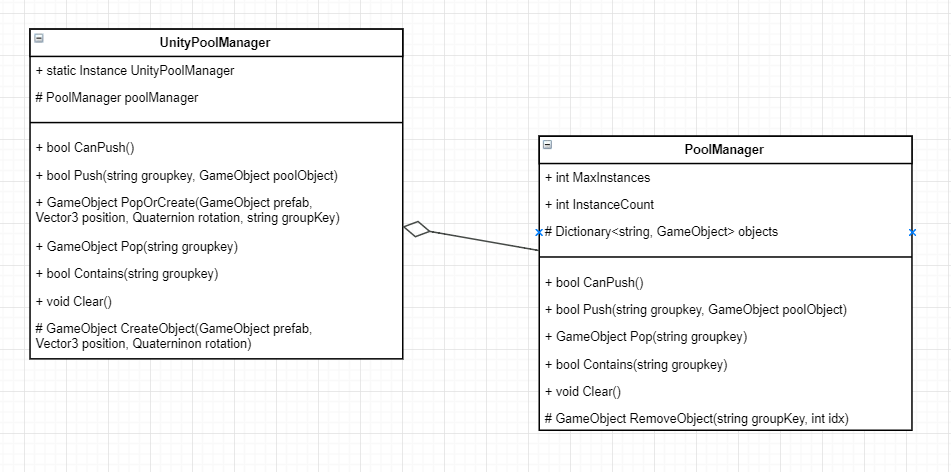
}

Mediator = new Mediator(GetComponentsInChildrenRoot<AbstractInvoker>());

self = this;

}

**Пул**



PoolManager:

public class PoolManager

{

public int MaxInstances { get; protected set; }

public int InstanceCount { get { return objects.Count; } }

protected Dictionary<string, List<GameObject>> objects;

public PoolManager(int maxInstance)

{

MaxInstances = maxInstance;

objects = new Dictionary<string, List<GameObject>>();

}

public bool CanPush()

{

return InstanceCount + 1 < MaxInstances;

}

public bool Push(string groupKey, GameObject value)

{

bool result = false;

if (CanPush())

{

value.SetActive(false);

if (!objects.ContainsKey(groupKey))

{

objects.Add(groupKey, new List<GameObject>());

}

objects[groupKey].Add(value);

}

else

{

throw new System.Exception("Ошибка в PoolManager->Push()");

}

return result;

}

public GameObject Pop(string groupKey)

{

GameObject result = null;

if (Contains(groupKey) && objects[groupKey].Count > 0)

{

result = objects[groupKey][0];

RemoveObject(groupKey, 0);

result.SetActive(true);

}

return result;

}

public bool Contains(string groupKey)

{

return objects.ContainsKey(groupKey);

}

public void Clear()

{

objects.Clear();

}

protected void RemoveObject(string groupKey, int idx)

{

if (idx >= 0 && idx < objects[groupKey].Count)

{

objects[groupKey].RemoveAt(idx);

if (objects[groupKey].Count == 0)

{

objects.Remove(groupKey);

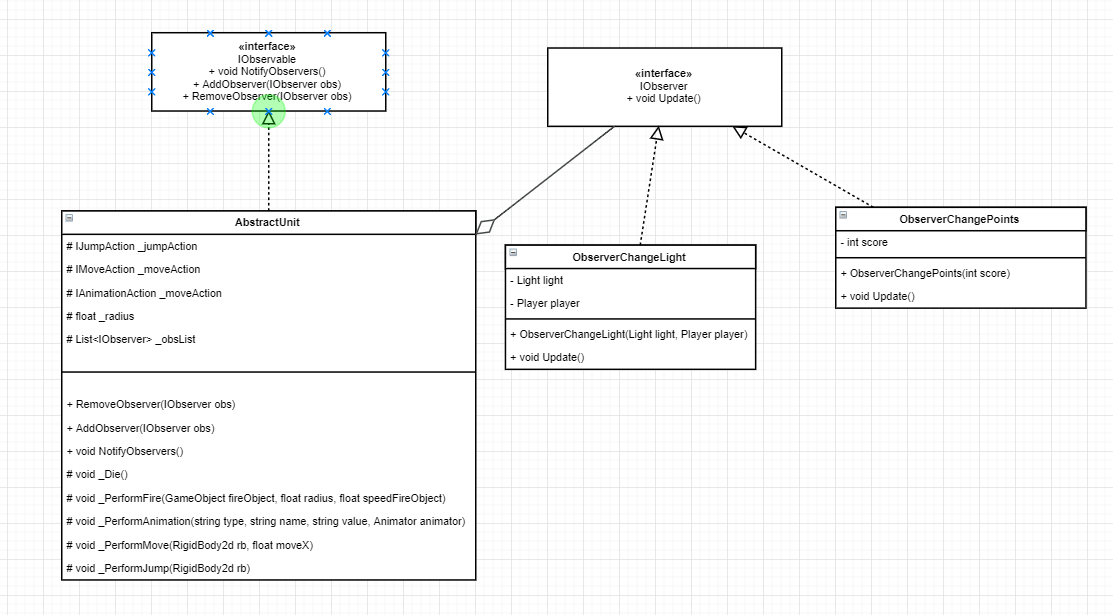
}

}

}

}

**Наблюдатель**



ObserverChangeLight:

public class ObserverChangeLight : IObserver

{

private Light light;

private Player player;

public ObserverChangeLight(Light light, Player player)

{

this.light = light;

this.player = player;

}

public void Update()

{

if (typeof(NormalState) == player.State.GetType())

{

light.color = Color.white;

}

else if (typeof(BerserkState) == player.State.GetType())

{

light.color = Color.red;

}

}

}

ObserverChangePoints:

public class ObserverChangePoints : IObserver

{

private int score;

public ObserverChangePoints(int score)

{

this.score = score;

}

public void Update()

{

GameController.self.AddScore(score);

}

}

AbstractUnit:

protected List<IObserver> \_obsList;

public void AddObserver(IObserver obs)

{

\_obsList.Add(obs);

}

public void RemoveObserver(IObserver obs)

{

\_obsList.Remove(obs);

}

public void NotifyObservers()

{

foreach (var obs in \_obsList)

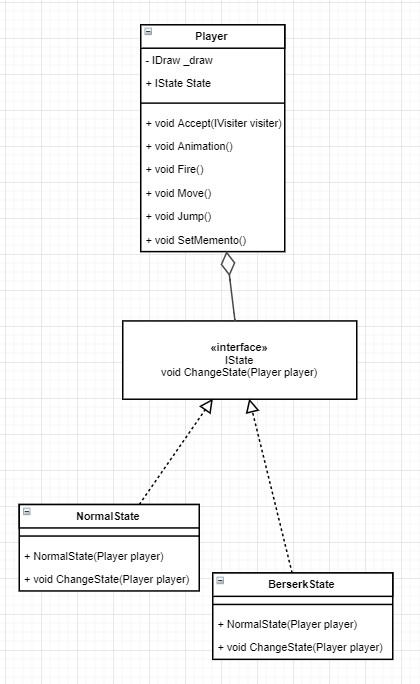
{

obs.Update();

}

}

**Состояние**



NormalState:

public class NormalState : IState

{

public NormalState(Player player)

{

player.JumpCoef = 1200;

player.SpeedCoef = 9;

}

public void ChangeState(Player player)

{

player.State = new BerserkState(player);

}

}

BerserkState:

public class BerserkState : IState

{

public BerserkState(Player player)

{

player.JumpCoef = 2000;

player.SpeedCoef = 5;

}

public void ChangeState(Player player)

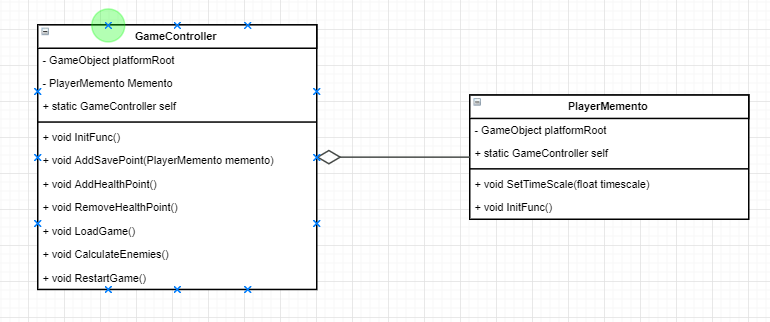
{

player.State = new NormalState(player);

}

}

**Хранитель**



PlayerMemento:

public class PlayerMemento

{

public Vector3 Position { private set; get; }

public PlayerMemento(Vector3 position)

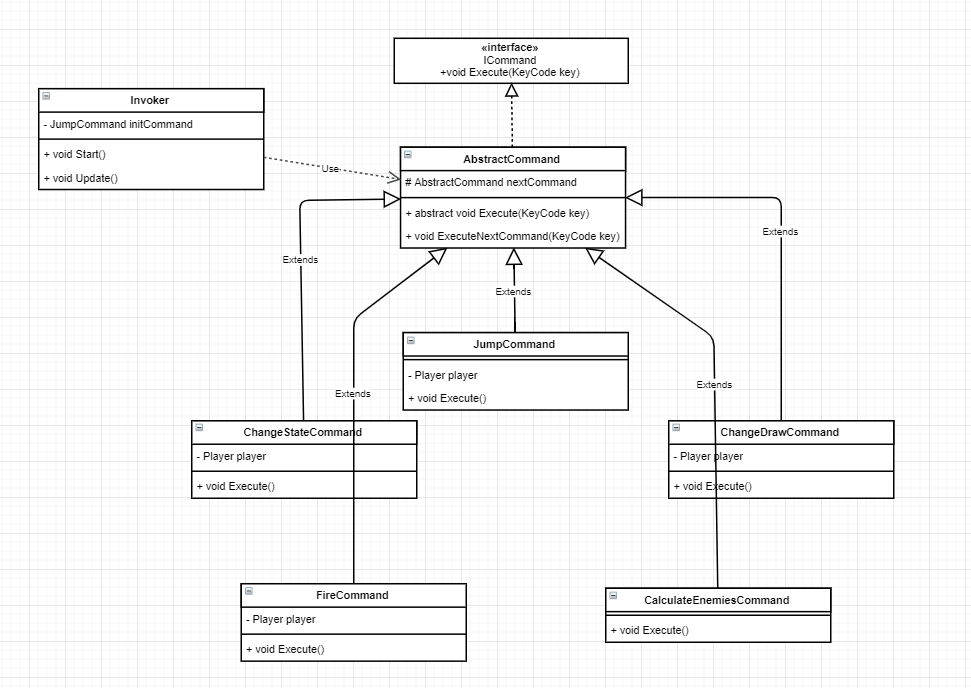
{

Position = position;

}

}

**Команда, цепочка**



Invoker:

public class Invoker : MonoBehaviour

{

private JumpCommand initCommand;

private void Start()

{

Player player = GameController.self.GetPlayer();

ChangeDrawCommand change = new ChangeDrawCommand(player, null);

ChangeStateCommand state = new ChangeStateCommand(player, change);

CalculateEnemiesCommand calculate = new CalculateEnemiesCommand(state);

FireCommand fire = new FireCommand(player, calculate);

initCommand = new JumpCommand(player, fire);

}

// Update is called once per frame

void Update()

{

foreach (KeyCode vKey in System.Enum.GetValues(typeof(KeyCode)))

{

if (Input.GetKeyDown(vKey))

{

initCommand.Execute(vKey);

}

}

}

}

AbstractCommand:

public abstract class AbstractCommand : ICommand

{

public abstract void Execute(KeyCode key);

protected AbstractCommand nextCommand;

public AbstractCommand()

{

nextCommand = null;

}

public AbstractCommand(AbstractCommand command)

{

nextCommand = command;

}

protected void ExecuteNextCommand(KeyCode key)

{

if (nextCommand != null)

{

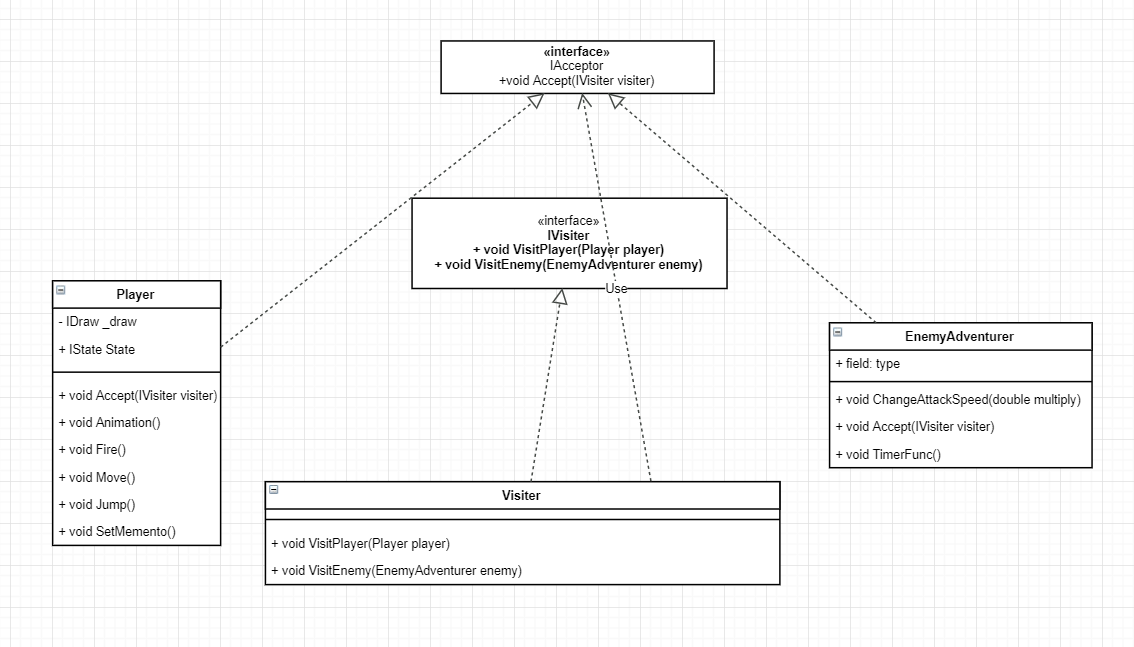
nextCommand.Execute(key);

}

}

}

**Посетитель**



Visiter:

public class Visitor : IVisitor

{

public void VisitEnemy(EnemyAdventurer unit)

{

unit.ChangeAttackSpeed(2);

}

public void VisitPlayer(Player player)

{

player.BonusHealth();

}

}

Player:

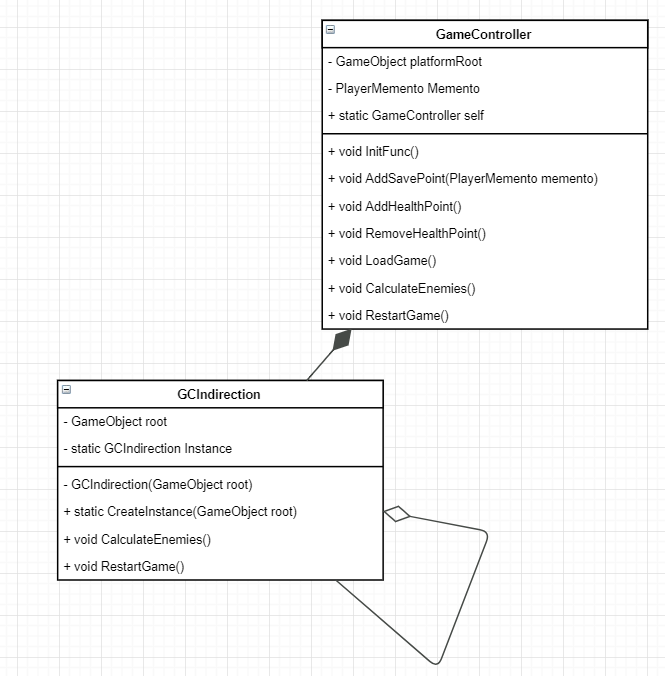
public void Accept(IVisitor visiter)

{

visiter.VisitPlayer(this);

}

**Перенаправление**



GameController:

public void RestartGame()

{

indirection.RestartGame();

}

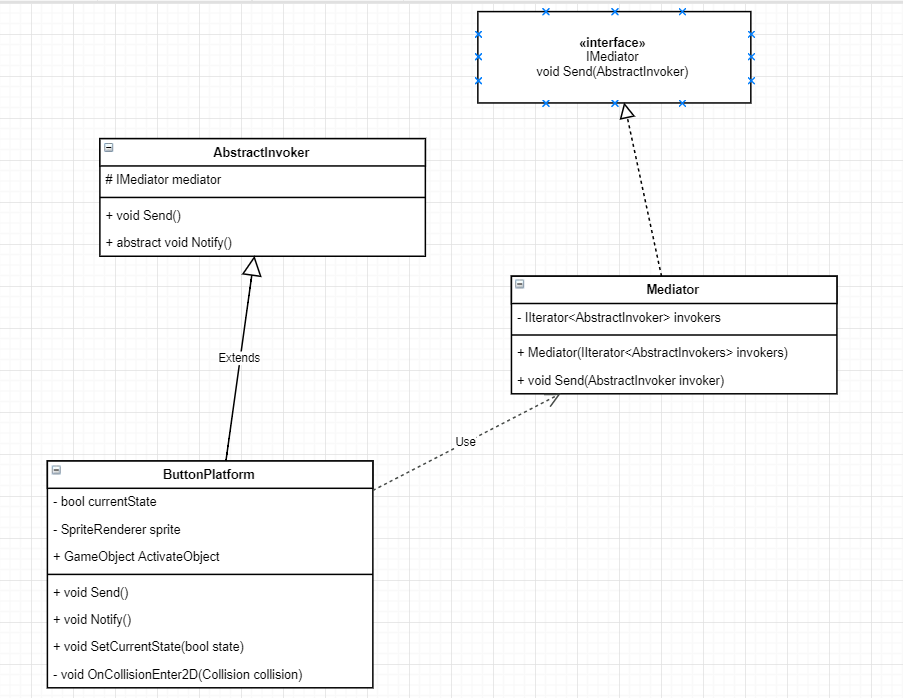
public void CalculateEnemies()

{

indirection.CalculateEnemies();

}

**Медиатор**



AbstractInvoker:

public abstract class AbstractInvoker : MonoBehaviour

{

protected IMediator mediator;

public void Send()

{

mediator.Send(this);

}

public abstract void Notify();

}

ButtonPlatfotm:

public class ButtonPlatform : AbstractInvoker

{

private bool currentState;

private SpriteRenderer sprite;

public GameObject ActivateObject;

private void Start()

{

currentState = false;

sprite = GetComponent<SpriteRenderer>();

SetCurrentState(currentState);

mediator = GameController.self.Mediator;

}

public override void Notify()

{

SetCurrentState(false);

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.GetComponent<Player>() == null)

{

return;

}

if (!currentState)

{

SetCurrentState(true);

mediator.Send(this);

}

}

private void SetCurrentState(bool state)

{

if (!state)

{

sprite.color = new Color(255, 0, 0);

ActivateObject.SetActive(false);

}

else

{

sprite.color = new Color(0, 255, 0);

ActivateObject.SetActive(true);

}

currentState = state;

}

}

Mediator:

public class Mediator : IMediator

{

private IIterator<AbstractInvoker> invokers;

public Mediator(IIterator<AbstractInvoker> invokers)

{

this.invokers = invokers;

}

public void Send(AbstractInvoker invoker)

{

invokers.First();

while (invokers.HasNext())

{

var inv = invokers.Next();

if (inv != invoker)

{

inv.Notify();

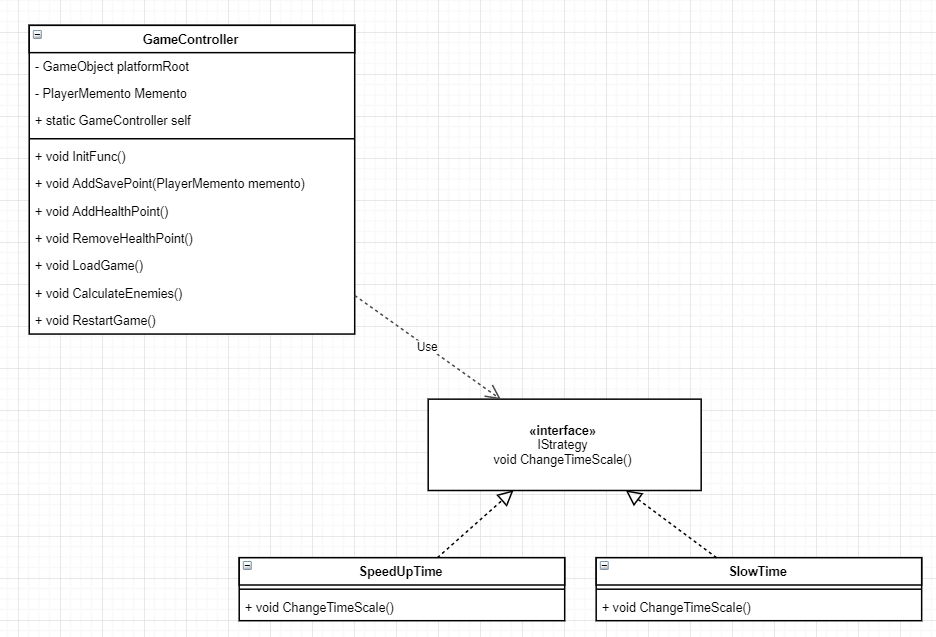
}

}

}

}

**Стратегия**



GameController:

void Update()

{

IStrategy strat = null;

if (Input.GetAxis("Mouse ScrollWheel") > 0f)

{

strat = new SpeedUpTime();

}

else if (Input.GetAxis("Mouse ScrollWheel") < 0f)

{

strat = new SlowTime();

}

if (strat != null)

{

strat.ChangeTimeScale();

}

}

SpeedUpTime:

public class SpeedUpTime : IStrategy

{

public void ChangeTimeScale()

{

float timeScale = Time.timeScale;

timeScale += 0.05f;

Time.timeScale = timeScale;

DebugConsole.Clear();

DebugConsole.Log(timeScale.ToString());

}

}

SlowTime:

public class SlowTime : IStrategy

{

public void ChangeTimeScale()

{

float timeScale = Time.timeScale;

timeScale -= 0.05f;

if (timeScale < 0)

{

timeScale = 0;

}

Time.timeScale = timeScale;

DebugConsole.Clear();

DebugConsole.Log(timeScale.ToString());

}

}