

**Московский государственный технический  
университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»  
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по ДЗ

**«Реализация модулярной системы  
базовых классов gamedeva на Godot C#.»**

Выполнил:  
студент группы ИУ5-31Б

Ларкин Б. В.

Подпись и дата:

Проверил:  
преподаватель каф. ИУ5

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2023 г.

# Цель работы

Реализовать псевдоабстрактные классы на Godot C#, то есть родительские классы для наследования с виртуальными функциями; экземпляры наследующих классов для написания проекта на платформе. Обеспечить низкую связность кода, его универсальность, возможность использовать код как open source для использования другими разработчиками в рамках своих программ. Обеспечить проект-экземпляр работы скриптов.

## Код программы

### Ability

```
using Godot;
using System;
using System.Data.Common;
using System.Runtime.CompilerServices;
public unsafe partial class Ability : Node2D, ICloneable
{
    public float CD;
    public float use_time;
    public float cost;
    protected Timer useTimer;
    protected Timer CDTimer;
    protected Node ParentalAbilityNode; //Connects parent ability node to inheriting
children classes and nodes.
    protected entity passive_application; //from
    protected entity active_application; //to
    protected string input_key; //input_key to enable kbm input in order to call the
function. Otherwise cast on CD

    public void perform(entity obj){
        if (get_state() & !is_oneshot()){
            Use(passive_application);
        }
        else{
            if (Input.IsActionJustPressed(input_key)){
                UseAbility(obj);
                passive_application = obj;
            }
        }
    }
    public void perform(entity from, entity to){
        if (get_state() & !is_oneshot()){
            Use(passive_application, active_application);
        }
        else{
            if (Input.IsActionJustPressed(input_key) || (input_key == null)){
```

```

        UseAbility(from, to);
        passive_application = from;
        active_application = to;
    }
}

public bool get_state(){
    return (bool)ParentalAbilityNode.GetMeta("is_using");
}

public void set_state(bool x){
    ParentalAbilityNode.SetMeta("is_using",x);
    return;
}

public void set_canuse(bool x){
    ParentalAbilityNode.SetMeta("CanUse",x);
    return;
}

public bool get_canuse_state(){
    return (bool)ParentalAbilityNode.GetMeta("CanUse");
}

public void set_oneshot(bool x){
    ParentalAbilityNode.SetMeta("OneShot",x);
    return;
}

public bool is_oneshot(){
    return (bool)ParentalAbilityNode.GetMeta("OneShot");
}

public object Clone()
{
    return this.MemberwiseClone();
}

protected virtual void Use(entity obj){}
protected virtual void Use(entity from, entity to){}
public void UseAbility(entity obj)
{
    if (get_canuse_state() == true){
        Use(obj);
        passive_application = obj;
        set_state(true);
        set_canuse(false);
        useTimer.Start();
    }
}

public void UseAbility(entity from, entity to)
{
    GD.Print(get_canuse_state());
    if (get_canuse_state() == true){
        Use(from, to);
        passive_application = from;
        active_application = to;
        set_state(true);
    }
}

```

```

        set_canuse(false);
        useTimer.Start();
    }
}
void _on_ready()
{
    useTimer = GetNode<Timer>("useTimer");
    CDTimer = GetNode<Timer>("CDTimer");
    useTimer.WaitTime = use_time;
    CDTimer.WaitTime = CD;
    ParentalAbilityNode = GetNode(this.GetPath());
    set_state(false);
    set_oneshot(true);
}

protected Ability(Ability Obj){
    if (this != Obj){
        this.CD = Obj.CD;
        this.use_time = Obj.use_time;
        this.cost = Obj.cost;
    }
}

protected Ability(float cd, float uset, float ct, string input_key){
    CD = cd;
    use_time = uset;
    cost = ct;
    this.input_key = input_key;
}

protected void _on_use_timer_timeout()
{
    set_state(false);
    CDTimer.Start();
}

protected void _on_cd_timer_timeout()
{
    set_canuse(true);
}

public Ability()
{
    CD=1.0f;
    use_time=0.5f;
    cost = 0;
}
}

```

## entity

```
using Godot;
using System;

public abstract partial class entity : CharacterBody2D, ICloneable
{
    public float max_speed;
    public float acceleration;
    public float friction;
    public float HP;
    public override void _PhysicsProcess(double delta){}
    public Vector2 direction;
    public Vector2 velocity;
    protected entity(entity Obj){
        if (this != Obj){
            max_speed = Obj.max_speed;
            acceleration = Obj.acceleration;
            friction = Obj.friction;
            this.HP = Obj.HP;
        }
    }
    protected entity(){
        max_speed = 300;
        acceleration = 150;
        friction = 100.0f;
        HP = 100.0f;
    }
    protected entity(float max_spd, float hp, float a, float fr){
        max_speed = max_spd;
        acceleration = a;
        friction = fr;
        HP = hp;
    }

    public abstract object Clone();

    public void perform(Ability a){
        a.perform(this);
    }
    public void perform(Ability a, entity to){
        a.perform(this, to);
    }
}
```

## Dash

```
using Godot;
using System;
public unsafe partial class Dash : Ability
{
    public float dash_speed;
    public bool ghost_on;

    protected override void Use(entity Obj)
    {
        Obj.velocity = Obj.direction * dash_speed;
    }
    public Dash(Dash Obj) : base(Obj)
    {
        dash_speed = Obj.dash_speed;
        ghost_on = Obj.ghost_on;
    }

    public Dash () : base()
    {
        dash_speed = 600f;
        ghost_on = false;
    }

    public void set(float cd, float uset, float ct, float dash_spd, bool ghost, string
input_key, bool one_s) //Connects Nodes together
    {
        CD = cd;
        use_time = uset;
        cost = ct;
        useTimer = GetNode("Ability").GetNode<Timer>("useTimer");
        CDTimer = GetNode("Ability").GetNode<Timer>("CDTimer");
        ParentalAbilityNode = GetNode("Ability");
        CDTimer.WaitTime = cd;
        useTimer.WaitTime = uset;
        dash_speed = dash_spd;
        ghost_on = ghost;
        this.input_key = input_key;
        this.set_oneshot(one_s);
    }
}
```

## Player

```
using Godot;
using System;

public partial class Player : entity
{
    private Dash dash;
    private AnimationTree AnimTree;
    private AnimationNodeStateMachinePlayback stateMachine;

    public override object Clone()
    {
        return new Player(this);
    }

    void _on_animation_tree_ready(){
        AnimTree = GetNode<AnimationTree>("AnimationTree");
        stateMachine =
        (AnimationNodeStateMachinePlayback)AnimTree.Get("parameters/playback");
    }
    void _on_dash_ready()
    {
        dash = GetNode<Dash>("Dash");
        dash.set(0.5f, 0.2f, 0, 400.0f, true, "ui_dash", true);
    }
    protected Player(Player Obj)
    {
        dash = new Dash(Obj.dash);
        HP = Obj.HP;
    }
    protected Player()
    {
        HP = 100.0f;
        max_speed = 200;
        acceleration=600;
        friction=500;
    }
    public override void _PhysicsProcess(double delta)
    {
        velocity = Velocity;
        // Input direction and handling the movement/deceleration.
        direction = Input.GetVector("ui_left", "ui_right", "ui_up", "ui_down");
        if (velocity.Length() > max_speed){
            velocity -= velocity.Normalized() * (float)(friction * delta)*1.1f;
        }
        if (direction == Vector2.Zero)
        {
            if (velocity.Length() > friction * delta){
                velocity -= velocity.Normalized() * (float)(friction * delta);
            }
        }
    }
}
```

```

        else{
            velocity = Vector2.Zero;
        }
    }
    else
    {
        //cut out unnecessary velocity
        if (direction.X == 0){velocity.X -= (float)(velocity.X * friction *
delta)*0.01f;}
        if (direction.Y == 0){velocity.Y -= (float)(velocity.Y * friction *
delta)*0.01f;}
        perform(dash);
        if (velocity.Length()<=max_speed)
        {
            velocity += direction * acceleration * (float)delta;
            velocity = velocity.LimitLength(max_speed);
        }
    }

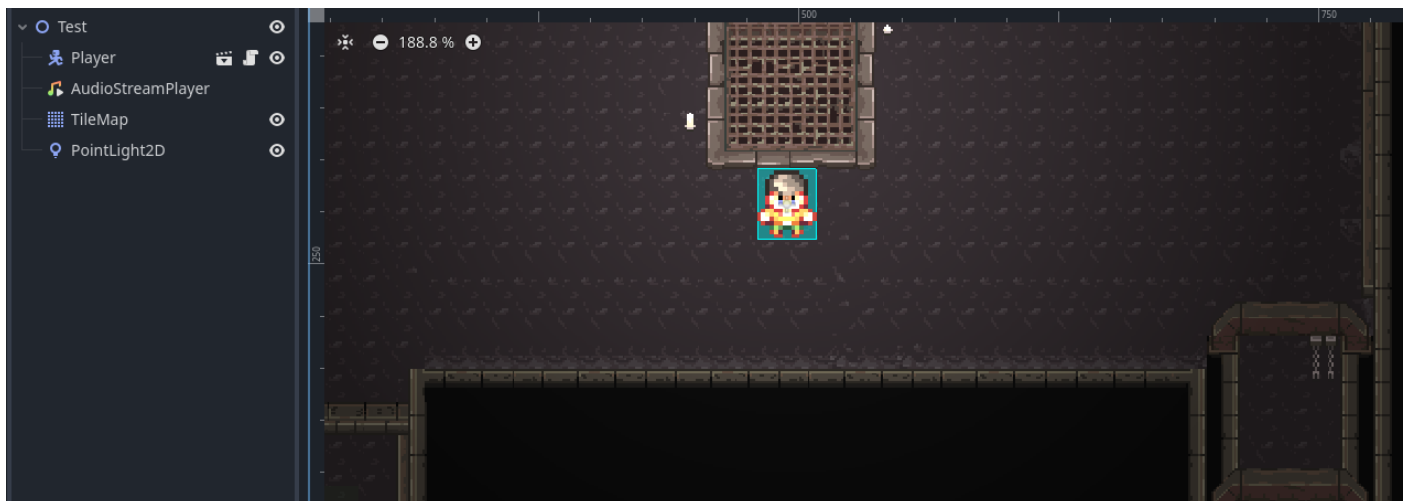
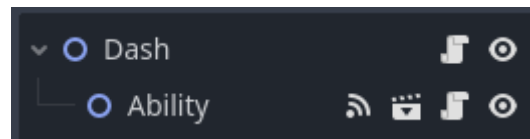
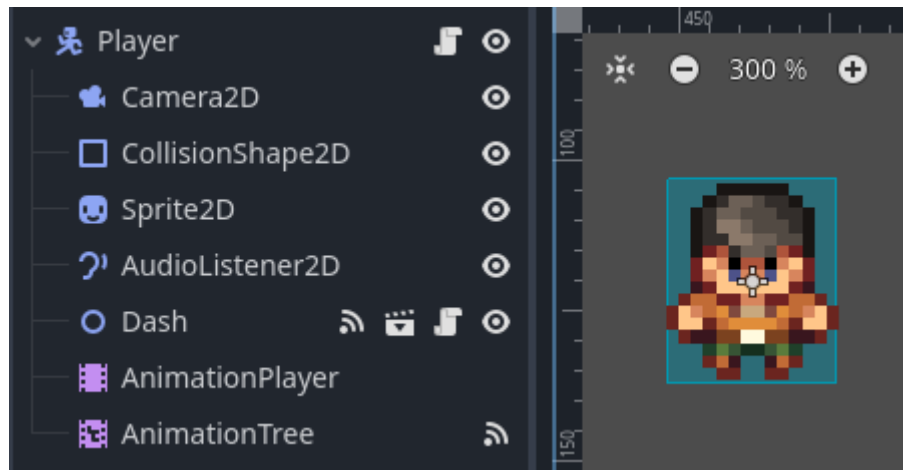
    if (velocity == Vector2.Zero){
        //res://entity/Player/Player.tscn::AnimationNodeStateMachineTransition_b1bmp
        stateMachine.Travel("IDLE");
    }
    else{
        stateMachine.Travel("Walk");
        AnimTree.Set("parameters/IDLE/blend_position",velocity.Normalized()); //Passes
velocity vector to choose animation played
        AnimTree.Set("parameters/Walk/blend_position",velocity.Normalized());
    }

    Velocity = velocity;
    MoveAndSlide();
}
}

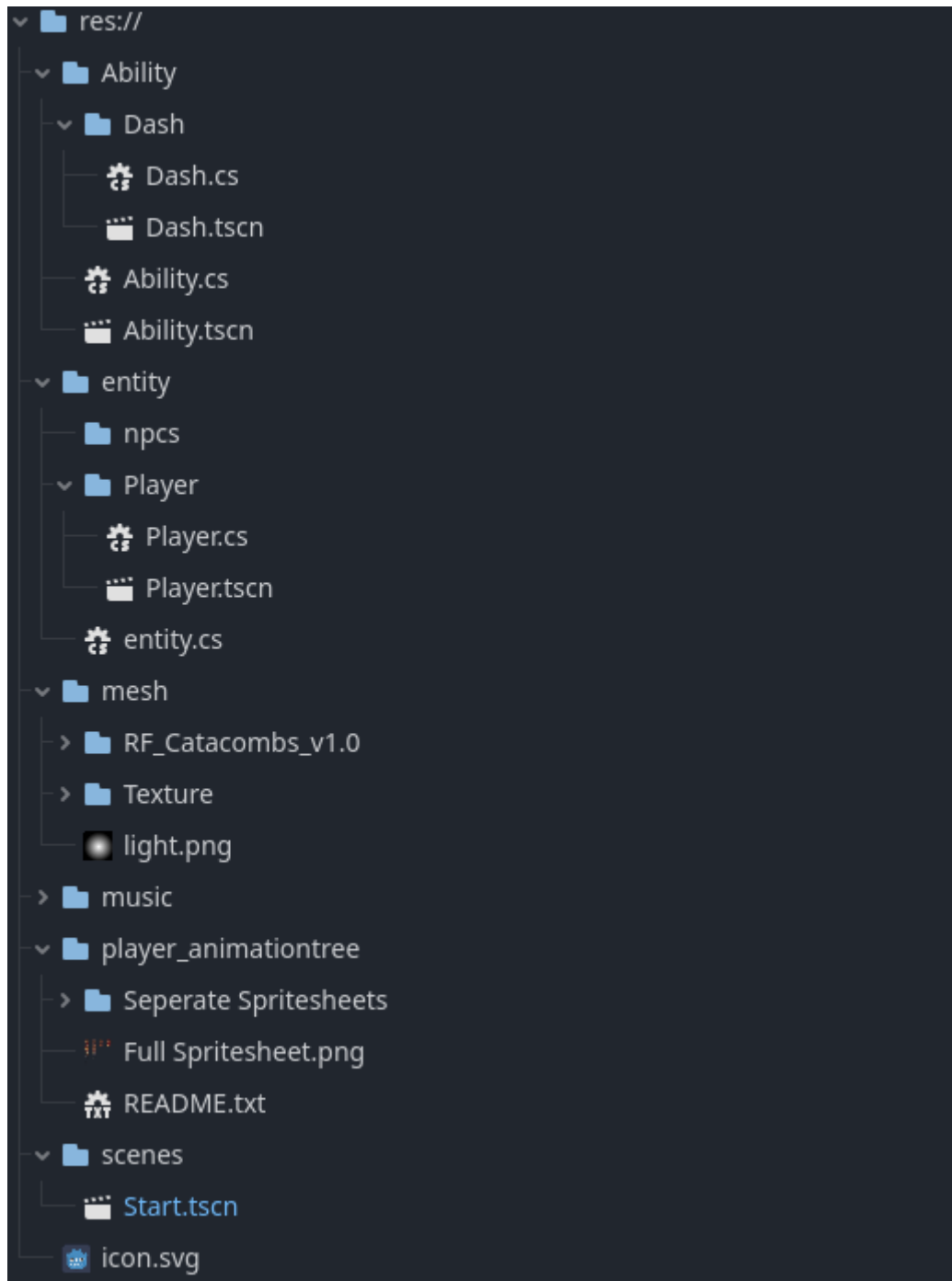
```



## Сцены в проекте



## Дерево файлов



## Результаты выполнения

Скрипты работают исправно, проект строится без ошибок и оптимально. Низкая связность кода обеспечена, добавление сущностям новых свойств и изменение прежних происходит с минимальными затратами со стороны разработчика.