DeathZone

```
using System.Collections;
using System.Collections.Generic;
using Platformer.Gameplay;
using UnityEngine;
using static Platformer.Core.Simulation;

namespace Platformer.Mechanics
{
    public class DeathZone : MonoBehaviour
    {
        void OnTriggerEnter2D(Collider2D collider)
        {
            var p = collider.gameObject.GetComponent<PlayerController>();
            if (p != null)
            {
                var ev = Schedule<PlayerEnteredDeathZone>();
                ev.deathzone = this;
            }
        }
    }
}
```

Ennemi controller

```
using System.Collections;
using System.Collections.Generic;
using Platformer.Gameplay;
using UnityEngine;
using static Platformer.Core.Simulation;

namespace Platformer.Mechanics
{
    [RequireComponent(typeof(AnimationController), typeof(Collider2D))]
    public class EnemyController : MonoBehaviour
    {
        public PatrolPath path;
        public AudioClip ouch;

        internal PatrolPath.Mover mover;
        internal AnimationController control;
        internal Collider2D _collider;
        internal AudioSource _audio;
        SpriteRenderer spriteRenderer;

        public Bounds Bounds => _collider.bounds;
```

```
        void Awake()
        {
            control = GetComponent<AnimationController>();
            _collider = GetComponent<Collider2D>();
            _audio = GetComponent<AudioSource>();
            spriteRenderer = GetComponent<SpriteRenderer>();
        }

        void OnCollisionEnter2D(Collision2D collision)
        {
            var player =
collision.gameObject.GetComponent<PlayerController>();
            if (player != null)
            {
                var ev = Schedule<PlayerEnemyCollision>();
                ev.player = player;
                ev.enemy = this;
            }
        }

        void Update()
        {
            if (path != null)
            {
                if (mover == null) mover =
path.CreateMover(control.maxSpeed * 0.5f);
                control.move.x = Mathf.Clamp(mover.Position.x -
transform.position.x, -1, 1);
            }
        }

    }
}
```

Health

```
using System;
using Platformer.Gameplay;
using UnityEngine;
using static Platformer.Core.Simulation;

namespace Platformer.Mechanics
{
    public class Health : MonoBehaviour
    {
        public int maxHP = 1;
        public bool IsAlive => currentHP > 0;

        int currentHP;
```

```
        public void Increment()
        {
            currentHP = Mathf.Clamp(currentHP + 1, 0, maxHP);
        }

        public void Decrement()
        {
            currentHP = Mathf.Clamp(currentHP - 1, 0, maxHP);
            if (currentHP == 0)
            {
                var ev = Schedule<HealthIsZero>();
                ev.health = this;
            }
        }

        public void Die()
        {
            while (currentHP > 0) Decrement();
        }

        void Awake()
        {
            currentHP = maxHP;
        }
    }
}
```

PatrolPath

```
using UnityEngine;

namespace Platformer.Mechanics
{
    public partial class PatrolPath : MonoBehaviour
    {

        public Vector2 startPosition, endPosition;

        public Mover CreateMover(float speed = 1) => new Mover(this,
speed);

        void Reset()
        {
            startPosition = Vector3.left;
            endPosition = Vector3.right;
        }
```

```
    }
}
```

PatrolPath.mover

```
using UnityEngine;

namespace Platformer.Mechanics
{
    public partial class PatrolPath
    {

        public class Mover
        {
            PatrolPath path;
            float p = 0;
            float duration;
            float startTime;

            public Mover(PatrolPath path, float speed)
            {
                this.path = path;
                this.duration = (path.endPosition -
path.startPosition).magnitude / speed;
                this.startTime = Time.time;
            }

            public Vector2 Position
            {
                get
                {
                    p = Mathf.InverseLerp(0, duration,
Mathf.PingPong(Time.time - startTime, duration));
                    return
path.transform.TransformPoint(Vector2.Lerp(path.startPosition,
path.endPosition, p));
                }
            }
        }
    }
}
```

PlayAudioClip

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

```
using UnityEngine;

public class PlayAudioClip : StateMachineBehaviour
{
    public float t = 0.5f;
    public float modulus = 0f;

    public AudioClip clip;
    float last_t = -1f;

    override public void OnStateUpdate(Animator animator,
AnimatorStateInfo stateInfo, int layerIndex)
    {
        var nt = stateInfo.normalizedTime;
        if (modulus > 0f) nt %= modulus;
        if (nt >= t && last_t < t)
            AudioSource.PlayClipAtPoint(clip,
animator.transform.position);
        last_t = nt;
    }
}
```

PlayerController

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Platformer.Gameplay;
using static Platformer.Core.Simulation;
using Platformer.Model;
using Platformer.Core;

namespace Platformer.Mechanics
{

    public class PlayerController : KinematicObject
    {
        public AudioClip jumpAudio;
        public AudioClip respawnAudio;
        public AudioClip ouchAudio;

        public float maxSpeed = 7;
        public float jumpTakeOffSpeed = 7;

        public JumpState jumpState = JumpState.Grounded;
        private bool stopJump;
        public Collider2D collider2d;
        public AudioSource audioSource;
        public Health health;
```

```
        public bool controlEnabled = true;

        bool jump;
        Vector2 move;
        SpriteRenderer spriteRenderer;
        internal Animator animator;
        readonly PlatformerModel model =
Simulation.GetModel<PlatformerModel>();

        public Bounds Bounds => collider2d.bounds;

        void Awake()
        {
            health = GetComponent<Health>();
            audioSource = GetComponent<AudioSource>();
            collider2d = GetComponent<Collider2D>();
            spriteRenderer = GetComponent<SpriteRenderer>();
            animator = GetComponent<Animator>();
        }

        protected override void Update()
        {
            if (controlEnabled)
            {
                move.x = Input.GetAxis("Horizontal");
                if (jumpState == JumpState.Grounded &&
Input.GetButtonDown("Jump"))
                    jumpState = JumpState.PrepareToJump;
                else if (Input.GetButtonUp("Jump"))
                {
                    stopJump = true;
                    Schedule<PlayerStopJump>().player = this;
                }
            }
            else
            {
                move.x = 0;
            }
            UpdateJumpState();
            base.Update();
        }

        void UpdateJumpState()
        {
            jump = false;
            switch (jumpState)
            {
                case JumpState.PrepareToJump:
                    jumpState = JumpState.Jumping;
                    jump = true;
```

```
                stopJump = false;
                break;
            case JumpState.Jumping:
                if (!IsGrounded)
                {
                    Schedule<PlayerJumped>().player = this;
                    jumpState = JumpState.InFlight;
                }
                break;
            case JumpState.InFlight:
                if (IsGrounded)
                {
                    Schedule<PlayerLanded>().player = this;
                    jumpState = JumpState.Landed;
                }
                break;
            case JumpState.Landed:
                jumpState = JumpState.Grounded;
                break;
        }
    }

    protected override void ComputeVelocity()
    {
        if (jump && IsGrounded)
        {
            velocity.y = jumpTakeOffSpeed * model.jumpModifier;
            jump = false;
        }
        else if (stopJump)
        {
            stopJump = false;
            if (velocity.y > 0)
            {
                velocity.y = velocity.y * model.jumpDeceleration;
            }
        }

        if (move.x > 0.01f)
            spriteRenderer.flipX = false;
        else if (move.x < -0.01f)
            spriteRenderer.flipX = true;

        animator.SetBool("grounded", IsGrounded);
        animator.SetFloat("velocityX", Mathf.Abs(velocity.x) /
maxSpeed);

        targetVelocity = move * maxSpeed;
    }
```

```
        public enum JumpState
        {
            Grounded,
            PrepareToJump,
            Jumping,
            InFlight,
            Landed
        }
    }
}
```

TokenController

```
using UnityEngine;

namespace Platformer.Mechanics
{

    public class TokenController : MonoBehaviour
    {
        [Tooltip("Frames per second at which tokens are animated.")]
        public float frameRate = 12;
        [Tooltip("Instances of tokens which are animated. If empty, token
instances are found and loaded at runtime.")]
        public TokenInstance[] tokens;

        float nextFrameTime = 0;

        [ContextMenu("Find All Tokens")]
        void FindAllTokensInScene()
        {
            tokens =
UnityEngine.Object.FindObjectsOfType<TokenInstance>();
        }

        void Awake()
        {

            if (tokens.Length == 0)
                FindAllTokensInScene();

            for (var i = 0; i < tokens.Length; i++)
            {
                tokens[i].tokenIndex = i;
                tokens[i].controller = this;
            }
```

```
        }

        void Update()
        {
            if (Time.time - nextFrameTime > (1f / frameRate))
            {
                for (var i = 0; i < tokens.Length; i++)
                {
                    var token = tokens[i];
                    if (token != null)
                    {
                        token._renderer.sprite =
token.sprites[token.frame];
                        if (token.collected && token.frame ==
token.sprites.Length - 1)
                        {
                            token.gameObject.SetActive(false);
                            tokens[i] = null;
                        }
                        else
                        {
                            token.frame = (token.frame + 1) %
token.sprites.Length;
                        }
                    }
                }
                nextFrameTime += 1f / frameRate;
            }
        }

    }
}
```

TokenInstance

```
using Platformer.Gameplay;
using UnityEngine;
using static Platformer.Core.Simulation;


namespace Platformer.Mechanics
{

    [RequireComponent(typeof(Collider2D))]
    public class TokenInstance : MonoBehaviour
    {
        public AudioClip tokenCollectAudio;
        [Tooltip("If true, animation will start at a random position in
```

```
the sequence.")]
        public bool randomAnimationStartTime = false;
        [Tooltip("List of frames that make up the animation.")]
        public Sprite[] idleAnimation, collectedAnimation;

        internal Sprite[] sprites = new Sprite[0];

        internal SpriteRenderer _renderer;

        internal int tokenIndex = -1;
        internal TokenController controller;
        internal int frame = 0;
        internal bool collected = false;

        void Awake()
        {
            _renderer = GetComponent<SpriteRenderer>();
            if (randomAnimationStartTime)
                frame = Random.Range(0, sprites.Length);
            sprites = idleAnimation;
        }

        void OnTriggerEnter2D(Collider2D other)
        {
            var player =
other.gameObject.GetComponent<PlayerController>();
            if (player != null) OnPlayerEnter(player);
        }

        void OnPlayerEnter(PlayerController player)
        {
            if (collected) return;
            frame = 0;
            sprites = collectedAnimation;
            if (controller != null)
                collected = true;
            var ev = Schedule<PlayerTokenCollision>();
            ev.token = this;
            ev.player = player;
        }
    }
}
```