Ces scripts sont disponibles sous le répertoire du projet : [BFL-CPNV/TPI_Projet_B-lock_BFL: Repository du projet B'lock pour le TPI (github.com)]

Zone_controller.cs

```
/*****************************************
 * Projet : B'lock
 * Nom du fichier : Lever_controller.cs
 *
 * Date des derniers changements : 23.05.2022
 * Version : 1.1
 * Auteur : Fardel Bastien
 *****************************************/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Zone_controller : MonoBehaviour
{
    // Variables
    private bool player_passed_through = false;

    // Objets associés au contrôleur
    private GameObject Player;
    private Player_controller player_script;
    [SerializeField] private GameObject exit;

    /// <summary>
    /// Awake est appelé quand l'instance de script est chargée
    /// </summary>
    private void Awake()
    {
        Player = GameObject.FindWithTag("Player");
        player_script = Player.GetComponent<Player_controller>();
    }

    /// <summary>
    /// OnTriggerEnter2D est une fonction spécifique qui est appelée lorsque le
collider détecte un objet entrant celui-ci
    /// </summary>
    /// <param name="collision"></param>
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player") && !player_passed_through) // Le joueur
doit être l'objet détecté dans la collision et il doit aussi y entrer pour la première
fois
        {
            exit.SetActive(!exit.activeSelf); // Referme la sortie derrière le joueur
            player_script.recorded_data.Clear(); // Nettoie les données enregistrées
pour empêcher le joueur de revenir à la zone précédente
            player_passed_through = !player_passed_through;
        }
    }
}
```

Player_controller.cs

```
/*****************************************
 * Projet : B'lock
 * Nom du fichier : Player_controller.cs
 *
 * Date des derniers changements : 20.05.2022
 * Version : 1.1
 * Auteur : Fardel Bastien
 *****************************************/

using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Linq;
using UnityEngine;

public class Player_controller : MonoBehaviour
{
    // Constantes
    private const int REWINDINDEX = 1;
    private const float MOVEMENTSPEED = 200;
    private const float JUMPFORCE = 7.5f;
    private const float CHECKRADIUS = 0.1f;

    // Variables
    private bool is_grounded;
    private bool is_rewinding = false;
    private float horizontal_movement;

    // Variables spécifiques à Unity
    private Vector3 velocity = Vector3.zero;
    private LayerMask what_is_ground;

    // Objets du joueur
    private Rigidbody2D player_rigidbody2d;
    private SpriteRenderer player_sprite_renderer;
    private Animator player_animator;
    private Transform feet_position;
    public List<RewindData> recorded_data;

    /// <summary>
    /// Awake est appelé quand l'instance de script est chargée
    /// </summary>
    private void Awake()
    {
        // récupération des composants attachés au joueur
        player_rigidbody2d = GetComponent<Rigidbody2D>();
        player_sprite_renderer = GetComponent<SpriteRenderer>();
        player_animator = GetComponent<Animator>();
        feet_position = transform.GetChild(0).GetComponent<Transform>();
        what_is_ground = LayerMask.GetMask("Environnement");
        recorded_data = new List<RewindData>();

        // S'assure que la liste est vide avant de l'utiliser pour la première fois
        recorded_data.Clear();
    }

    /// <summary>
    /// FixedUpdate est appelé pour chaque trame avec un taux fixe, si le
MonoBehaviour est activé
    /// </summary>
```

```
    private void FixedUpdate()
    {
        int index = recorded_data.Count - REWINDINDEX; // Obtention de l'index de la
dernière entrée

        // Afin d'éviter de remonter le temps lorsqu'il n'y a pas de données
enregistrées.
        if (index > REWINDINDEX)
        {
            is_rewinding = Input.GetKey(KeyCode.R);
        }
        else
        {
            is_rewinding = false;
        }

        player_animator.SetBool("rewind", is_rewinding);

        if (is_rewinding) // Bloque les actions du joueurs si celui-ci remonte le
temps
        {
            RewindData();
        }
        else
        {
            horizontal_movement = Input.GetAxis("Horizontal") * MOVEMENTSPEED *
Time.deltaTime; // Récupère l'input du joueur pour l'utiliser pour le déplacement

            MovePlayer(horizontal_movement);
            CheckIfGrounded();
            RecordData(transform.position.x, transform.position.y,
player_sprite_renderer.flipX, horizontal_movement);
        }
    }

    /// <summary>
    /// Update est appelé une fois par mise à jour de trame
    /// </summary>
    private void Update()
    {
        CheckIfGrounded();

        if (!is_rewinding) // Bloque les actions du joueurs si celui-ci remonte le
temps
        {
            if (is_grounded && Input.GetKey(KeyCode.W))
            {
                player_rigidbody2d.velocity = Vector2.up * JUMPFORCE; // Applique une
force de saut au joueur
            }
        }

        if (is_grounded)
        {
            player_animator.SetBool("is_jumping", !is_grounded); // Renvoie la valeur
false puisque le joueur ne saute pas
        } else
        {
            player_animator.SetBool("is_jumping", !is_grounded); // Renvoie la valeur
true puisque le joueur saute
        }
    }
```

```
    /// <summary>
    /// MovePlayer est une fonction qui applique une vitesse au joueur afin que celui-
ci puisse se déplacer de gauche à droite
    /// </summary>
    /// <param name="horizontal_movement"></param>
    private void MovePlayer(float horizontal_movement)
    {
        Vector3 targetVelocity = new Vector2(horizontal_movement,
player_rigidbody2d.velocity.y); // Création de la vitesse cible du joueur
        FlipSprite(horizontal_movement);
        player_animator.SetFloat("speed", Mathf.Abs(horizontal_movement));

        player_rigidbody2d.velocity = Vector3.SmoothDamp(player_rigidbody2d.velocity,
targetVelocity, ref velocity, .05f); //déplacement du joueur en appliquant une vitesse
à son rigidbody2D par rapport à la vitesse actuelle et à celle ciblée
    }

    /// <summary>
    /// CheckIfGrounded est une fonction qui vérifie si le joueur est actuellement au
sol en se basant sur la position de ses pieds et la distance de ceux-ci par rapport au
sol
    /// </summary>
    private void CheckIfGrounded()
    {
        is_grounded = Physics2D.OverlapCircle(feet_position.position, CHECKRADIUS,
what_is_ground); // crée un cercle autours des pieds du joueur avec un radius constant
afin de vérifier si le sol est touché
    }

    /// <summary>
    /// FlipSprite est une fonction qui retourne le sprite du joueur pour qu'il soit
face à la direction dans laquelle celui-ci se dirige
    /// </summary>
    /// <param name="horizontal_movement"></param>
    private void FlipSprite(float horizontal_movement)
    {
        if (horizontal_movement > 0.01f)
        {
          player_sprite_renderer.flipX = false;
        }
        else if (horizontal_movement < -0.01F)
        {
          player_sprite_renderer.flipX = true;
        }
    }

    /// <summary>
    /// RecordData est une fonction qui enregistre les données nécessaires au retour
dans le temps, elle reçoit en paramètres la position du joueur (x,y), l'état de
retournement du sprite et la vitesse actuelle du joueur
    /// </summary>
    /// <param name="x_position"></param>
    /// <param name="y_position"></param>
    /// <param name="is_flipped"></param>
    /// <param name="current_speed"></param>
    private void RecordData(float x_position, float y_position, bool is_flipped, float
current_speed)
    {
        Vector2 player_current_position = new Vector2(x_position, y_position); // Crée
un nouveau vecteur pour stocker les positions x,y plus facilement
```

```
        int index = recorded_data.Count - REWINDINDEX;

        if (!recorded_data.Any() || recorded_data[index].player_position !=
player_current_position) // S'assure que la liste n'est pas vide ou que la dernière
position enregistrée n'est pas la même que celle qui doit l'être
        {
            RewindData data_to_record = new RewindData();

            data_to_record.player_position = player_current_position;
            data_to_record.is_flipped = is_flipped;
            data_to_record.player_speed = current_speed;

            recorded_data.Add(data_to_record);
        }
    }

    /// <summary>
    /// RewindData est une fonction qui permet au joueur de remonter le temps en
lisant ses actions enregistrée une à la fois
    /// </summary>
    private async void RewindData()
    {
        if (recorded_data.Count > 0) // S'assure que la liste n'est pas vide
        {
            await Task.Delay(200); // Permet de ralentir et rendre le retour dans le
temps "un peu plus lisse"
            int index = recorded_data.Count - REWINDINDEX;

            transform.position = recorded_data[index].player_position;
            player_sprite_renderer.flipX = recorded_data[index].is_flipped;
            player_animator.SetFloat("speed",
Mathf.Abs(recorded_data[index].player_speed));

            recorded_data.RemoveAt(index); // Retire la donnée qui a été lue

            player_rigidbody2d.velocity = Vector2.zero ;
        }
    }
}
```

Pause_menu_controller.cs

```
/*****************************************
 * Projet : B'lock
 * Nom du fichier : Player_controller.cs
 *
 * Date des derniers changements : 23.05.2022
 * Version : 1.0
 * Auteur : Fardel Bastien
 *****************************************/

using UnityEngine;
using UnityEngine.SceneManagement;

public class Pause_menu_controller : MonoBehaviour
{
    // Variables
    public static bool game_is_paused = false;

    // Objets
    private Player_controller player_script;
    private GameObject Player;
    [SerializeField]private GameObject pause_menu_ui;

    /// <summary>
    /// Awake est appelé quand l'instance de script est chargée
    /// </summary>
    private void Awake()
    {
        Player = GameObject.FindWithTag("Player");
        player_script = Player.GetComponent<Player_controller>();
    }

    /// <summary>
    /// Update est appelé une fois par mise à jour de trame
    /// </summary>
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.P))
        {
            if (game_is_paused)
            {
                ResumeGame();
            }
            else
            {
                PauseGame();
            }
        }
    }

    /// <summary>
    /// ResumeGame est une fonction qui est appelée lorsque le joueur souhaite
reprendre la partie
    /// </summary>
    public void ResumeGame()
    {
        player_script.enabled = true; // Activation du script du joueur pour lui
permettre de se déplacer
        pause_menu_ui.SetActive(false); // Désactivation du menu pause
        Time.timeScale = 1; // Réactivation du temps
        game_is_paused = false;
```

```
    }

    /// <summary>
    /// PauseGame est une fonction qui est appelée lorsque le joueur souhaite mettre
en pause la partie
    /// </summary>
    private void PauseGame()
    {
        player_script.enabled = false; // Désactivation temporaire du script du joueur
afin d'éviter les déplacements non voulus
        pause_menu_ui.SetActive(true); // Activation du menu pause
        Time.timeScale = 0; // Arrêt du temps
        game_is_paused = true;
    }

    /// <summary>
    /// QuitGame est une fonction qui est appelée lorsque le joueur appuie sur Quitter
    /// </summary>
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

Main_menu_controller.cs

```
/*****************************************
 * Projet : B'lock
 * Nom du fichier : Player_controller.cs
 *
 * Date des derniers changements : 23.05.2022
 * Version : 1.0
 * Auteur : Fardel Bastien
 *****************************************/

using UnityEngine;
using UnityEngine.SceneManagement;

public class Main_menu_controller : MonoBehaviour
{
    // Permet de ne pas avoir le level à charger codé en dur, il suffit de le préciser
dans l'inspecteur
    [SerializeField] private string level_to_load;

    /// <summary>
    /// StartGame est une fonction qui est appelée lorsque le bouton du même nom est
appuyé pour lancer le jeu
    /// </summary>
    public void StartGame()
    {
        SceneManager.LoadScene(level_to_load);
    }

    /// <summary>
    /// QuitGame est une fonction qui est appelée lorsque le bouton du même nom est
appuyé pour quitter le jeu
    /// </summary>
    public void QuitGame()
    {
        Application.Quit();
    }
}
```

Lever_controller.cs

```
/*****************************************
 * Projet : B'lock
 * Nom du fichier : Lever_controller.cs
 *
 * Date des derniers changements : 20.05.2022
 * Version : 1.1
 * Auteur : Fardel Bastien
 *****************************************/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Lever_controller : MonoBehaviour
{
    // Variables
    private bool is_player_in_range;
    private bool is_lever_off = true; // État de base du levier
    private bool is_rewinding;

    // Objets du levier
    private Animator lever_animator;
    [SerializeField] private GameObject[] connected_obstacles;

    /// <summary>
    /// Awake est appelé quand l'instance de script est chargée
    /// </summary>
    private void Awake()
    {
        // récupération du composant attaché au levier
        lever_animator = GetComponent<Animator>();
    }

    /// <summary>
    /// Update est appelé une fois par mise à jour de frame
    /// </summary>
    void Update()
    {
        is_rewinding = Input.GetKey(KeyCode.R);

        if (!is_rewinding) // S'assure que le joueur ne puisse pas interagir avec le
levier s'il remonte le temps
        {
            if (Input.GetKeyDown(KeyCode.E) && is_player_in_range)
            {
                InteractWithLever();
            }
        }
    }

    /// <summary>
    /// InteractWithLever est une fonction qui permet au joueur d'interagir avec les
leviers du jeu, modifiant ainsi l'environnement de celui-ci pour résoudre des puzzles
    /// </summary>
    private void InteractWithLever()
    {
        is_lever_off = !is_lever_off;
        lever_animator.SetBool("is_lever_off", is_lever_off);
        lever_animator.SetTrigger("interact");
        foreach (var obstacle in connected_obstacles)
```

```
        {
            obstacle.SetActive(!obstacle.activeSelf);
        }
    }

    /// <summary>
    /// OnTriggerEnter2D est une fonction spécifique qui est appelée lorsque le
collider du levier détecte un objet entrant celui-ci
    /// </summary>
    /// <param name="collision"></param>
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player")) // Vérifie si le joueur a été détecté
entrant dans la collision
        {
            is_player_in_range = true;
        }
    }

    /// <summary>
    /// OnTriggerExit2D est une fonction spécifique qui est appelée lorsque le
collider du levier détecte un objet quittant celui-ci
    /// </summary>
    /// <param name="collision"></param>
    private void OnTriggerExit2D(Collider2D collision) // Comme pour l'entrée, la
sortie interdit l'interaction au levier.
    {
        if (collision.CompareTag("Player")) // Vérifie si le joueur a été détecté
quittant la collision
        {
            is_player_in_range = false;
        }
    }
}
```

RewindData.cs

```
/*********************************************
 * Projet : B'lock
 * Nom du fichier : RewindData.cs
 *
 * Date des derniers changements : 17.05.2022
 * Version : 1.0
 * Auteur : Fardel Bastien
 *********************************************/

using UnityEngine;

/// <summary>
/// RewindData est une classe utilisée pour instancier une liste en précisant les
éléments qu'elle contient
/// </summary>
public class RewindData
{
    public Vector2 player_position;
    public bool is_flipped;
    public float player_speed;
}
```