Multi-Input First Person Controller

Manual

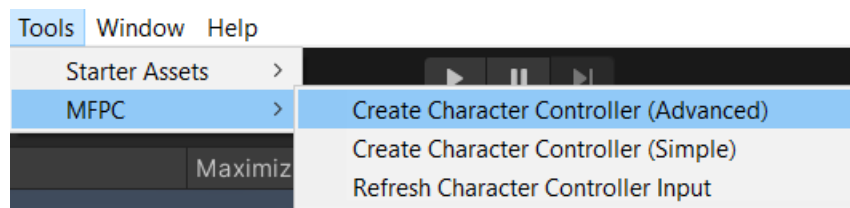# Content

About Asset

MFPC - Multi-Input First Person Controller includes an easy-to-use and flexible first-person character control system that supports all input types.

Quick start

Go to the Tools/MFPC/Create Character Controller (Advanced) or (Simple)

And create character controller.

In the Old Input System, use StandaloneInputModule. For a New Input System, use InputSystemUIInputModule.

Input System

This asset supports both the **New** and **Old** Input System.
The Old Input System supports:
- Mobile control
- Keyboard/Mouse control

The new input system supports:
- Mobile control
- Keyboard/Mouse control
- Gamepad control

If a character has been created in your scene and you want to change the Input System. After changing to New or Old. Go to the Tools/MFPC/Refresh Character Controller Input.
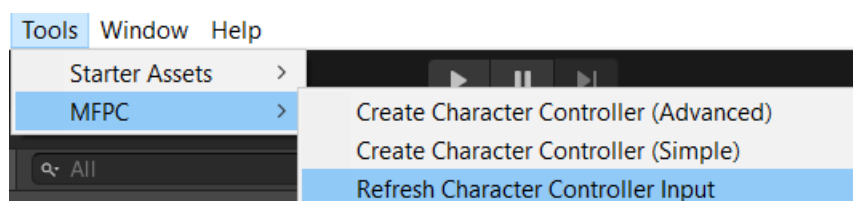
A window will open where you need to click on the "Refresh Input" button.
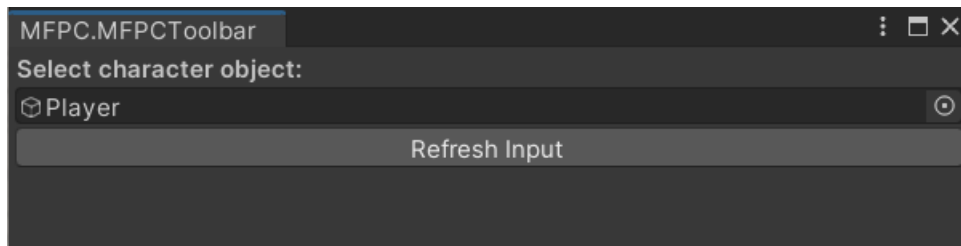
If you want to use a New Input System, then you must have it imported.
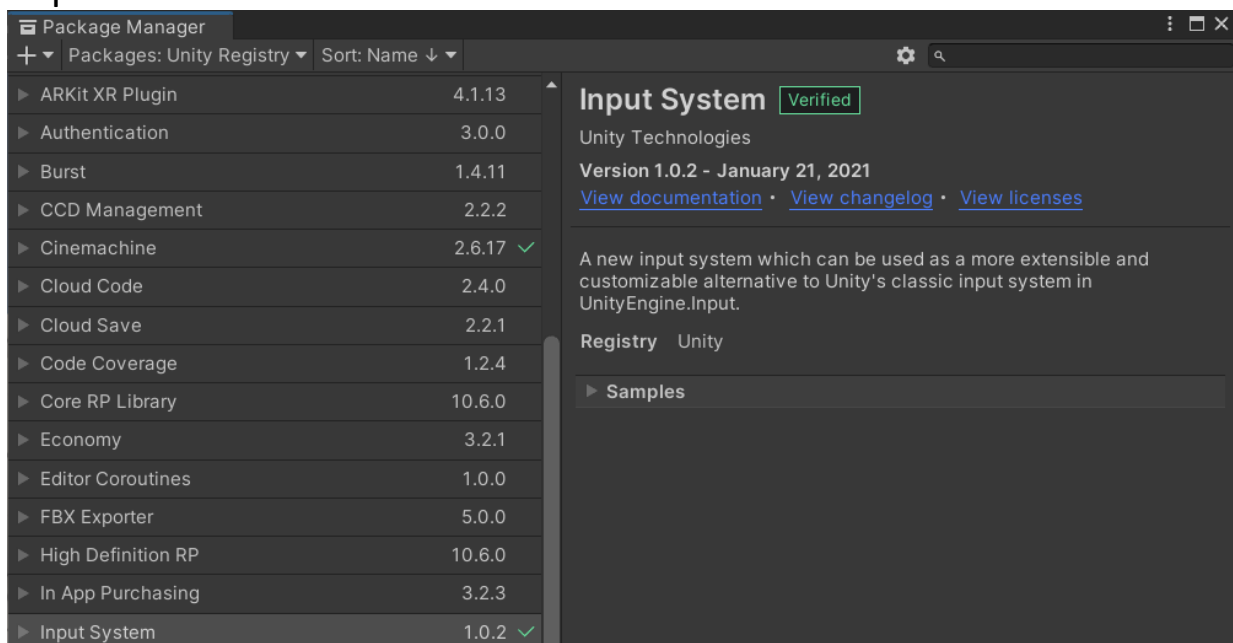
And in Project Settings/Player in the Active Input Handling you must chose Input System Package (New)
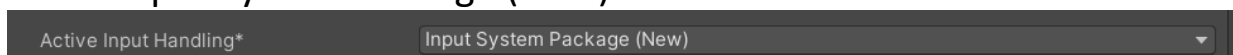
In the MFPC files you can find Input Config. Initial Input Type means what Input will be when you first load the game. When Adaptive Input is enabled, the Initial Input Type will be selected automatically.
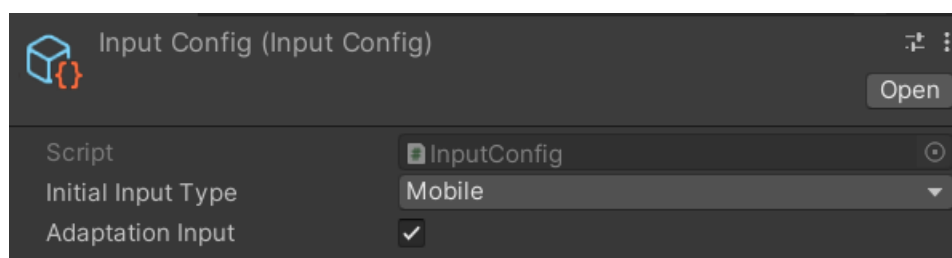
Input map

Action	Keyboard	Gamepad
Move forward	W	Left analog stick forward
Move backward	S	Left analog stick backward
Turn left	A	Left analog stick left
Turn right	D	Left analog stick right
Jump	Space	B
Sit	Ctrl	A
Run	Shift	L2
Lean left	Q	L1
Lean right	E	R1
Settings	Tab	Start button

Ladder system

You need to add a "Ladder" prefab to the object you want to climb. The ray that comes from the prefab indicates the direction of the climb to the ladder.
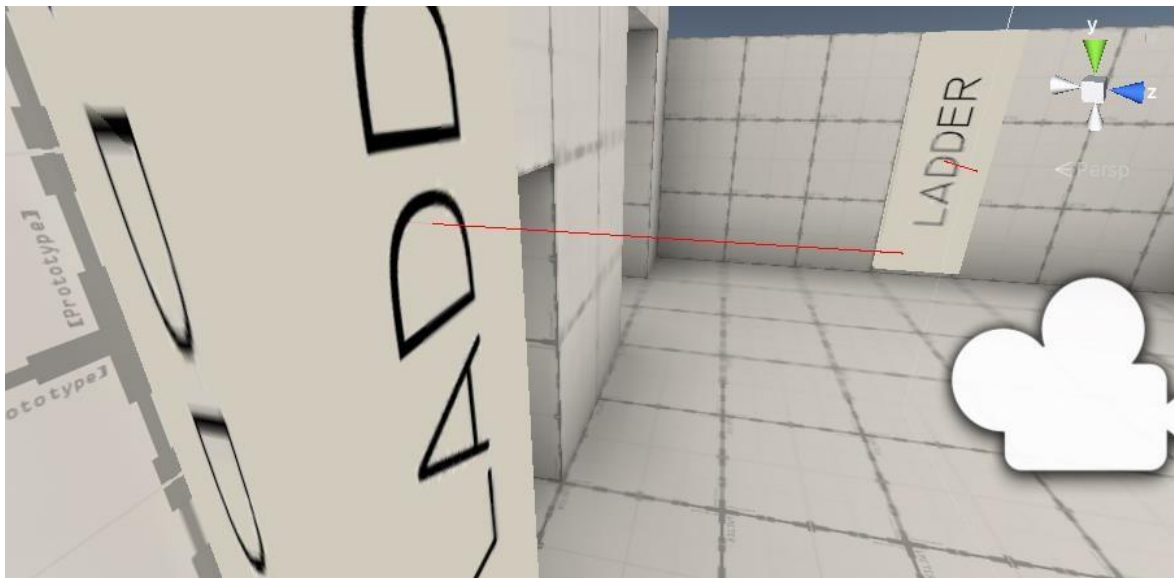
You can also make the trigger invisible by disable the "Mesh

Renderer" component.
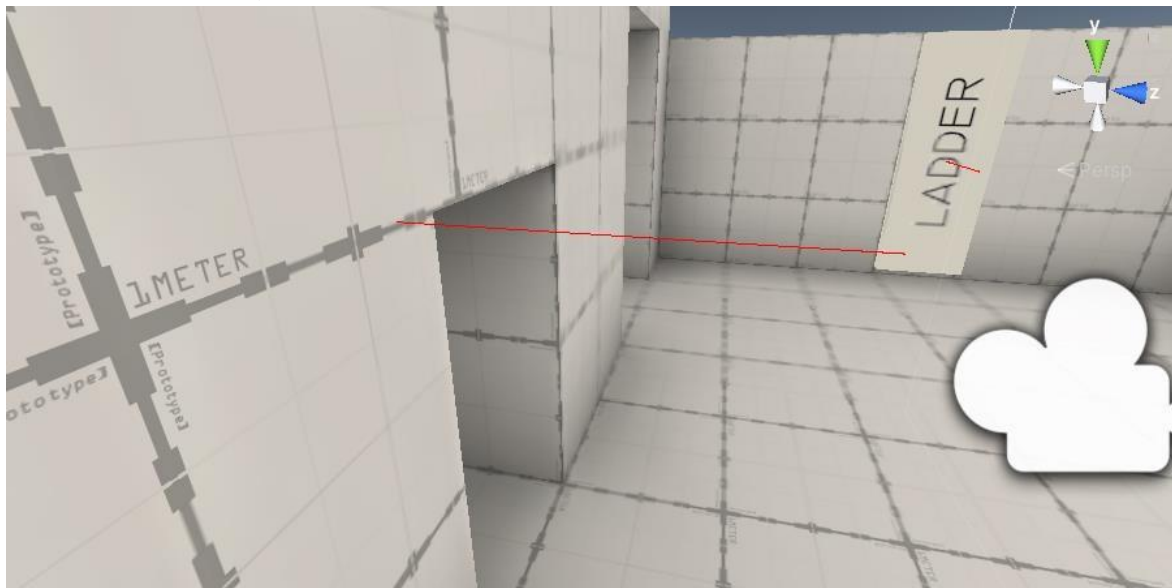
Attention! For this to work, the staircase state must be enabled in the Player.

Footstep System

You can add your own templates for footstep sounds. Create a folder Assets/Resources/StepsData. Right click there. Select Create/Scriptableobject/StepData and create a template.
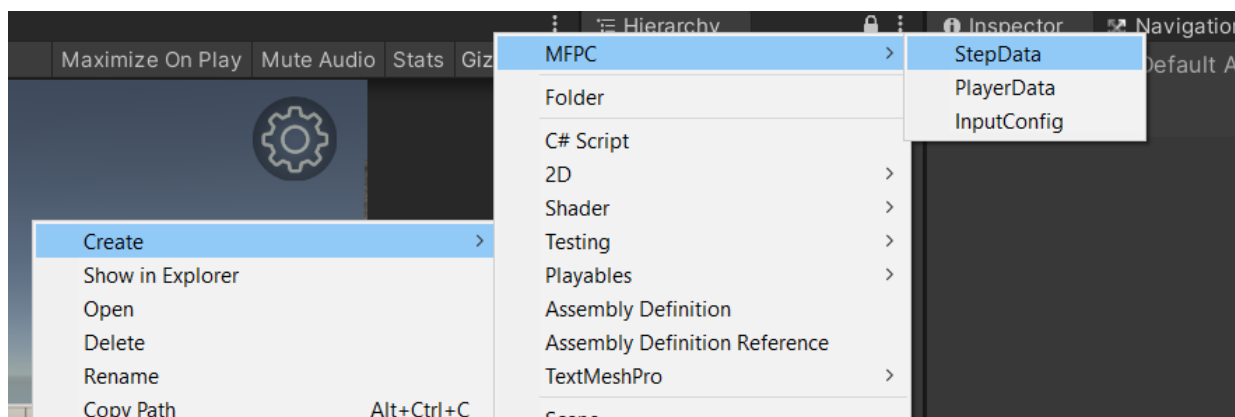
In the first section, add textures that will play random footstep sounds. In the second section, add the sound of one step. I recommend adding 4 or more sounds.
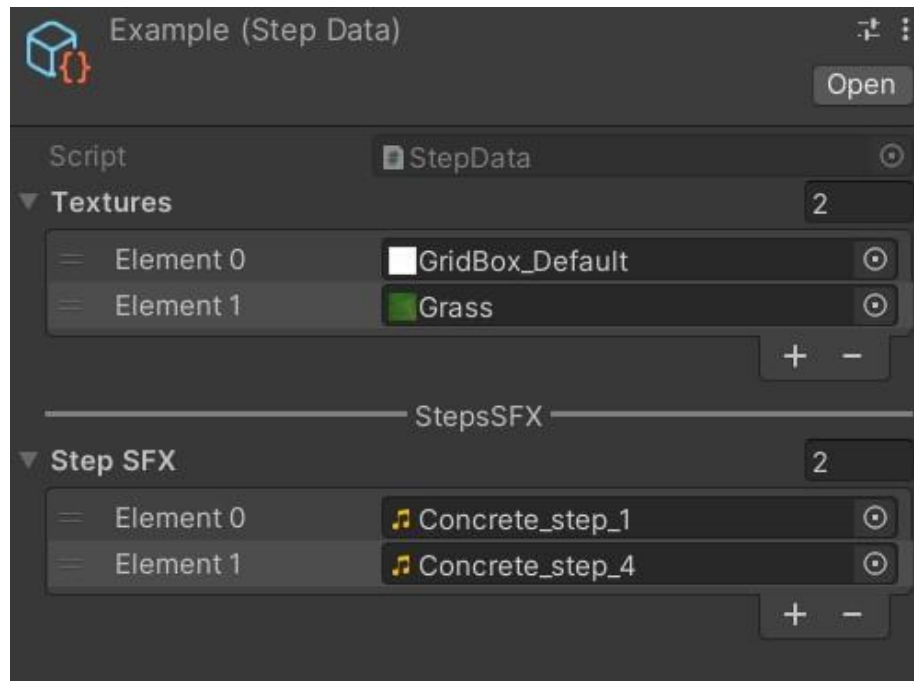
Next, add the "MFPC Footstep SFX" script to the player and specify the templates we need in the Steps Data section.
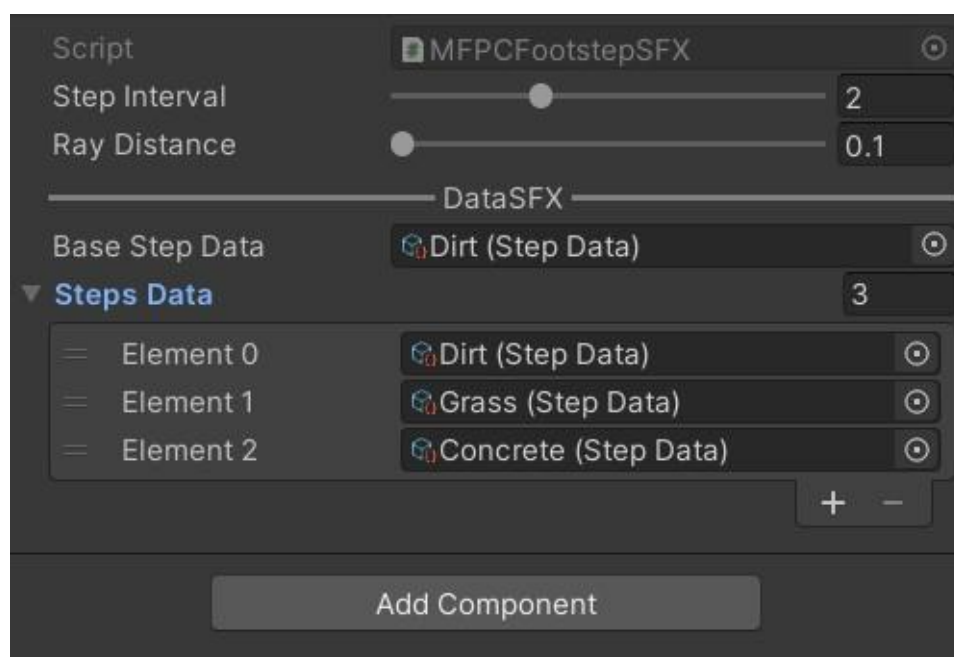
Interactions with physical objects

To do this, you need to add "MFPCPushObject" to the player. If you do not need this mechanic, remove this component from the player prefab.

Player System

And with an asset, you can very easily customize the player's possible states for yourself. To do this, you need to configure them in Player Data. To enable a state, you need to select it from the top field. And to delete, click the "Remove" button on it.
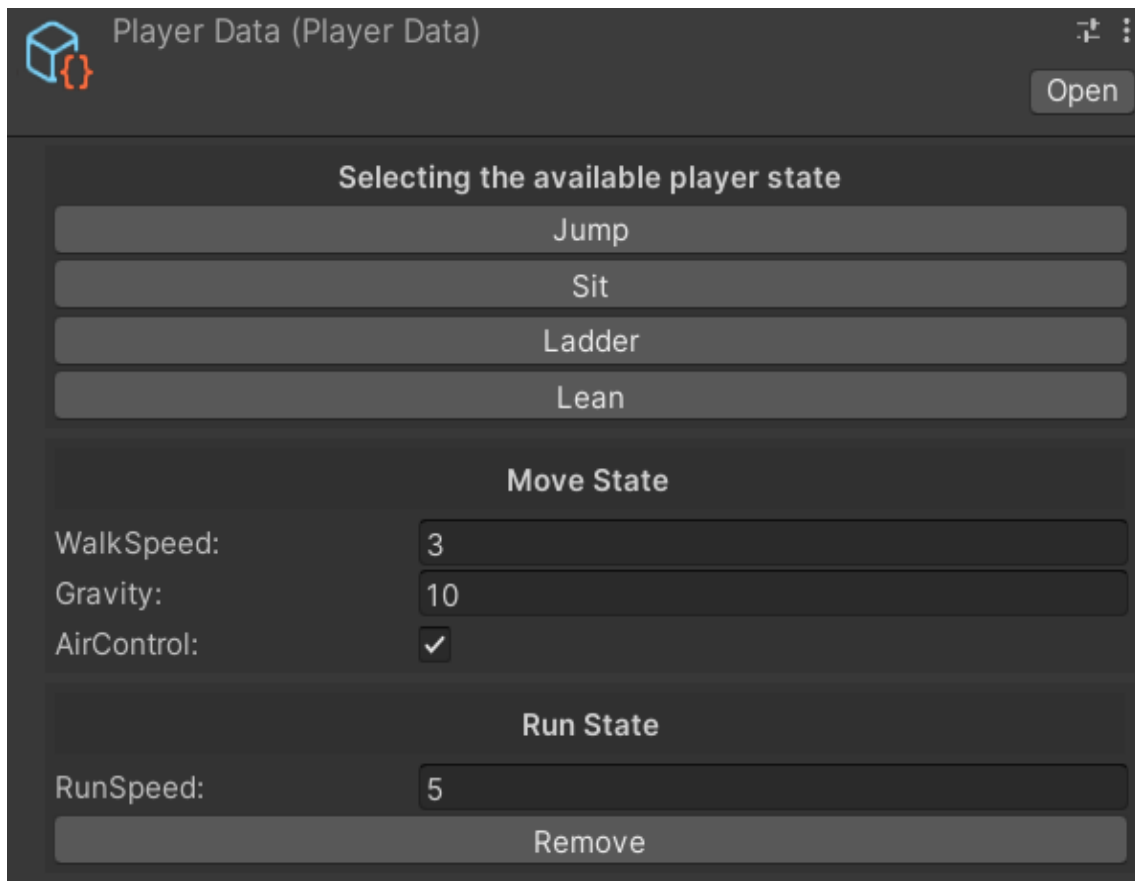
Player Stats

To add stats, the player needs to add the desired stat to the character.
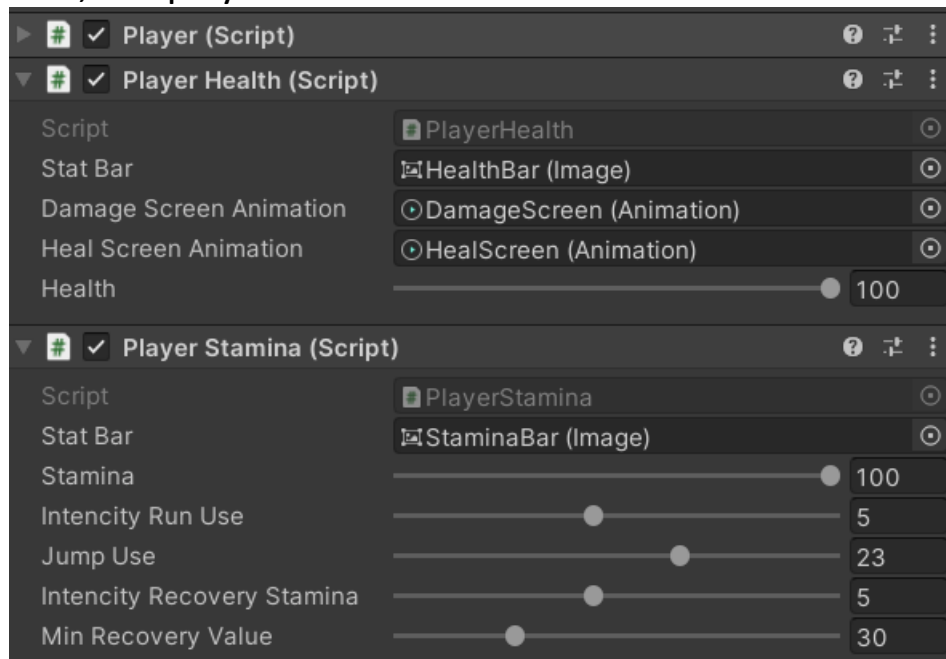
Script Reference

<u>Control Components</u>

public class TouchField : MonoBehaviour

```
Tracks the distance between the position of the wheelbarrow on the
previous frame and the current one.
```

public *Vector2* GetSwipeDirection;
```
The distance between the position of the wheelbarrow on the previous
frameand the current one.
```

___

public class RunField : MonoBehaviour

```
Allows you to create a button that will set the running state.
```

public bool isLockRun;
```
Equals True when the button is active
```

___

public class Joystick : MonoBehaviour
```
Sets the direction of movement depending on the direction of the joystick
stick
```

public event *Action<Vector2>* OnJoystickDragged;

Notifies about changes in motion vector

protected Image joystick, handle;

Joystick components.

protected float timeToFadeIn;

Time after how many seconds the joystick will change the transparency to 0.

private bool fixedJoystick;

If it is True, then the joystick will always be in the same place.

private bool fadingJoystick;

If it is True, then the joystick will change transparency.

protected virtual void FadingJoystick();

Smoothly changes the transparency of the entire joystick to 0.

protected virtual void ChangeJoystickAlpha(float alpha);

Changes the transparency of the entire joystick.

`public bool SetFixedJoystickState;`

Change fixedJoystick value.

`public bool SetFadingJoystickState;`

Change fadingJoystick value.

---

public class JoystickWithRunField : Joystick

Add an additional way to control the character.

private RunField runField;

Run button

Player States

public class MFPCPlayer : MonoBehaviour

public event Move MoveCondition;

The event sends the motion state at the current time.

public *PlayerStateMachine* StateMachine;
StateMachine - Designed to change the player's state

public void ChangeMoveCondition(MoveConditions newMoveCondition);

Changes the current state of movement if the old state is different.

---

public *class* PlayerStateMachine

public *PlayerState* CurrentState;
Current executable state

---

public abstract *class* PlayerState

protected static *Vector3* moveDirection;
The direction in which the player is heading

protected static *float* lookDirection;
The direction the player is facing (Vertical)

public virtual *bool* IsChanged()
True if it is possible to go to this state

---

public class MFPCMovement : PlayerGroundedState
The basic movement script allows you to move, turn the camera horizontally
to direct the position

private void MovePlayer();

Responsible for moving the character using the "Character Controller"

Component.

---

public class MFPCRun : PlayerGroundedState
Allows the player to move forward only.

private *bool* TryCheckRunAbility()
Check if the player has a stamina component, if so, checks if it does.

---

public class MFPCJump : PlayerGroundedState
Allows the character to jump.

private *void* Jump()
Directs the character up.

private *bool* TryCheckJumpAbility()
Check if the player has a stamina component, if so, checks if it does.

---

public class MFPCSit : PlayerGroundedState

Allows the character to sit down (reducing character height).

private void Sit()
Changes character's height.

private *bool* IsStandUp()
Checks if space is at the top.

---

public class MFPCLadderMovement : PlayerState
Allows the player to climb ladder.

private void Climb();
Character movement process.

private void RotatePlayer();
Rotate the character for the direction of movement.

public void ClimbUp(LadderArea ladderArea);
Ladder start interaction action.

private void ClimbDown(Vector3 climbDownDirection);
Character stops interacting with ladder.

public class MFPCCameraRotation : MonoBehaviour

Allows you to rotate the camera vertically.

public float SetRotation;
Changes the angle of rotation of the camera.

public class MFPCCameraAnimation : MonoBehaviour
Animates the camera based on the movement state.

private float changeAnimationSpeed;
How smoothly the movement animations will change.

private float timeToFallAnimation;
The time after which the fall animation will play. This is necessary so that it is not performed from small falls.

private void OnMoveAnimation(MoveConditions moveCondition);
Changes the animation based on the current movement state.

---

public class MFPCPushObject : MonoBehaviour

Allows you to interact with objects with a Ridgetbody component.

private float forcePush;
The force with which the object will be pushed.

---

public class LadderArea : MonoBehaviour
The area the player enters switches their state to ladder movement.

public Transform LadderTransform;
Object transform.

public Vector3 BottomLadderPosition
The position where the ray from the top corner of the collider cross the surface from below.

private Vector3 StartLadderPosition;
The lowest part of the collider stairs.

private Vector3 EndLadderPosition;
The topmost part of the collider.

private void FindBottomPosition();
Find the lowest position after which it is impossible to continue descending;

public bool IsBottomPosition(Vector3 playerPosition);
The range of position Y at which the playerPosition climb down.

public bool IsStartLadderPosition(Vector3 playerPosition);

Checks if the player has descended to the start of the ladder position.

bool IsEndLadderPosition(Vector3 playerPosition);
Checks if the player has climbed to the top of the ladder.

---

public class PlayerHealth : PlayerStat

public void SetDamage(float damage)
```
Decreases amount of health.
```

public void SetHeal(float heal)
```
Increases amount of health.
```

---

public class PlayerStamina : PlayerStat

public *bool* RunAbility;
```
True if it is possible to continue running
```

public *bool* JumpAbility;
```
True if it is possible to jump
```

private float intencityRunUse;
```
The intensity of the use of stamina while running.
```

private float jumpUse;
```
The number that subtracts from the stamina value when jumping.
```

private float intencityRecoveryStamina;
```
Stamina recovery rate.
```

private float minRecoveryValue;
```
The value at which the Run Field will be turned back on.
```

---

public class MFPCFootstepSFX : MonoBehaviour

private float stepInterval;
```
Distance to play the next sound.
```

private float rayDistancef;
```
Ray length at which texture is detected.
```

private StepData baseStepData;
```
Play if not found StepData with texture.
```

public class StepData : ScriptableObject
Allows you to set the textures on which the sound of steps will be played.

private Texture[] textures;
Textures on which the sounds of steps will be played.

public AudioClip GetStepSFX();
Getting the sound of a step.

public bool CompareTexture(Texture targetTexture);
Checks if the texture we need is in StepData.

public class PlayerData : ScriptableObject

public *PlayerStates* AvailablePlayerStates;
Available states for the player selected in the inspector.

Move

public *float* WalkSpeed;
Normal movement speed.

public *float* Gravity;
The force with which the player will fall.

Run

public *float* RunSpeed;
The speed at which the character will run.

Jump

public *float* JumpForce;
The strength with which the character will jump.

public *AudioClip* JumpSFX ;
Jump sound.

Sit

public *float* SitWalkSpeed;
The speed at which the player crawls

public float SitHeight;
CharacterController height when character is crouched.

public float ChangeSitSpeed;
Altitude change rate.

public *float* RayDistance;
The length of the ray that checks if space is at the top.

public *AudioClip* SitSFX
Sound when a character crouches and *stands* up

Ladder

public float climbSpeed;
The speed at which the character will climb to the top.

public float NormalizationSpeed;
Character movement speed to the center of the ladder.

public *float* NormalizationRotateSpeed;
Character rotate speed to the ladder.

public *Vector2* RangeCameraRotation;
character rotation limitation *on* stairs.

public *AudioClip* ClimbSFX;
The sound when a character climbs onto *the* stairs.

public *bool* IsAvailableState(*PlayerStates state*)
Checks whether such a condition exists.

Lean
public float LeanAngle;
Lean camera angle.

public float LeanOffsetPositionX;
Lean camera position offset.

public float LeanSpeed;
Camera shift speed.

public float LeanMoveSpeed;
Player movement speed in Lean State.

Support

Check the Publisher Page for contact information.