Documentation

This code defines the behavior of a player character in a 2D platformer game. It allows the player to move left or right, jump, climb up ladders, and attack enemies. The player can interact with objects and exit levels. The code also includes functions to handle player death and the player being hit by enemies. It uses Unity physics components such as Rigidbody2D, BoxCollider2D, and PolygonCollider2D, and UnityStandardAssets.CrossPlatformInput to enable cross-platform input. The code makes use of animations to represent the player's movements and actions.

The Player class represents the behavior and actions of the game's main character. It extends the MonoBehaviour class and is attached to the player's game object.

# Fields

* runSpeed (float): the speed at which the player can move horizontally.
* jumpSpeed (float): the speed at which the player can jump.
* climingSpeed (float): the speed at which the player can climb.
* attackRadius (float): the radius of the area where enemies can be hit by the player's attack.
* hitKick (Vector2): a vector representing the direction and force of the kick when the player hits an enemy.
* hurtBox (Transform): a transform representing the area where enemies can be hit by the player's attack.
* myRigidBody2d (Rigidbody2D): a reference to the player's Rigidbody2D component.
* myAnimator (Animator): a reference to the player's Animator component.
* myBoxCollider2D (BoxCollider2D): a reference to the player's BoxCollider2D component.
* myPlayersFeet (PolygonCollider2D): a reference to the player's PolygonCollider2D component.
* startingGravityScale (float): the player's gravity scale at the start of the game.
* isHurting (bool): a flag indicating whether the player is currently being hurt.

# Methods

* Start()

This method is called before the first frame update. It initializes the player's Rigidbody2D, Animator, BoxCollider2D, and PolygonCollider2D components. It also sets the starting gravity scale of the player's Rigidbody2D component.

* Update()

This method is called once per frame. It checks whether the player is currently hurting or not, and then calls the Run(), Jump(), Climb(), Attack(), and ExitLevel() methods.

* ExitLevel()

This method checks if the player is touching the layer Interactable and if the player has pressed the Vertical button. If both conditions are true, it loads the next level through the ExitDoor component.

* Attack()

This method is called when the player presses the Fire1 button. It triggers the Attacking animation and then uses Physics2D.OverlapCircleAll() to find all the enemies within the attackRadius. Then, it calls the Dying() method of each enemy within the radius.

* PlayerHit()

This method is called when the player collides with an enemy. It kicks the player away from the enemy using the hitKick vector, sets the isHurting flag to true, plays the Hitting animation, and calls the ProcessPlayerDeath() method through the GameSession component. It then starts the StopHurting() coroutine.

* StopHurting()

This coroutine is called by the PlayerHit() method. It waits for 2 seconds and then sets the isHurting flag to false.

* Climb()

This method checks if the player is touching the Climbing layer. If it is, it sets the Climbing animation to true, disables gravity, and allows the player to move up and down with the Vertical input. If it is not, it sets the Climbing animation to false and re-enables gravity.

* The Jump()

Jump method is responsible for allowing the player to jump if they are touching the ground. It first checks if the player's feet are touching any objects with the "Ground" layer mask. If the player is not on the ground, the method exits and the player cannot jump. If the player is on the ground, the method checks if the player has pressed the jump button (Jump), which is handled by the CrossPlatformInputManager. If the jump button is pressed, the method sets the player's vertical velocity to the jumpSpeed value specified in the class's serialized fields. The horizontal velocity is not affected, so the player will continue moving horizontally at the same speed. Overall, the Jump() method allows the player to jump only when they are on the ground and have pressed the jump button.

* private void Climb()

This method is called in the Update method to check if the player is touching any climbing layers. If the player is touching a climbing layer, the method gets the control input from the player (vertical axis) and sets the player's velocity to move only vertically with a speed of "climbingSpeed" while keeping the horizontal velocity the same as before. The animator is set to the "Climbing" state and the player's gravity scale is set to 0 so that the player can move up and down the ladder without falling. If the player is not touching any climbing layer, the animator is set to the "not climbing" state and the player's gravity scale is set back to the starting gravity scale.

* private void Jump()

This method is called in the Update method to check if the player is touching the ground. If the player is not touching the ground, the method returns and the player cannot jump.If the player is touching the ground, the method checks if the "Jump" button has been pressed. If the "Jump" button has been pressed, the player's velocity is set to a new vector with the same horizontal velocity as before but a vertical velocity of "jumpSpeed", which makes the player jump.

* private void Run()

This method is called in the Update method to get the horizontal control input from the player and set the player's velocity to move horizontally at a speed of "runSpeed". The method also calls two other methods: "FlipSprite()" and "ChangingToRunningState()".

* private void ChangingToRunningState()

This method is called by the "Run()" method to check if the player is running horizontally. If the player's horizontal velocity is greater than "Mathf.Epsilon" (a very small positive number), the animator is set to the "Running" state. Otherwise, the animator is set to the "not running" state.

* private void FlipSprite()

This method is called by the "Run()" method to flip the player's sprite horizontally depending on the player's horizontal velocity. If the player is moving to the right (positive horizontal velocity), the sprite faces right. If the player is moving to the left (negative horizontal velocity), the sprite faces left.

* private void OnDrawGizmosSelected()

This method draws a wire sphere around the player's "hurtBox" to visualize the area of effect for the player's attack. The sphere has a radius of "attackRadius" and is drawn in the Scene view of the Unity Editor.