



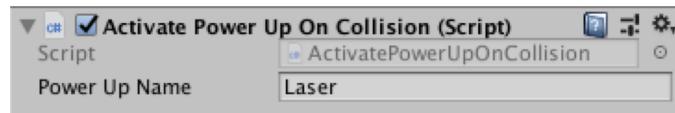
## Power-ups

## Power-ups

- Cada *power-up* es implementado por un **componente** distinto
  - Componentes [Enlarge](#) y [Laser](#) para el jugador
- Un *power-up* está **habilitado** si su componente está activado en el jugador
  - Todos deshabilitados al principio
  - Para probarlos, se activan/desactivan desde el editor
- No puede haber más de un *power-up* activo de forma simultánea

# Gestión de los *power-ups*

- La activación y desactivación de los *power-ups* se gestiona en el jugador
  - Con el componente [PowerUpManager](#)
- Para que funcione, los *power-ups* necesitan un componente [ActivatePowerUpOnCollision](#)
  - Configurado con el nombre del componente asociado



## PowerUpManager

- Gestiona los *power-ups* del jugador
  - Permite activar *power-ups*
  - Controla si algún *power-up* está habilitado y cuál
- Proporciona el método  
[ActivatePowerUp \(string powerUpName\)](#)  
que recibe el nombre del *power-up* a activar y lo activa
  - El nombre debe coincidir con el nombre del componente
- Si había un *power-up* previo activo, lo desactiva

# PowerUpManager

```
public class PowerUpManager : MonoBehaviour {  
  
    MonoBehaviour currentPowerUp;  
  
    public void ActivatePowerUp (string powerUpName) {  
  
        // Localiza el componente powerUpName asociado  
        MonoBehaviour powerUp = GetComponent (powerUpName) as MonoBehaviour;  
  
        if (powerUp == null) {  
            Debug.Log("Componente power-up " + powerUpName + " no encontrado. Se ignora."  
            return;  
        }  
  
        if (powerUp == currentPowerUp)  
            // Si ya está activo, no hace nada (termina la ejecución del método)  
            return;  
  
        if (currentPowerUp != null)  
            // Desactiva el power-up activo  
            currentPowerUp.enabled = false;  
  
        // Activa el power-up indicado  
        powerUp.enabled = true;  
        currentPowerUp = powerUp;  
  
        Debug.Log("Componente power-up " + powerUpName + " activado.");  
    }  
}
```

# ActivatePowerUpOnCollision

- Activa un *power-up* del jugador cuando el objeto al que pertenece colisiona con él
  - Configurable el nombre del *power-up* a activar
    - Debe ser el nombre del componente que lo implementa
- El *game object* debe pertenecer a una capa física que únicamente colisiona con el jugador
  - Si el componente detecta una colisión con cualquier cosa que no sea el jugador (capa física *Player*), dará un aviso y no hará nada
  - El *game object* del jugador debe tener el componente *PowerUpManager* que gestiona los *power-ups*

# ActivatePowerUpOnCollision

```
public class ActivatePowerUpOnCollision : MonoBehaviour {

    [SerializeField]
    string powerUpName = null;
    int playerLayer;

    void Start() {
        // Identificador de la capa física "Player"
        playerLayer = LayerMask.NameToLayer("Player");
    }

    void OnCollisionEnter2D(Collision2D info) {
        GameObject other = info.gameObject;

        if (other.layer != playerLayer) {
            Debug.Log("Un power-up se ha chocado con algo distinto al jugador."
                + " Debes tener mal la configuración de las capas de colisión.");
            return;
        }

        PowerUpManager pum = other.GetComponent<PowerUpManager> ();
        if (pum == null) {
            Debug.Log("Jugador sin gestor de power-ups. Se ignora el power-up conseguido.");
        }
        else {
            pum.ActivatePowerUp (powerUpName);
        }
    }
}
```

## Comportamiento de los power-ups

- **Enlarge**
  - Al activarse, el jugador ve incrementada su anchura
    - En un 25%
    - No debe atravesar los bordes ni la bala con su nueva anchura
  - Al desactivarse, recupera su tamaño normal
- **Laser**
  - Al activarse, cambia el *sprite* del jugador y cada vez que el usuario pulsa el botón de salto (barra espaciadora), se instancia un objeto predefinido *Fire* saliendo de la pala, a modo de balas
  - Al desactivarse, el jugador recupera su *sprite* habitual

# Enlarge

- Al menos, tres formas de conseguirlo
  1. Cambiar la escala del objeto
  2. Cambiar el *sprite* del objeto
  3. Cambiar a otro objeto ya pre-configurado
- Consecuencias de decantarse por cada una de ellas
  1. Afecta a la escala de sus objetos hijos
    - Habría que ajustarla por código
  2. No afecta al *collider*
    - Habría que ajustarlo por código
  3. Cambios en el diseño de los *scripts*
    - El diseñador deja los objetos configurados
      - No reutilizable si queremos hacer cambios en el porcentaje de aumento

## Reajustar la escala de los hijos

- Cuando se cambia la escala del *game object*
  - El cambio en el *transform* afecta a los descendientes, que también se ven afectados por el cambio
- Para impedir que la pelota engorde al cambiar la escala de la pala, hay que controlarlo

```
// Afecta a las entidades hijas y hay que
// deshacerlo, para que no les afecte
newScale.x = 1/scale;

for (int i = 0; i < transform.childCount; i++) {
    transform.GetChild(i).localScale = newScale;
}
```

- Y reasignar la escala original al desactivar el *power-up*