MegaMan Engine for Unity

Made by MegaChibisX

Weapon Creation

Weapons in this project are split in two main scripts. One handles everything from the player's side, like the input and animations. The other one handles everything in-game, like the contact of the bullet, enemy damage and deflections.

Pl_Weapon:

The **Pl_Weapon** script is the script that handles everything in-game. It's the simplest of the two. Making custom scripts based on this will be mostly up to you. You need to put the creativity in here yourself, I can't tell you how to make your weapons act. There are however a few important variables and methods the script comes with. Before you get to work, it's best to know how it works.

Variables:

This script comes with a few variables which can tell the bullet how to act.

- **weaponType**: This tells what the **Special Weapon** that fired the bullet was. It can be used by an enemy to tell if they are weak to the weapon that hit them.
- **damage**: This is the base damage the weapon will do to an enemy. The enemy shouldn't be shielded or invincible.
- **ignoreInvis**: This will allow the weapon to pierce invisibility frames.
- **ignoreShield**: This will allow the weapon to pierce shielded enemies.
- **destroyOnWalls**: The weapon will be destroyed once it hits a wall.
- **destroyOnAllEnemies**: The weapon will be destroyed by any enemy it hits. (Ex: Normal shot).
- **destroyOnBigEnemies**: The weapon will be destroyed by enemies that have more health than the weapon's damage. It can pierce multiple weak enemies. (Ex: Charged shot)**.**
- **deflectClip**: The sound effect that plays once the weapon gets deflected.

And then there's the methods:

- **Start()**: This is called when the weapon is shot.
- **Deflect()**: The enemy calls this method to deflect the bullet. This starts the deflected behavior.
- **OnCollisionEnter2D(Collision2D collision)**: If the weapon has a non-trigger collider, and that collider hits another non-trigger collider, this method is called.
- **OnTriggerEnter2D(Collider2D other)**: If one or both of the colliders are triggers, this is called instead.

This is really all there is to this script. Enemy damage happens from the **Enemy** script, in the **Damage()** method.

There is also the **Pl_Shot** script, which is pretty much the same script, but the bullet moves to its local right instantly. There's a good chance you will just need to use this script for some weapons. For example, both the Pharaoh Shot and Metal Blade use just this script.

If you do make custom bullets, the prefix **PlWp_** can help you remember what script this is. It's good to keep your project tidy.

Pl_WeaponData:

This script is where all the magic happens. Again, both the Pharaoh Shot and Metal Blade bullets can be made simply by adding the **Pl_Shot** component to a GameObject with the right sprite. What tells them how and where to fire is the **Pl_WeaponData** script.

First it's a good idea to pay the base **Pl_WeaponData** script a visit.

On the very top you will likely notice an enum called **Weapons** and an array called **WeaponList**. Those two are connected, the array is where the weapons themselves are found, and things like their names and colors are held in, and the enum is used as a pointer to the array. For example, the first element of **WeaponList** is the MegaBuster. Similarly, the first element of **Weapons** is the MegaBuster. Each enum needs to match the position of its weaponin the **WeaponList**. "**Length**" has no match, is used by various scripts, and needs to be last.

When you make your own weapons, you are also going to expand this list with your own weapons. You can also remove weapons the engine came with. Just make sure the enum and array are matching, and **Length** always exists in the enum and is last.

Now it's time for the variables:

- **owner**: This is the current player of the weapon. It doesn't need to be assigned by default, as the player's side will handle this when the weapon is equipped.
- **baseColors**: These are the main colors of the weapon. These always need to be assigned, and the player will have them on their armor when the weapon is equipped. There's three, and their order is **Light, Dark, Outline**.
- **chargeColors**: These colors are colors the player can change into if the weapon can be charged. This is a 2D array, meaning it's an array of arrays. Assuming you have **chargeColors[x, y]**, **x** is meant to handle charge levels, and **y** is meant to handle the many different colors for each charge level. For example, the MegaBuster has two charge levels, and each has 3 different colors you can change into.
- **weaponEnergy**: This keeps track of the weapon's current energy.
- **maxWeaponEnergy**: This caps the weapon's energy, normally to 28.
- **energyBarFill** and **energyBarEmpty**: These are used to draw the weapon's energy on the UI. Can be found in "**Assets/Resources/Sprites/Menus/BarsMenu**". You can edit this sprite, or make your own if you know how to load sprites from the resources yourself
- **weaponIcon** and **weaponIconGray**: These are the icon of the weapon that should show in the pause menu, when it is selected and when it is not. Can be found under "**Assets/Resources/Sprites/Menus/WeaponIcons**". You can edit this sprite, or make your own if you know how to load sprites from the resources yourself.
- **menuBarFull** and **menuBarEmpty**: These are the bars that show when the weapon is displayed in the pause menu. Doesn't need to be assigned.

And of course, the methods follow:

- **Pl_WeaponData(Player _owner, string _menuName, WeaponColors _baseColor, WeaponColors[,] _chargeColors)**:
  This is the script's constructor. This is what gets called when the script is first loaded, during the game's first loading moment. This should be left as it is, as it may load before other important Unity files, so things like loading from the Resources might not work as expected. These things should be done in **Init()** instead.
- **Init()**: This is called when the stage first starts. Won't be called upon losing a life, only when you get here from the Stage Select.
- **Start()**: This is called when the player equips the weapon.
- **Update()**: This is called every frame the weapon is active.
- **Press()**: This is called when the fire button is pressed. Shooting bullets usually happens here.
- **Hold()**: This is called every frame the fire button is held. Charging usually happens here.
- **Release()**: This is called when the fire button is released. Releasing charged shots usually happens here.
- **Cancel()**: This is called when the weapon's actions need to be canceled. Used to prevent issues like the charge sound playing when the player dies or changes weapons.
- **OnGUI(float x, float y)**: This is the method that draws the weapon energy on the screen. **x** and **y** each are 1/32 of the screen's **width** and **height**.
- **GetColors()**: This is where the player's armor's current colors are decided. It normally returns the weapon's default colors. You will have to change the charge colors yourself if you want your weapon to be chargable.

Setting up the script:

Start by making your script and giving it the appropriate name. The prefix **PlWpDt** is useful.

As soon as the script is made, the Editor gives an error. This is because the parent script has a **Constructor** defined, and so the child script also needs a **Constructor**. You can simply add this code:

```
  public PlWpDt_<WeaponName>(Player _owner, string _menuName, WeaponColors _baseColors,
WeaponColors[,] _chargeColors) : base(_owner, _menuName, _baseColors, _chargeColors)
  {

  }
```

You can expand it if you want, but this can work as it is.

Now go to the original **Pl_WeaponData** script. Find the **Weapons** and **WeaponList** variables. Add your weapon's name in the enum, anywhere before the **Length** element. Now add your weapon in the same exact spot in the **WeaponList**. For example, if your **Weapons** is third, your **WeaponList** should also be third.

Next step is to assign the **weaponIcon** and **weaponIconGray** sprites in the **Init()** method., as you will get an error if you try entering the pause menu without them. Load them with the following code:

```
    Sprite[] sprites = Resources.LoadAll<Sprite>("Sprites/Menus/WeaponIcons");
    weaponIcon = sprites[x];
    weaponIconGray = sprites[x + 1];
```

x is the location of your sprite in the sprite sheet. You can also use your own.

You can also load the **energyBarEmpty** and **energyBarFull** similarly, with the sprite being found at "**Sprites/Menus/Bars**", or you can take them from the Player's **bars**, if you define a variable for one yourself.

Shooting a normal bullet:

Let's start by making Bass's weapon. While Bass's weapon is multi-directional, but we'll stick with the normal shot for this section, and expand it later.

To shoot a bullet when the **Fire** button is pressed, the **Press()** method will be used. We will instantiate a bullet, put it in front of the player and give it the right damage and speed. We already have the **Pl_Shot** script available for the bullet, so it will use that.

To shoot a bullet, a method will be made for convenience. This can be used to shoot a bullet normally:

```
protected void Shoot(Vector3 offset, bool shootLeft, int damageMultiplier, float speed, float charge) {
    Pl_Shot newShot = null;
    newShot = Object.Instantiate(Resources.Load<Pl_Shot>("Prefabs/PlayerWeapons/MegaManSmallShot"));

    // Finds the right position and orientation for the shot.
    newShot.transform.position = owner.transform.position + offset;
    if (shootLeft)
        newShot.transform.localScale = Vector3.Scale(newShot.transform.localScale, new Vector3(-1, 1, 1));
    // Sets the damage and speed of the shot.
    newShot.damage *= damageMultiplier;
    newShot.speed = speed;
}
```

You can replace the bullet path with your own, as well as the damage and speed. To rotate the bullet, use the following:

```
newShot.transform.localRotation = Quaternion.AngleAxis(<angle> * (shootLeft ? -1 : 1), Vector3.forward);
```

Replace <angle> with the desired angle in degrees.

Alright, variables:

- **offset**: Relative position the bullet will be shot from.
- **shootLeft**: If the bullet should move left or right.
- **damageMultiplier**: damage multiplier.
- **speed**: Speed of the bullet. Default is 300.
- **charge**: The charge level in seconds. You can change the **newShot** object during instantiation based on your charge level.

Back to the **Press()** method. Wdd the **Shoot()** method to shoot the bullet. To play the shooting animation, we will set the player's **shootTime** to 0.2f. And finally, we will play a shoot sound. This should make Bass shoot.

Multi Directional Shooting:

Building upon the previous stuff, we will find the shooting direction of Bass's bullets. The bullets can go up, up and right, right or down and right. For each case, we'll need to find the angle. This is easily done by using the player's **input** variable. If input.y == 0, then it's straight forward. If not, then it depends.

It's all easily done in the **Press()** method. The **Shoot()** method also gets a new parameter, so we can give it the angle ourselves. Where the angles are calculated, we also check if the owner of the weapon is Bass, and if it is we tell him to play the appropriate animation.

Rapid Fire:

To fire bullets rapidly, we will replace the **Press()** method with the **Hold()** method. Now bullets will be fired every frame. And with that, we've made rapid fire!

Add a cooldown timer that gets reduced in **Update()** and check if the cooldown <= 0. If it is, fire a bullet.