Character Stats

A system to organize stats and their modifiers.

If you want an in-depth look at the implementation of this asset, watch this YouTube playlist: https://www.youtube.com/playlist?list=PLm7W8dbdflohqccuwJxjYRKuDUnk131XO.

Quick Usage Guide

Using Character Stats is simple. All you have to do is:

1. Import the namespace.

```
using Kryz.CharacterStats
```

2. Declare a variable of type `CharacterStat` and instantiate it.

```
CharacterStat strength = new CharacterStat(baseValue: 10);
```

3. Declare and instantiate a `StatModifier`. Modifiers have one of three types: `Flat`, `PercentAdd` or `PercentMult`.

```
StatModifier modifier = new StatModifier(5, StatModType.Flat);
```

4. Add the modifier to the stat.

```
strength.AddModifier(modifier);
```

5. If you want to remove the modifier.

```
strength.RemoveModifier(modifier);
```

Modifiers with source

You can set any object to be the `Source` of a modifier. For example, an `Item` class that applies modifiers to the player's stats when equipped.

```
public class Item
{
```

```
    public void Equip(Player player)
    {
        // Source is set to 'this' item
        StatModifier modifier = new StatModifier(5, StatModType.Flat, source:
this);

        player.Strength.AddModifier(modifier);
    }
}
```

You can then use the `RemoveAllModifiersFromSource()` function to remove all modifiers that were applied by a particular source.

```
public class Item
{
    public void Unequip(Player player)
    {
        // Removes all modifiers that were applied by 'this' item
        player.Strength.RemoveAllModifiersFromSource(this);
    }
}
```

CharacterStat

This is the public API of the `CharacterStat` class:

BaseValue

```
public float BaseValue;
```

The base value is initially set in the constructor. It can also be changed by assigning the `BaseValue` field. This is the value of the stat when it has no modifiers.

Value

```
public float Value { get; }
```

The stat's `Value` is calculated by applying all the modifiers to the `BaseValue`. The final value is only recalculated if the modifiers or the `BaseValue` have changed since the last time the `Value` property was accessed.

StatModifiers

```
public readonly ReadOnlyCollection<StatModifier> StatModifiers;
```

A public read-only way to access the modifiers that have been applied to the stat. Useful to display detailed information to the player about what is modifying their stats.

AddModifier()

```
public virtual void AddModifier(StatModifier mod)
```

Adds a new modifier to the stat.

RemoveModifier()

```
public virtual bool RemoveModifier(StatModifier mod)
```

Removes a specific modifier from the stat. Returns true the modifier was found and removed. False if the requested modifier was not present in the stat's modifiers list.

RemoveAllModifiersFromSource()

```
public virtual bool RemoveAllModifiersFromSource(object source)
```

Removes all modifiers from a particular source. Returns true if at least one modifier was removed. False otherwise.

StatModifier

`StatModifer` is a `readonly struct`. That means none of its values can be changed after instantiation.

```
public readonly struct StatModifier : IEquatable<StatModifier>
{
    public readonly float Value;
    public readonly StatModType Type;
    public readonly int Order;
    public readonly object Source;
}
```

These are all the values you can supply to the constructor. Some of them are required and others are optional.

Value

```
public readonly float Value;
```

Required. How much the stat should be modified by. It will affect the stat differently depending on `Type`.

Type

```
public readonly StatModType Type;
```

Required. How the modifier will affect the stat. Possible values are:

1. `StatModType.Flat`

   Adds directly to the stat's base value.

2. `StatModType.PercentAdd`

   Multiplies the stat by `1 + Value`. I.e., a modifier value of `0.5` will increase the stat by 50% (multiply by `1.5`). While a value of `-0.5` will decrease the stat by 50% (multiply by `0.5`).

   Stacks additively. I.e., two `0.5` modifiers will result in a 100% increase to the stat: `BaseValue * (1 + 0.5 + 0.5)`.

3. `StatModType.PercentMult`

   Same as `PercentAdd`, except these stack multiplicatively. I.e., two `0.5` modifiers will result in a 125% increase to the stat: `BaseValue * 1.5 * 1.5`.

Order

```
public readonly int Order;
```

Optional. This is the order in which the modifier will apply. By default the order is determined by the `Type`:

- `StatModType.Flat` - Order = 100
- `StatModType.PercentAdd` - Order = 200
- `StatModType.PercentMult` - Order = 300

However, you can manually set your own `Order` if you want a modifier to apply differently. For example, if you want a special kind of `Flat` modifier that applies between `PercentAdd` and `PercentMult`, set an `Order` anywhere between `201` and `299`.

Source

```
public readonly object Source;
```

Optional. Sets the source of the modifier. In other words, stores the entity/object that applied the modifier. Can be used to easily remove all modifiers of a particular source or to simply display information about where a certain modifier originated.