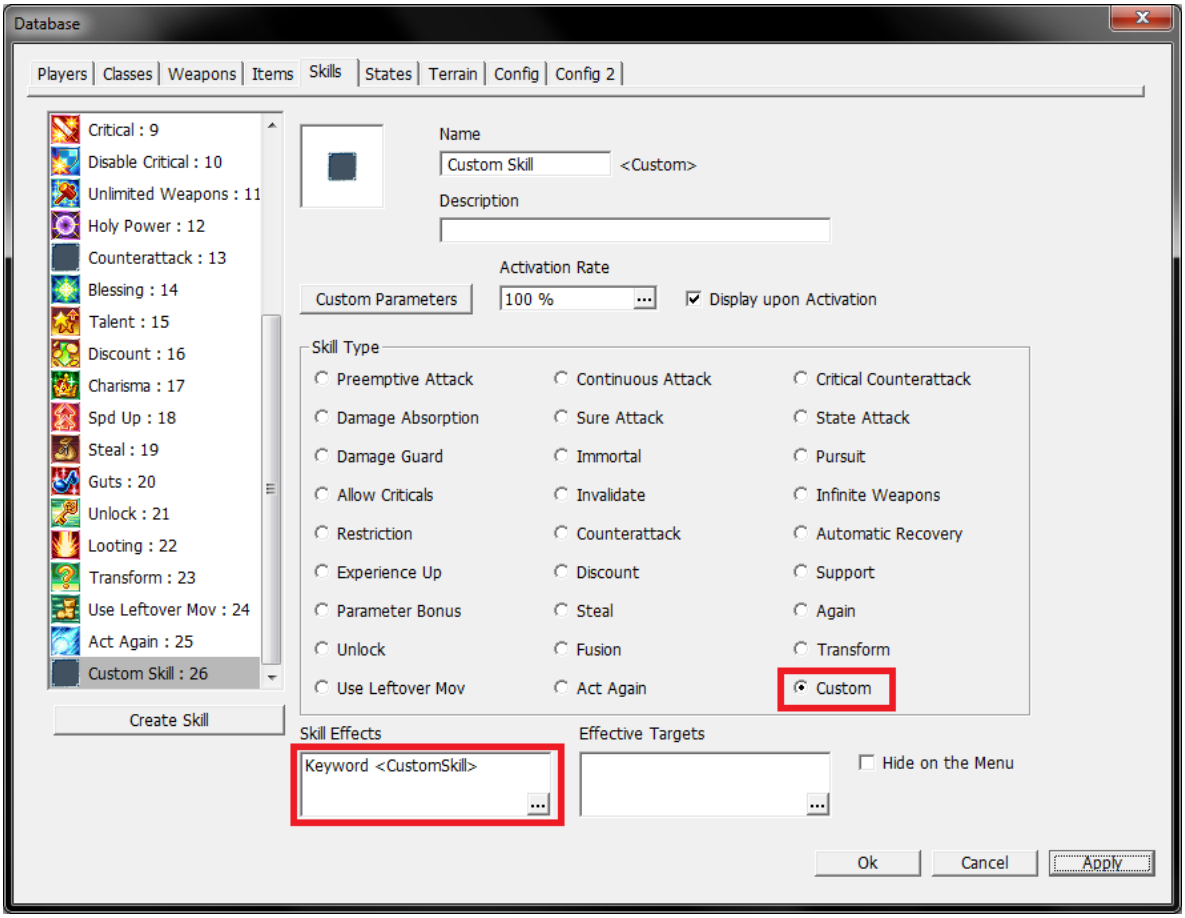
Chapter 2

Custom Skills

By Goinza

This guide will cover how to create plugins that allow the use of custom skills. But before doing that, we need to explain how to create a custom skill using the editor.



As you can see, a custom skill is created like any other common skill. You just need to make sure that the Skill Type is set to “Custom”. Also, you need to add a custom keyword, which will be used to determine what the effect of the skill is.

SkillControl

SkillControl is the name of the object that handles most of the logic about skills. Its main use is to determine if a unit has a skill and if a skill has been triggered during a battle. This object can be found in the file singleton-skillcontrol.js. For the purpose of custom skills, we will only focus on the functions getPossessionCustomSkill and checkAndPushCustomSkill.

Checking if a unit has a skill

The function getPossessionCustomSkill is used to check if a unit has a custom skill with a specific keyword. If the unit has that skill, it returns the skill object, else it returns null. In the next example, we will modify the function calculateRoundCount. This function is used to determine if a unit can make a double attack, based on its speed. We will change it in a way that the unit will always make a double attack if it has the custom skill “AlwaysDouble”.

```
var alias1 = Calculator.calculateRoundCount;
Calculator.calculateRoundCount = function(active, passive, weapon) {
    var rounds = alias1.call(this, active, passive, weapon);
    var skill = SkillControl.getPossessionCustomSkill(active, "AlwaysDouble");
    if (skill!=null) {
        //Unit has the skill
        rounds = 2;
    }

    return rounds
}
```

As you can see, the getPossessionCustomSkill has two parameters: the unit and the skill’s keyword.

Note that, by using this function, the skill won’t “trigger” during a battle. By trigger, we mean that during the battle there won’t be a pop-up to notify that the skill has been activated. This type of skill is always active, both in and out of combat.

Custom parameters on a custom skill

You can use custom parameter to further customize the application of a custom skill. For example, we can make a new skill with keyword “MoreAttacks” that gives additional attacks for each round of combat for the unit. The amount of attacks will be determined by the parameter “attack”. To make this, we will use the function calcualteAttackCount from the file singleton-calculator.js.

```
var alias2 = Calculator.calculateAttackCount;
Calculator.calculateAttackCount = function(active, passive, weapon) {
    var count = alias2.call(this, active, passive, weapon);

    var skill = SkillControl.getPossessionCustomSkill(active, "MoreAttacks");
    //Initialized in zero in case the skill or the parameter are not present
    var extra = 0;
    if (skill!=null) {
        var extra = 0;
        if (skill.custom.attack != null) {
            extra = skill.custom.attack;
        }
    }

    return count + extra;
}
```

Note that is important to check that the custom parameter is not null. The user may have forgotten to add it, or it may be an optional parameter. In either case, it is important to check for null values to avoid crashes in the game.

Checking for multiple skills with the same keyword

So far, we have used the function getPossessionCustomSkill, which returns the first skill object that finds with the given keyword. But if we want to have access to all the skills the unit has with that keyword, we need to use the function getDirectSkillArray. This returns an array of objects, in which each object contains one of the skill objects. For example, let's change how the calculateAttackCount function works by considering that a unit may have more than one instace of the "MoreAttacks" skill.

```
    var alias2 = Calculator.calculateAttackCount;
    Calculator.calculateAttackCount = function(active, passive, weapon) {
        var count = alias2.call(this, active, passive, weapon);

        var skillArr = SkillControl.getDirectSkillArray(active,
SkillType.CUSTOM, "MoreAttacks");
        var extra = 0;
        var skill;
        for (var i=0; i<skillArr; i++) {
            skill = skillArr[i].skill;
            if (skill.custom.attack != null) {
                extra += skill.custom.attack;
            }
        }

        return count + extra;
    }
```

Note that the function has an additional parameter: the skill type. This is because the function can be used to find any type of skill, so we have to specify that we are looking for a custom skill.
Also, it is important to clarify that the function doesn't return an array of skills. It returns an array of objects, where each object contain a skill object. That is why we access the "skill" property in each element of the array.

In conclusion, both functions explained above can be used in any part of the code, as long as you can access the object representing the unit. This allows for a more flexible use of skills, but the downside is that these type of skills are "static", they don't have a random change of triggering. For that, we use another function, explained below.

Triggering Skills

It is possible to make a custom skill that is triggered during battle. The trigger rate can be specified as any other common battle skill. For example, it is possible to make a skill that has a percent chance of dealing additional damage.
To do this, we use the function checkAndPushCustomSkill to determine if a custom skill has been triggered. We also need to modify a specific function of the engine, called isCustomSkillInvokedInternal, which is located in singleton-skillcontrol.js

```
    var alias3 = SkillRandomizer.isCustomSkillInvokedInternal;
    SkillRandomizer.isCustomSkillInvokedInternal = function(active, passive,
skill, keyword) {
        if (keyword === 'ExtraDamage') {
            //Check if activation rate is satisfied.
            return this._isSkillInvokedInternal(active, passive, skill);
        }

        //Default function is called if the skill didn't have the keyword
        return alias3.call(this, active, passive, skill, keyword);
    }

    var alias4 = AttackEvaluator.HitCritical.calculateDamage;
    AttackEvaluator.HitCritical.calculateDamage = function(virtualActive,
virtualPassive, attackEntry) {
        var damage = alias4.call(this, virtualActive, virtualPassive, at-
tackEntry);
        var skill = SkillControl.checkAndPushCustomSkill(virtualActive.unit-
Self, virtualPassive.unitSelf, entry, true, 'ExtraDamage');
        //If skill is triggered, increase damage by 5
        if (skill != null) {
            damage += 5;
        }

        return damage;
    }
```

In this example, we created a custom skill with the keyword “ExtraDamage”. When the skill is triggered, the damage of an attack is increased by 5. Note that we have two functions here: isCustomSkillInvokedInternal, used to make the skill able to trigger, and calculateDamage, which is the function that determines the damage done during an attack. In the last one, we use the function checkAndPushCustomSkill. This function will return the skill object only if the skill has been successfully triggered. In that case, the variable skill is not null and we can increase the damage by 5.

Let’s analyze in more detail how the function checkAndPushCustomSkill works:

```
checkAndPushCustomSkill: function(active, passive, attackEntry, isActive, keyword)
```

The function has 5 parameters. Some of them may be easy to understand: active refers to the attacking unit, passive to the defender, and keyword to the skill’s custom keyword. But there are two parameter that can be confusing, so they will be explained in more detail.

- attackEntry: This is a big object that contains a lot of values related to one instance of an attack during combat. Among its data, it contains things like damage dealt, who is initiating the attack, which skills had been triggered, etc. This entry can be accessed in a few functions, like the calculateDamage

from the example above. If you can't access this attack entry in a function, you can't call checkAndPushCustomSkill, meaning that you can't use trigger skills in that function.

- isActive: A boolean value, it needs to be true if the skill is related to an offensive attack, like dealing more damage, and must be false if it is related to a defensive stance, like receiving less damage.

Keep in mind that these type of cases we can also custom parameters, so instead of being the extra damage always 5, the skill could had a parameter that determines the value of the additional damage.

Sample Code

This guide comes with a sample JavaScript file, named sample-skill.js, which has all the functions that we used as example here. You can use that file as a plugin for you project and test for yourself the code that adds these custom skills. The sample code can also be useful to use as a reference for the creation of your own skills.