RPG Tool – Design Document

By: Sarthak Saxena

# Brief Introduction

The RPG Tool will be designed to help developers create their Role-Playing Game with more flexibility. The tool consists of a range of scriptable object systems including, character creation, weapons creator, monster creation, attack phase, skills creation, and items.

The character creation scriptable can be dragged on the character, from which the developer can easily select health, mana or energy through the Unity editor. The editor will include other object references such as the weapons creator, if the developer wishes to add weapons to the character’s array of weapons in the Unity editor.

Weapons creator will allow the developer to add their own weapon models, name, and use the scriptable object weapon creator on the weapons, to add attributes/stats. These attributes can range from cores stats such as damage, attack speed, de-buffs such as bleed.

Monster creation is a scriptable object that derives from character creation class. It will include multiple stats from character class, and an array of weapons if the developer chooses to add them.

Skills creation is a scriptable object assigned to the characters, or monsters. This includes an array of skills that the developer can add into. The skills will hold attack values, mana or energy cost, an assignable animation, array of de-buffs.

The scriptable object can be attached to item presets. The developer may then add a model, name, modify drop rates and buffs or de-buffs gained when equipped.

Attack is a scriptable object which calculates the damage done between two entities passed into the object by the developer. The calculation returns an output of the entities with the new stats, from which the developer can display the scene accurately.

# Mathematical Equations Used

## Attacking

* Strength (10/ (10 + Defense)) + buffs/de-buffs calculated%.
* Strength + buffs/de-buffs calculated

“Strength” is used as a place holder name for attack damage. The developer may choose to use magic power as a replacement for “Strength”. The buffs will be calculated that effect the attack, then once the attack has been outputted it will either be reduced or increased by the buff/de-buff.

* If the developer wished to add dodge stats to the enemy monsters than the damage done equation becomes

Step 1. Total damage done = Strength (10/ (10 + Defense)) + buffs/de-buffs calculated%)

Step 2. Total damage done = Total damage done - (Total damage done \* dodge multiplier)

* If the developer wished to add accuracy stat to the “player”.   
  Step 1. Accuracy ranges between 1-10, (1 being 0%, 10 being 100%).

Step 2. Generate Random number between 1 and 10  
Step 3. If accuracy equals 25%, if the number generated is between 1-2.5 the attack is successful.

## Buffs & De-buffs

* Total Stat Buff + Related Character Stat = Current Related Stat Value

As an example, a potion gives the character a 10% percent boost to all stats. On the other hand, the player has a de-buff of ‘bleed’ reducing all stats by 5%. Therefore, the potion will only boost up to 5%.

## Status Effects, Armor and Weapon Benefits

* A simple addition and subtraction will be used to add to the stats on character selected by Developer. Stat Buff + Character Stat Max = Character Stat Max.

## Item Drop Rates

* Item Drop Rate Percentage will be set by the Developer. For example, 10% drop rate will be 10 drop per every 100 monsters.

# Advanced Algorithms

## Character Creator

The scriptable object will include a class of character which will have a few base stats such as “LifeUsed”, “ActionCost”, “Damage”, and “Defense”. These stats will have derived stats such as “attack damage”. The editor will have a create stat function.  
The developer can rename the stat to “Energy”, then choose from a range of options such as Life, Action Cost, Hit Cost, Damage, Physical Defense, Magic Defense, etc. As shown in reference image, the developer has added in a value of Energy which they have selected to act as an action cost.

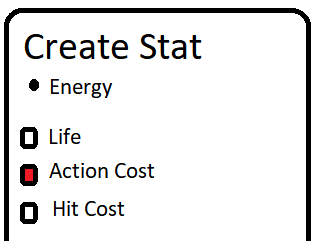


Figure . Custom Stat

The developer may then assign click save to character, which resets the editor and transfers the created stat onto an array of stats at the bottom of the editor. This stat will now behave as an action cost on the player, and will be used in any calculation requiring the action cost stat.

**Managing Stats**

The stats created from the scriptable object will be placed into an array that is displayed to the developer through the editor. From this the developer can assign base values to the stats, and at the end choose to save changes which will go into a reference script of “Player”. The script on the player will contain a class of character stats which will hold stats member variables such as “int LifeUsed, int ActionCost, int Defense and int Armor, int Accuracy, int evasion”. The assigned variables from the editor will have been stored in their appropriate member variables. If the developer doesn’t add “Defense”, the attack calculation will directly output a number without a defense, same with accuracy and evasion.

## Monster/Enemy Creation

The enemy scriptable object can be edited to create an enemy. The scriptable object will take into reference the character script and see the stats being used. It will also contain the same member variables as it will inherit from the character class. Therefore, the enemy will take into account the same stats as well with adjustable values.

## Item/Weapon Creation

The scriptable object of item will allow the developer to edit the droppable items. The script allows the developer to add prefabs of items onto the scriptable object which spawns the items on monster deaths. This item will have adjustable stats when the developer adds in the item prefab and will display options based on the following algorithm.

Step 1. Read character stats for null values.  
Step 2. Take all initialized stats with values.  
Step 3. Display found initialized stats onto editor to be modified.  
Step 4. Select between “Increase/Decrease”, select “Percentage/Integer”, enter value  
Step 5. Select Target options: “Self”, “Ally”, “Target”.  
  
**To Add Stats onto Item/Weapon Prefab**

There will be a preset script from the unity package which will contain a class of string variables for names, float variables for stat, array of stats and method to get stats into item from editor.

Step 1. Add Stats with name, float value from editor and target string onto temporary stat class object.  
Step 2. Add stat object into an existing array of stats.  
Step 3. If target equal self, display if “increase” write “+” before value of stat, and vice versa.  
Step 4. If target equal ally, display if “increase” write “+ on ally” before value of stat, and vice versa.  
Step 5. If target equal target, display “increase” write “+ on target” before value of stat, and vice versa.

## Skills

The array contained on the character will hold many prefab made skills by the developer. These skills can be created through a skill creator scriptable object which takes in a prefab with a preset script named skill, from which the script will run the following algorithm.  
Step 1. Find “Player”, if “Player” display stats onto editor  
Step 2. Display options for “increase”, “decrease”, “Damage Over Time (DOT)” on those stats if increase or decrease selected.  
Step 3. Set Input field for developer to add value for the stat applied onto skill.  
Step 4. Set Target: “self”, “ally” and “target”  
Step 5. Set Skill cost to be ‘used’ value.  
Step 6. If Save button clicked in editor, the script on prefab will hold the stat values.

## Attacking

### Action

Step 1. If Action cost is applied, get the action cost on skill.  
Step 2. Run equation Action Cost – Skill cost.  
Step 3. If action cost greater than zero allow attack phase to run action.  
Step 3. If skill had DOT selected, each iteration of delta time increase or decrease stat by selected value.  
Step 4. Run animation on prefab created by the developer.  
Step 5. Run algorithms for damage below.

### With Defense, Without Accuracy and Evasion

Step 1. Take into account both entities attached by the developer.  
Step 2. Input Damage values into temporary damage variables from both entities.  
Step 3. Check if entity has any extra buff from weapons or items.  
Step 4. Loop through each weapon damage stat, and add to damage variable  
Step 5. Loop through each item stat under type “Defense” and “Damage”, and add to entity damage and defense  
Step 6. Perform steps 3-5 with second entity  
Step 7. If “Defense” as a stat exists use Attack calculation with no accuracy and evasion. Otherwise use Attack calculation without “Defense”.   
Step 8. Output “Life” type stat when put through calculation.  
Step 9. Determine if any entity has “Life” stat under zero  
Step 10. Output “winner” and “loser” entity name, and stats relative to the entity.

### With Accuracy and Evasion

Step 1. Take into account both entities attached by the developer.  
Step 2. Input Damage values into temporary damage variables from both entities.  
Step 3. Check if entity has any extra buff from weapons or items.  
Step 4. Loop through each weapon damage stat, and add to damage variable  
Step 5. Loop through each item stat under type “Damage”, and add to entity damage  
Step 6. Perform steps 3-5 with second entity  
Step 7. If “Defense” as a stat exists use Attack calculation with accuracy and evasion. Otherwise if accuracy or evasion doesn’t exist use relative equations for attack.   
Step 8. Output “Life” type stat when put through calculation.  
Step 9. Determine if any entity has “Life” stat under zero  
Step 10. Output “winner” and “loser” entity name, and stats relative to the entity.

# Modular

The tool is modular as it takes into account factors from each RPG style by placing them under a general category. Stats such as “Constitution” and “Health” cannot used at the same time therefore, will be placed under “Life” to determine life of character. This can then be re-written by the developer to be “Constitution” if they wish to. The same follows with stats such as “Defense”, “Attack Cost”, etc.  
  
Tool will allow the developer to add in multiple skills, and change values of skills and stats flexibly through the unity editor. This allows the tool to be used for each developer for each game as it allows the values to change per game. Values such as accuracy, evasiveness that effect the calculation of damage are added for developers who wish to add in such mechanics to their game, but are not mandatory.

Per each stat, there can’t be a similar one added such as health and constitution. This is in place as the script runs a function which sets the stats of items, weapons and skills to use the initialized stats rather than some other stats. This allows the tool to stay modular, no matter what stat is added, will always include the added custom stats created from the developer.

# How to Integrate into Project

To use the following package simply un-package the tool folder. Attach each script for skill to skills, weapons to weapons, items to items, player to player and enemy to enemy. This will allow the scriptable objects to reference the correct scripts when accessing the relative prefab being used in the methods.