Class System

Table of Contents

Overview

The Class System allows for a user defined upgrade structure for units in both fortress mode and adventure mode. Any unit can be given a class if they meet the requirements. The main features of the system are;.

- Working experience system - Gain experience through killing, using interactions, and reactions. Experience defaults to 1 per creature, but can be changed in the creature itself.
- Class requirements - Restrict classes based on experience, attributes, traits, skills, creature class, caste, and more.
- Class bonuses - Gain/lose attributes, skills, traits, and more based on class level. There are two types of bonuses, one that permanently increases/decreases each level you achieve and one that only lasts until you change classes.
- Class trees - Create complicated class trees by requiring other classes. Multiple other classes can be required.
- Subsystems - Two subsystems accompany the class system to expand on the functionality. The Spell and Feat SubSystems.

Subsystems

Spell SubSystem

This allows for expanded “spell” (i.e. Interaction) usages. It can be loaded in addition to the Class System, but is not required. The advantages for using this SubSystem are;

- Casting Times - You can specify casting times for your spells. Casting times can be shortened or lengthened based on a number of factors including casting skill and equipment.
- Casting Experience - Easily gain experience for each spell cast. Useful for allowing your healers to level up.
- Casting Skill - Gain casting skill each time you cast a spell, just like a miner gains mining skill.
- Casting Exhaustion - Add a configurable amount of exhaustion to a unit whenever it casts a spell
- Spell Cost - Make learning spells cost a certain amount of class experience. This limits the amount of spells a unit can learn to be based on their experience level.

Feat SubSystem

This allows for additional bonuses to be applied to units based on how much experience they have acquired. Every so often, configurable in the base file, a unit will be given a feat point. These points can be spent to earn special user defined feats. It can be loaded in addition to the Class System, but is not required. The advantages for using this SubSystem are;

- Further Customization - Two units of the same class and same level can now be vastly different depending on their feats.
- Class Specials - Making class specific feats gives your units even more room to advance.
- General Feats - General feats can be made so that you can make your healer slightly tougher or your tank slightly faster.
- Restrictions - Feats can be made that are restricted from a certain class or another feat.

Synergies

While the class system can be used on its own, it synergizes well with the other scripts and systems that are included in this collection.

Civilization System

The Civilization System can be used to add or remove certain classes from each entity. As entities send units for sieges or ambushes some of those units will belong to one of the classes that they have access to.

Event System

The Event System can check for certain classes to be present on the map and then only affect those classes (or everyone but those classes)

Start Here

In order to use the Class System there are a number of files you will have to create and place in the systems/Classes folder (note you can place them in the raw/objects folder if you prefer). In addition you will have to create a file for syndromes, but that will be covered later.

The first file you will create is the file necessary for defining what each class is, and all of the bonuses and requirements, along with the spells that class can use. It can be named classes.txt or classes_something.txt. More than one class file is allowed.

classes.txt

This text file will contain all of your defined classes, each following a specific format. The structure of the classes can be broken down into four separate parts, the base, bonuses, requirements, and spells. Here is an example class with all currently supported tokens.

```
[CLASS:SQUIRE]
# Base tokens
    [NAME:squire:squires]
    [EXP:10:20]
    [LEVELS:2]
# Bonus tokens
    [BONUS_PHYS:STRENGTH:50:75:100]
    [BONUS_MENT:WILLPOWER:10:20:30]
    [BONUS_SKILL:AXE:1:2:2]
    [BONUS_TRAIT:ANGER:-5:-5:-5]
    [LEVELING_BONUS:PHYSICAL:AGILITY:25]
    [LEVELING_BONUS:MENTAL:FOCUS:100]
    [LEVELING_BONUS:SKILL:SHIELD:1]
    [LEVELING_BONUS:TRAIT:GREED:-5]
# Requirement tokens
    [REQUIREMENT_PHYS:STRENGTH:1500]
    [REQUIREMENT_MENT:WILLPOWER:1000]
    [REQUIREMENT_SKILL:AXE:4]
    [REQUIREMENT_TRAIT:ANGER:45]
    [REQUIREMENT_CLASS:PEASANT:1]
    [REQUIREMENT_COUNTER:TRAIN:5]
    [FORBIDDEN_CLASS:ACOLYTE:1]
    [REQUIREMENT:CREATURE:DWARF:MALE]
# Spell tokens
    [SPELL:SPELL_TEST_1:0]
        [SPELL_REQUIRE_PHYS:AGILITY:1500]
        [SPELL_REQUIRE_MENT:FOCUS:1500]
        [SPELL_FORBIDDEN_CLASS:ACOLYTE:0]
        [SPELL_FORBIDDEN_SPELL:SOME_OTHER_SPELL]
        [SPELL_COST:100]
        [SPELL_EXP_GAIN:2]
        [SPELL_AUTO_LEARN]
    [SPELL:SPELL_TEST_2:1]
        [SPELL_COST:100]
        [SPELL_UPGRADE:SPELL_TEST_1]
```

Note that if you are using the Spell SubSystem, the only spell tokens that will be read from this file are the starting spell token (i.e. [SPELL:SPELL_TEST_1:0]) and [SPELL_AUTO_LEARN]. Each token is discussed in detail below.

Base tokens - Mandatory tokens for each class

Each class must have these three tokens

- NAME - the name displayed in-game
- LEVELS - number of levels the given class has
- EXP - experience amounts needed for leveling, must have the same number of amounts as levels

Bonus tokens - Tokens which give a unit bonuses based on class level

Each class can have as many or as few of these tokens as wanted

- Non-Permanent Bonuses
  These bonuses are lost when changing class and are not additive. Must have the same number of amounts as levels + 1.
  - BONUS_PHYS - amount of specific physical attribute to add to unit at each level
  - BONUS_MENT - amount of specific mental attribute to add to unit at each level
  - BONUS_SKILL - amount of specific skill to add to unit at each level
  - BONUS_TRAIT - amount of specific trait to add to unit at each level
- Permanent Bonuses
  These bonuses are gained each level and last between class changes.
  - LEVELING_BONUS:PHYS - amount of specific physical attribute to add to unit at each level
  - LEVELING_BONUS:MENT - amount of specific mental attribute to add to unit at each level
  - LEVELING_BONUS:SKILL - amount of specific skill to add to unit at each level
  - LEVELING_BONUS:TRAIT - amount of specific trait to add to unit at each level

Requirement tokens - Tokens which specify certain requirements a unit must meet to be the class

Each class can have as many or as few of these tokens as wanted

- REQUIREMENT_PHYS - amount of specific physical attribute a unit must have to be the class
- REQUIREMENT_MENT - amount of specific mental attribute a unit must have to be the class
- REQUIREMENT_SKILL - amount of specific skill a unit must have to be the class
- REQUIREMENT_TRAIT - amount of specific trait a unit must have to be the class
- REQUIREMENT_COUNTER - amount of specific counter needed for unit to be the class
- REQUIREMENT_CLASS - level of a specific class that the unit must have reached to be the class
- FORBIDDEN_CLASS - level of a specific class that a unit can not have reached to be the class
- REQUIREMENT_CREATURE - the creature:caste combination a unit must be to be the class

Spell tokens - Tokens which define what spells a class can use

Each class can have as many or as few of these tokens as wanted

- SPELL - each spell declaration begins with this token, giving the spell ID and required level. Note that multiple classes can use the same spell by using the SPELL_ID.
- SPELL_REQUIRED_PHYS - amount of specific physical attribute a unit must have to learn the spell
- SPELL_REQUIRED_MENT - amount of specific mental attribute a unit must have to learn the spell
- SPELL_FORBIDDEN_CLASS - level of a specific class that a unit can not have reached to learn the spell
- SPELL_FORBIDDEN_SPELL - a specific spell that the unit must not know to learn the spell
- SPELL_COST - cost associated with learning the spell. Spell costs are covered below
- SPELL_UPGRADE - a specific spell that the new spell replaces. Used in instances where you don't want a unit know two interactions (e.g. replace Fireball with Greater Fireball)
- SPELL_AUTO_LEARN - automatically learn the skill upon advancing to the correct level

Misc tokens - Specialty tokens that don't fall into another category

Currently the only token in this category is;

- AUTO_UPGRADE - this allows the unit to automatically change to a different class on reaching the max level of the current class.

The next files are the necessary files for the Spell SubSystem and Feat SubSystem. These can be ignored if you are not running these systems. These files should be named spells.txt (or spells_something.txt) and feats.txt (or feats_something.txt)

spells.txt

This text file contains the information about each spell. The file itself contains one entry for each SPELL listed in the classes file as well as any other spells you may want to create. Here is an example spell with each of the currently supported tokens.

[SPELL:SPELL_TEST_1]
[NAME:curse]
[DESCRIPTION:Curse a target with lowered toughness and endurance]
[ANNOUNCEMENT:SOURCE_NAME curses TARGET_NAME]
[SKILL_GAIN:SPELL_CASTING:10]
[EXP_GAIN:2]
[CAST_TIME:2]
[CAST_EXHAUSTION:100]
[REQUIREMENT_PHYS:STRENGTH:1000]
[REQUIREMENT_MENT:WILLPOWER:1500]
[FORBIDDEN_CLASS:SQUIRE]
[FORBIDDEN_SPELL:SPELL_TEST_2]
[COST:200]
[SCRIPT:unit/attribute-change -unit SOURCE -attribute [TOUGHNESS ENDURANCE] -type Fixed -amount [\-250 \-250]]

feats.txt

This text file contains details for each feat in the game. Here is an example feat with each of the currently supported tokens.

[FEAT:FEAT_TEST_1]
[NAME:improved strength]
[DESCRIPTION:The unit has improved strength[
[REQUIRED_CLASS:BLAH]
[REQUIRED_FEAT:BLAH]
[FORBIDDEN_CLASS:BLAH[
[FORBIDDEN_FEAT:BLAH[
[COST:1]
[EFFECT:unit/attribute-change -unit SOURCE -attribute STRENGTH -type Fixed -amount 200]

Set Up

In previous editions there was a python script that would read the classes.txt file and output several game files for use. This has been removed due to the adding of the Spell and Feat SubSystems. In the future it may make a return, but for now you will have to set up the game files yourself. Below is a walkthrough of what is needed.

syndromes

You will need to create syndromes for each class, spell, and feat so that the system knows that they actually exist. All of the syndromes can go on one inorganic or multiple, depending on what you want. They must be named the same as the class, spell, or feat. Examples are found below;

Class Syndromes

Class syndromes are used to label each class, here are three example syndromes in an inorganic.

```
[INORGANIC:CLASS_NAME_SYNDROMES]
    [USE_MATERIAL_TEMPLATE:STONE_TEMPLATE]
    [SYNDROME]
        [SYN_NAME:WARRIOR]
        [SYN_CLASS:CLASS_NAME]
        [CE_DISPLAY_NAME:NAME:warrior:warriors:warrior:START:0]
    [SYNDROME]
        [SYN_NAME:HEALER]
        [SYN_CLASS:CLASS_NAME]
        [CE_DISPLAY_NAME:NAME:priest:priests:priest:START:0]
    [SYNDROME]
        [SYN_NAME:MAGE]
        [SYN_CLASS:CLASS_NAME]
        [CE_DISPLAY_NAME:NAME:mage:mages:mage:START:0]
```

Note that for class syndromes you can also add any other syndrome effects you would want, for example changing body size and such. These are just the minimum required features. (You can also remove the [CE_DISPLAY_NAME] if you don't want your class names to be displayed in game)

Spell Syndromes

Spell syndromes are used in order to teach units "spells" (i.e. interactions). They are more complicated than class syndromes since they need to have the entire CDI syndrome tokens in them. Here are two example syndromes;

```
[INORGANIC:SPELL_LEARN_SYNDROMES]
    [USE_MATERIAL_TEMPLATE:STONE_TEMPLATE]
    [SYNDROME]
        [SYN_NAME:RALLY]
        [SYN_CLASS:SPELL_LEARN]
        [CE_CAN_DO_INTERACTION:START:0]
            [CDI:INTERACTION:RALLY]
            [CDI:ADV_NAME:rallying cry]
            [CDI:TARGET:A:SELF_ONLY]
            [CDI:USAGE_HINT:FLEEING]
            [CDI:VERB:cast rallying cry:casts rallying cry:rallying cry]
            [CDI:MAX_TARGET_NUMBER:A:1]
            [CDI:WAIT_PERIOD:500]
    [SYNDROME]
        [SYN_NAME:BATTLE_SHOUT]
        [SYN_CLASS:SPELL_LEARN]
        [CE_CAN_DO_INTERACTION:START:0]
            [CDI:INTERACTION:BATTLE_SHOUT]
            [CDI:ADV_NAME:battle shout]
            [CDI:TARGET:A:LINE_OF_SIGHT]
            [CDI:TARGET_RANGE:A:20]
            [CDI:USAGE_HINT:ATTACK]
            [CDI:VERB:cast battle shout:casts battle shout:battle shout]
```

[CDI:MAX_TARGET_NUMBER:A:1]
[CDI:WAIT_PERIOD:500]

Feat Syndromes

Feat syndromes are not currently required or implemented.

interactions

In order to use the spells you need to create interactions for them. Below are the two interactions for the above spell syndromes;

```
[INTERACTION:RALLY]
    [I_SOURCE:CREATURE_ACTION]
    [I_TARGET:A:CREATURE]
        [IT_LOCATION:CONTEXT_CREATURE]
        [IT_MANUAL_INPUT:creature]
    [I_EFFECT:ADD_SYNDROME]
        [IE_TARGET:A]
        [IE_IMMEDIATE]
        [SYNDROME]
            [CE_SPEED_CHANGE:SPEED_ADD:1:START:0:END:1]
[INTERACTION:BATTLE_SHOUT]
    [I_SOURCE:CREATURE_ACTION]
    [I_TARGET:A:CREATURE]
        [IT_LOCATION:CONTEXT_CREATURE]
        [IT_MANUAL_INPUT:creature]
    [I_EFFECT:ADD_SYNDROME]
        [IE_TARGET:A]
        [IE_IMMEDIATE]
        [SYNDROME]
            [CE_SPEED_CHANGE:SPEED_ADD:1:START:0:END:1]
```

Note that both of these interactions don't actually do anything. That is because the actual effects are carried out by the Spell SubSystem and are kept in the spells.txt [SCRIPT] token. These interactions are merely used to interface with the modtools/interaction-trigger script.

reactions

In order to change classes, learn spells, and add feats you will need to create reactions and tie them into the modtools/reaction-trigger script. The scripts you will need are classes/add-feat, classes/learn-spell, and classes/change-class. Simply create a reaction and then add this to onload.init

modtools/reaction-trigger -reactionName SOME_REACTION -command [classes/change-class -unit \\WORKER_ID -class SOME_CLASS_NAME]

This will then change the class of the unit that runs the reaction to SOME_CLASS_NAME.

onload.init

In addition to the reactions you will need to add to onload.init you will also have to add the interactions. There are two ways to do this.

1. Without using the Spell SubSystem

   When not using the Spell SubSystem you will need to put all of the information for the spell in the onload.init. This means each spell will have an entry and will look something like this,

   modtools/interaction-trigger -onAttackStr "casts rallying cry" -command [what you want the spell to do goes here]

2. With using the Spell SubSystem

   If you are using the Spell SubSystem you already defined what you want the spell to do, in that case you simply need to link it using the same method as above but changed so that you are calling spells/trigger

   modtools/interaction-trigger -onAttackStr "casts rallying cry" -command [spells/trigger -source \\ATTACKER_ID -target \\DEFENDER_ID -spell RALLY]

Note that it may seem like using the Spell SubSystem doesn't give any advantages, but the -command entry when not using it can be very very long. This just simplifies the onload.init file

Final Step

The last component necessary is to initialize the class system in onLoad.init. To do this, simply copy

base/roses-init -classSystem [Spells Feats]

and paste it in onLoad.init.

Note 1: If you are using any other systems in addition to the class system you only need one call of base/roses-init. Just add any additional systems by taking the - argument.

Note 2: If you just want the class system without the subsystems you only need -classSystem. If you want just the spells or just the feats subsystem simply remove the one you don't want.

Note 3: You can add -forceReload to the line if you have changed the details of any of your classes, spells, or feats and it will reload them into the game. If you do not have this line it will not read the files again after a load.

Additional Information

Addendum 1: Experience System

By default the game awards 1 experience point for each kill, whether it be a turtle or a dragon, to address this issue there are several avenues a modder can take.

Increase Experience Values

You can change the amount awarded by each creature by adding [CREATURE_CLASS:EXPERIENCE_X], where X is some positive integer, to the creatures raws. Note that leaving this out of a creature is equivalent to putting in [CREATURE_CLASS:EXPERIENCE_1].

Share Experience Gains

By default, only the last attacker will get the experience for a kill. You can change this to instead give the experience to each friendly unit within a specified radius. In order to do this you will need to open systems/base.txt. At the top of the file you will see [EXPERIENCE_RADIUS:-1]. Changing this number to a different positive integer will increase the radius for which to award experience.

Gain Experience from Spells (without using the Spell SubSystem)

You can assign experience gains to specific spells, allowing, for example, a healer to gain experience through using heals. In order to accomplish this you will need to copy

> *modtools/interaction-trigger -onAttackStr 'YOUR_CDI:VERB_HERE' -command [classes/add-experience -unit \\ATTACKER_ID -amount X]*

into your onLoad.init (replacing YOUR_CDI:VERB_HERE' and X with the appropriate values. These gains are specific to the unit performing the interaction and are not shared amongst others, even if the radius is increased. If you are using the Spell SubSystem, the experience gained is declared in the spell itself.

Gain Experience from Reactions

Experience can be gained through reactions by placing

> *modtools/reaction-trigger -reaction 'YOUR_REACTION_HERE' -command [classes/add-experience -unit \\WORKER_ID -amount X]*

into your onLoad.init, and every time you run the given reaction, you will gain X experience for your current class.These gains are specific to the unit running the reaction and are not shared amongst others, even if the radius is increased.

Addendum 2: Spells

Learning Spells

Spells are learned in one of three ways.

1. Automatically - spells with SPELL_AUTO_LEARN means that a unit will automatically acquire the spell when reaching the spells required class level.
2. Command Line - a user can force a unit to learn a spell by using the command
   classes/learn-skill -unit UNIT_ID -spell SPELL_ID
   The addition of the token -override to the command will mean that the requirements for the spell are skipped
3. Reaction - you can also place the above command line into a reaction for a creature to learn.

Spell Cost

By default it costs nothing to learn a spell (other than the reagents you may have used in the reaction). This can be changed by adding the COST token to a spell. Anyone that learns this spell will have to pay the corresponding spell cost in experience. This experience is stored separately from class and global experience and so does not affect a units leveling. Skill experience is not shared across classes, note that this is a change from previous versions of this system. An example:

Unit becomes class Squire
Unit kills 20 experience worth of creatures

Unit now has 20 class experience, 20 global experience and 20 skill experience
Unit learns a spell that costs 20 skill points
Unit now has 20 class experience, 20 global experience, and 0 skill experience
Unit then changes to class Warrior
Unit kills 10 experience worth of creatures
Unit now has 10 class experience, 30 global experience, and 10 skill experience

Changing Classes

When a unit changes classes they will “forget” all the spells associated with their previous class. And “learn” all the spells associated with their new class that they have already learned. What this means in gameplay terms is that when a unit switches classes, they will lose the ability to do the interactions associated with their previous class. However, if they switch back to that same class they will automatically reacquire the ability to do the interactions. If any of the spells are shared between the two classes, and they meet the requirements, they will keep those interactions.