Black & White 2
New Miracles:
Adding Custom Miracles

Created By: Bill
Date: February 2 2020

Version	Changes	Date
0	Initial Release	February 2 2020

Table of Contents

Introduction

One of the requirements laid out for the New Miracle project was to design it in such a way as to allow others to easily add their own custom miracles. The Totem would act as intermediary between the player and the miracles. It detects when a miracle is clicked, how the player uses the miracle, calculating power for the miracles, moving the miracles as towns migrate.

This document describes the process of how to create a new miracle and add it to the project. What needs to be edited in the Template file, adding in new constants, and editing the communication scripts. However coding a new miracle's effects when activated will not be covered.

Editing the Template

In the package should exist a file called *NMs_TEMPLATE.txt*, this file is the miracle template. It possess the skeleton of a miracle. To create a new miracle starting with this template it is recommended that you make a copy of this template and rename it. It is best to keep the standard prefix for the file "NMs_" followed by the name of the miracle. For example, "NMs_Custom.txt".

The first 7 lines is the header. It would be best if the "Purpose" was updated to provide a brief description of what your miracle will do.

The next step is to update the variable names, constants, and script names contained within the file. You'll notice the constant *NMsTEMP_MAXACTIVE*, the variables like *NMsTemp_Count* and the script names like *NMsTemp_CastPour*. They all contain a variation of the word "temp" after the prefix "NMs". While "NMs" is the over all prefix for the entire project, a short form of the miracle is tacked on. This is similar to all the other miracles provided in the package. The prefixes for the other miracles are: wheat, NMsWheat_; forest, NMsFrst_; ore, NMsOre_; lava, NMsLava_ etc...

The word you tack onto the project prefix isn't important, what is, is that it's constant throughout the file. The best way to update all the names is by doing a search and replace. Not all editors provide a feature like this if yours does not I'd recommend Notepad++ or Geany.

Notice that the constant's prefix is all in caps, this isn't entirely necessary, however it would be more consistent with everything else.

Now that all the naming is customized to your miracle, it is time to begin editing the scripts.

Create Symbolic Bubble

The first script of the file is responsible for creating the bubble at the Totem. The script takes the following parameters:

- idx – The index the miracle is to be store in the array *NMs_Miracles*.
- xPos – The x co-ordinate of the miracle.
- yPos – the y co-ordinate relative to ground height. As you'll notice the ground height is calculated and added onto this value passed in.
- zPos – the z co-ordinate of the miracle.
- t_idx – The index of the Totem the miracle is being added to.

The variable *NMs_Miracles* is mentioned throughout this entire script. The variable is a 2 dimensional array which is used to keep track of the miracles. The structure of the array is important to understand as it limits what you can do with your symbolic bubble design. The first index is the id of a miracle, the second index references objects, visual, or other important information about the miracle.

1. Symbol – This is the object contained within the bubble, for example wheat uses the hay bale. It is important that this object is large, as the Totem looks for when the player clicks the symbol to begin the activation process for this miracle.
2. Type – What type of miracle this is, which we will get to later when discussing constants.
3. Totem – Id of the Totem this miracle is being added to.
4. Object 1 – The first visual effect.
5. Object 2 – The second visual effect.
6. Visible – 1 = true, 0 = false.
7. Mana – Stores how much mana is required to cast the miracle.

The symbol is best kept as a hay bale then use the override mesh command to swap it to another object. As see in the following code:

```
NMs_Miracles[calIdx + NMs_MIRSYMBOL] = create with angle 0 and scale 0.25
SCRIPT_OBJECT_TYPE_FEATURE FEATURE_INFO_AZTC_SUNTEMPLE at {xPos,yPos,zPos}

override mesh for NMs_Miracles[calIdx + NMs_MIRSYMBOL] with "..\models\m_tree_oak"
SCRIPT_OBJECT_PROPERTY_TYPE_YPOS of NMs_Miracles[calIdx + NMs_MIRSYMBOL] = yPos
```

All the default miracles that come in the package use the Aztec Sun Temple, then override with another model. In the above case it's an oak tree. The mesh of an object can be overridden with any *bwm* file located in *Black & White 2\Data\Art*.

The reasoning behind this method is that the Aztec Sun Temple allows it's y co-ordinate to be edited. Other objects won't accept a y value and fall to the ground.

In the structure of the miracle array, notice how there are only two spots for visual effects. This means you are limited to only two visual effects for your symbolic bubble. All the visual effect constants can be found in the file *VisualEffectTypeEnum.h*.

The forest miracle uses VISUAL_MIRACLE_TORNODO_SEED and VISUAL_EFFECT_HAND for it's look. When picking visuals the first one should be a seed effect because it provides the bubble look.

Following the visual effect are the property edits to the visuals. Here you may do whatever you wish to the visual effects, like change their scale, colour, strength, speed, etc...

Create Miracle in Hand

The second script is responsible for creating a representation of the miracle in the player's hand. The edits needed for this script are simple, you need only to replicate the look of the bubble here. However the first visual, generally the bubble, may need to be replaced with a visual ending with IN_HAND.

Generally the object in hand uses the mobile object, hay bale. Even though its mesh is overridden most of the time it must be a mobile object unlike the symbolic bubble which is a feature.

Calculate Mana
This script is responsible for calculating the required mana to cast the miracle. It is run periodically by the Totem scripts.

Here you need to add in your formula for how much mana your miracle requires. The power level of the Totem is passed in as the variable *pwr*.

Cast Pour
Now we get to the interesting parts. These next two scripts are where the real meat of your miracle is done. The first of which is the pouring script. Here you must program what happens when the player is holding down the action button. Now the template provides the basics of what this script needs to do.

Some basic setup is done, then there is the power calculation area. Here is where, using the *pwr* parameter, the overall effect is calculated. After that there is some more initialization, then we come to the main loop.

The main loop contains all the code for keeping the miracle in the player's hand, checking for cancellation, making sure the action button is being held down, and that the player's hand is still in his influence.

Generally pouring allows the player to spread the effect of the miracle wherever he can reach rather than throwing it, which results in a single effect at the contact point. For this reason a timer was coded into the loop, where each iteration of the miracle's effect occurs when the timer hits 0. The amount of time is under your control, the default is 0.3s.

The variable *amt* is used to track how much of the miracle's overall effect has been expelled. It is incremented by the variable *t_PerTick* every iteration. The loop finally ends when either the cancel action is performed or *amt* becomes larger or equal to *t_Amount*.

Once the loop is over any visuals or objects that need to be delete must be done.

Cast Throw
Similar to the pour script the first chunk of this script is dedicated to power calculations to determine the extent of the miracle's effect.

After power calculations the script waits for the miracle to hit something, or for a 60 second timer to run out. The timer is there just in case something gets messed up and the miracle doesn't end up hitting anything for whatever reason. It would bad for the script to remain at the wait until command forever taking up processing time.

You as the programmer must write the effect of the miracle in the if statement at the bottom.

Miracle Integration

When your custom miracle is ready Totem and Global files need updating, so that those scripts are aware of your miracle object. There are two steps (1) add and update the needed constants, and (2) editing the communication scripts.

The custom miracle file must also be added to the challenge file.

Constants

The first constant, located in *NMs_Globals.txt*, that requires updating is *NMs_MIRACLETYPES*. By default in the package the value is set to 6 which is equal to the number of miracles provided. Since more miracles are being added to the system the value of this constant must be incremented.

NMs_MIRACLETYPES is used for setting up a 2D array located in *NMs_Totem.txt*. Thus it is necessary to update the number of elements in that array to make room for our new miracle. The array in question is:

```
global NMs_TotemMiracle[NMs_TOTEMDEF2]
```

This array counts the number of each miracle belonging to the Totems. The number of array elements is defined by the constant *NMs_TOTEMDEF2* which is initialized on line 49. The formula for how many elements are required for this constant is:

$$NMs\_TOTEMDEF2 = NMs\_NUMBERTOTEM \times NMs\_MIRACLETYPES$$

Next, a new constant needs to be added in for your miracle. In *NMs_Globals.txt* there is a series of constants for the different miracle types; they can be found under the comment "//Miracle Types" around line 31. Here you need to add a new constant for your miracle type like this example:

```
//Miracle Types
define NMs_NONE          = 0
define NMs_WHEAT         = 1
define NMs_ANIMALS       = 2
define NMs_FOREST        = 3
define NMs_ORE           = 4
define NMs_LIGHT         = 5
define NMs_LAVA          = 6
define NMs_CUSTOM        = 7
```

Each time a new miracle is added on its value must be equal to the previous miracle's value plus 1. For example with the example above, if I were to add another miracle it's value would be 8.

Communication Scripts

In *NMs_Totem.txt* there is a series of scripts referred to as "communication scripts." These scripts allow the Totem control scripts to call required scripts that belong to the miracles. Thus it is necessary to add in the needed code to these communication scripts, so that the Totem control scripts can access your miracle. Don't worry the code is simple.

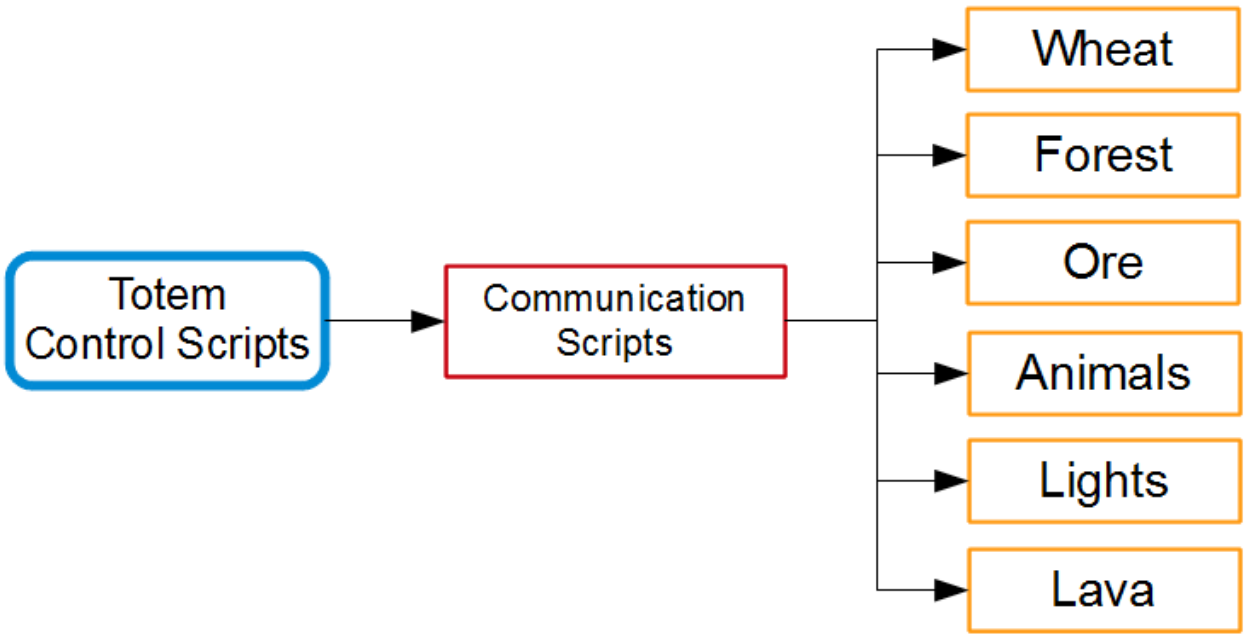

Illustration 1: Block diagram of internal script calls

All the communication script have the same structure. They take the needed information, and using an if statement, it runs the correct miracle script. For example the first communication script is *NMs_CallCreateFunction*. Each if statement has a condition which checks the miracle type, then it runs the script and increments that miracle's internal counter.

```
if(m_type == NMs_WHEAT)
    run script NMsWheat_createSymbolicBubble(idx,xPos,yPos,zPos,t_idx)
    NMsWheat_Count++
```

Thus all that needs to be done to integrate a new miracle is an additional elsif to check for the new miracle. This is the code needed, you'll have to replace the bold parts.

```
elsif(m_type == <Miracle Constant>)
    run script <Miracle create symbolic bubble script>(idx,xPos,yPos,zPos,t_idx)
    <Miracle internal counter>++
```

<Miracle Constant> refers to the constant created in the previous step, for example, NMs_CUSTOM.
<Miracle create symbolic bubble script> is the first script of the miracle file as described in the first section "Editing the Template."
<Miracle internal counter> was created when doing the search and replace when editing the template for the miracle file. It should be something like NMs<Prefix>_Count.

This is the process for all the communication scripts. Below, all the scripts are listed with the basic elsif statement needed to integrate the new miracle.

NMs_MoveMiracleFunction

```
elsif(m_type == <Miracle Constant>)
```

run script **<Miracle create symbolic bubble script>**(m_idx,xPos,yPos,zPos,t_idx)

NMs_CallHandCreateFunction
elsif(m_type == **<Miracle Constant>**)
run script **<Miracle create in hand representation>**
NMs_HandIdxResult = **<NMs<Miracle Prefix>_HandResult>**

NMs_CallHandDeleteFunction
elsif(m_type == **<Miracle Constant>**)
run script **<Miracle delete in hand representation>**(NMs_HandIdxResult)

NMs_CallPourFunction
elsif(m_type == **<Miracle Constant>**)
set **<NMs<Miracle Prefix>_Hand>**[NMs_HandIdxResult] in player 0 hand
run script **<Miracle cast pour script>**(pwr,NMs_HandIdxResult)

NMs_CallThrowFunction
elsif(m_type == **<Miracle Constant>**)
run script **<Miracle cast throw script>**(pwr,NMs_HandIdxResult)

NMs_CallDeleteFunction
elsif(m_type == **<Miracle Constant>**)
run script **<Miracle hide symbolic bubble>**(m_idx)

NMs_CalManaFunction
elsif(m_type == **<Miracle Constant>**)
run script **<Miracle calculate mana>**(m_idx,NMs_Totems[t_ele + NMs_TOTPWR])

At this point the integration is complete, the code should compile. Be sure it add some code to add your new miracle to a Totem before you test it.