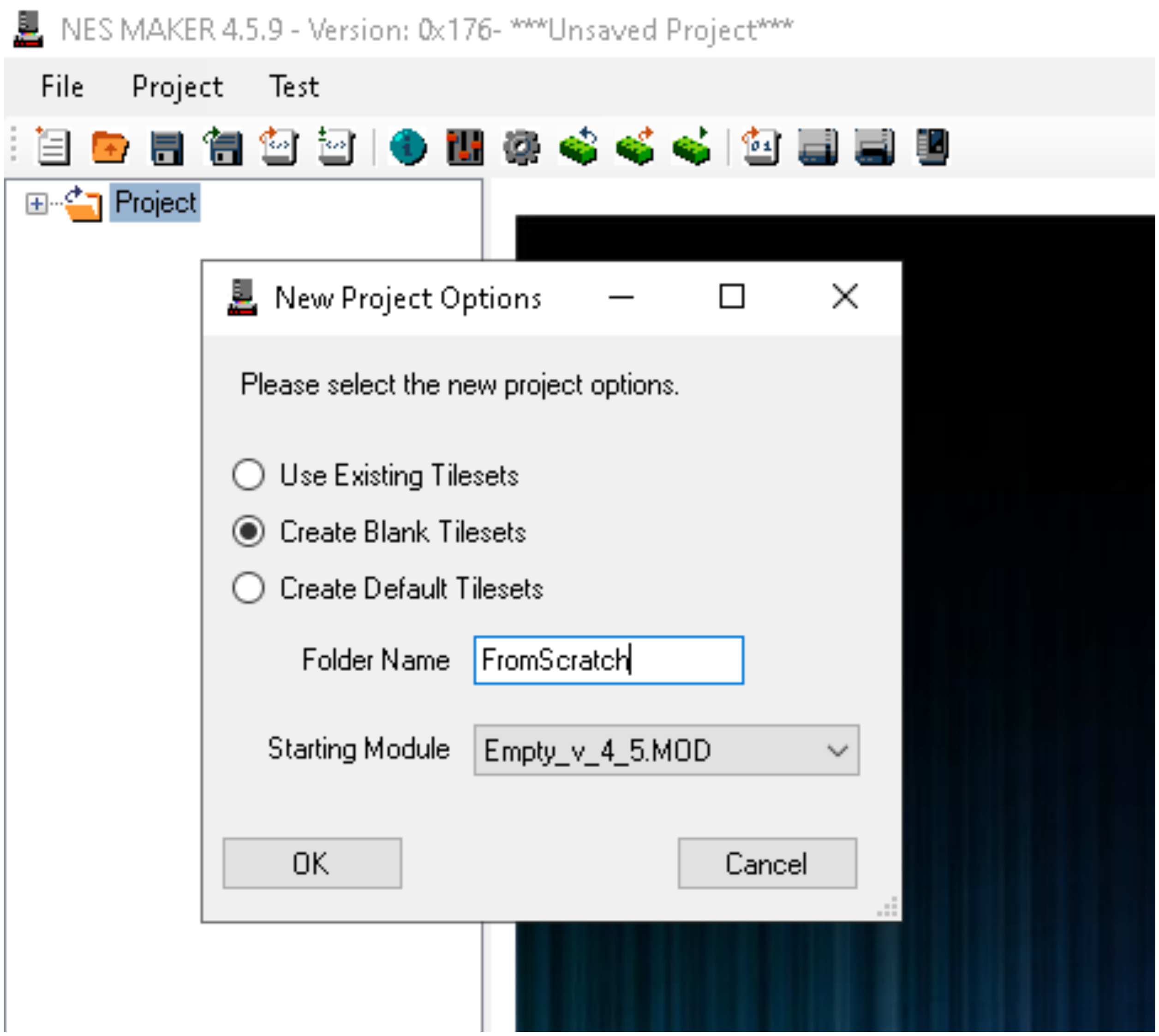
Step 1: Create an empty project.
Start a new project, use Create Blank Tilesets, and call the graphics folder FromScratch. Choose the Empty module from the drop down and press ok.

Run the ROM to make sure it loads, but *nothing* will load. Look around the emulator, especially at the PPU. See that there is nothing, or junk data, there. This is starting from near zero, with just a emaciated backbone in place to allow it to assemble.



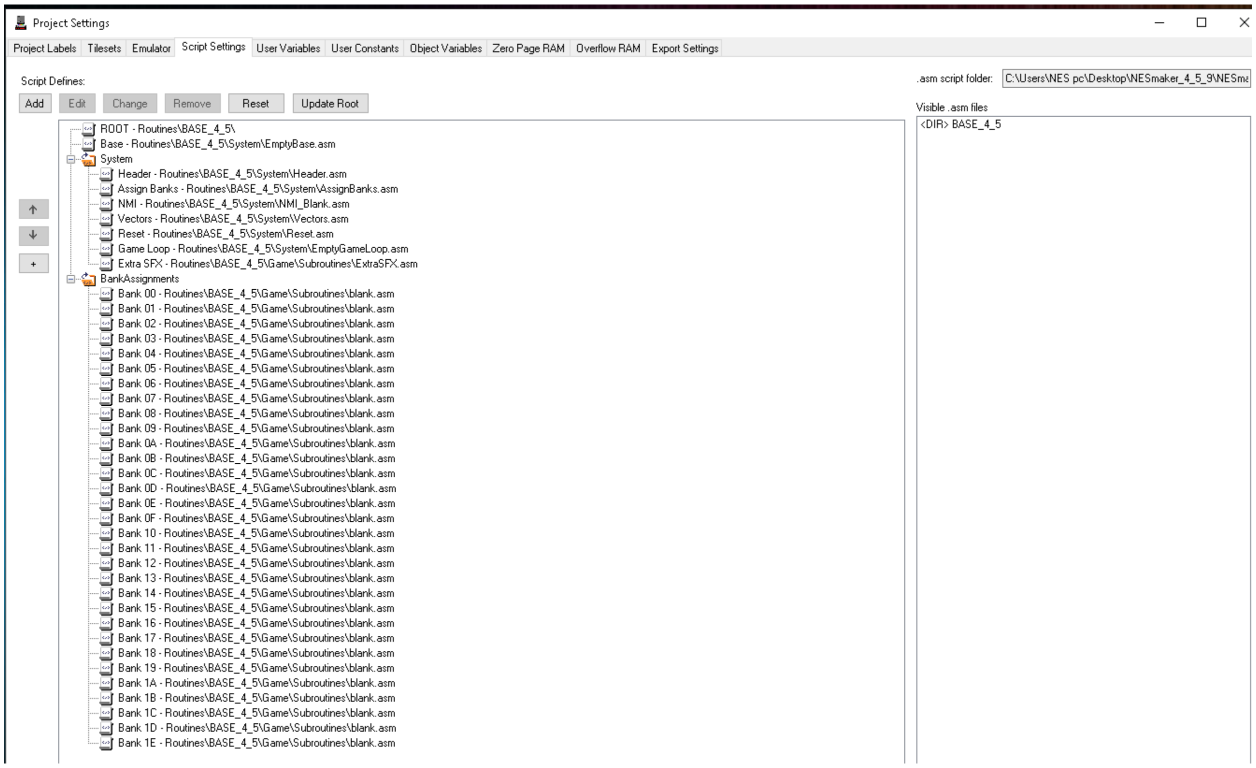
Please select the new project options.

Use Existing Tilesets [✓ Create Blank Tilesets] Create Default Tilesets

Folder Name: FromScratch

Starting Module: Empty_v_4_5.MOD

Really, our project is centered around our BASE script. Right now, if we go to project settings and look at our Script Settings, we can see that there are a few things included. This is our entire game thus far. There is a header, which is a bunch of settings that let an emulator know what to do with the rom. Then there is one variable that is necessary to make this skeleton code work. Then there is a bank assignment scripts, which just assigns empty scripts to all bank areas. Then there is the RESET function, which is what happens when a RESET is triggered. Then an empty game loop, an empty NMI, and the Vectors which are necessary to have set up for our empty project to function. There’s really not much here.Effectively,this is any empty project.This is starting from scratch.It has just enough there to clear out our memory, give us a few hook, and allow us to assemble and play a rom without a crash.



Script Defines:

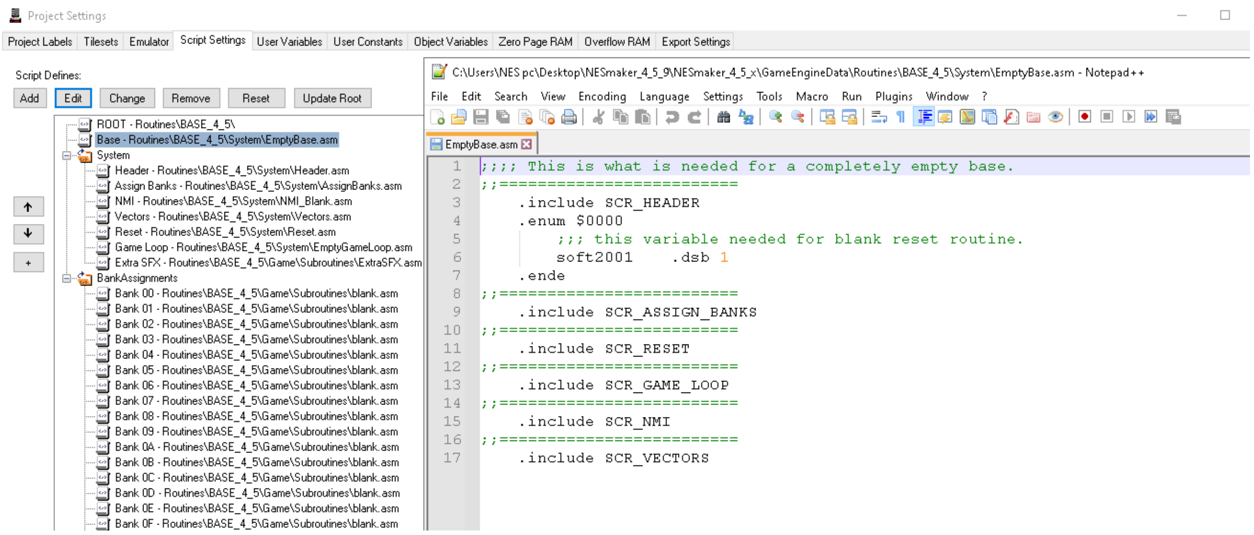
.asm script folder: C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESma

Visible .asm files

<DIR> BASE_4_5

```
ROOT - Routines\BASE_4_5\
Base - Routines\BASE_4_5\System\EmptyBase.asm
System
    Header - Routines\BASE_4_5\System\Header.asm
    Assign Banks - Routines\BASE_4_5\System\AssignBanks.asm
    NMI - Routines\BASE_4_5\System\NMI_Blank.asm
    Vectors - Routines\BASE_4_5\System\Vectors.asm
    Reset - Routines\BASE_4_5\System\Reset.asm
    Game Loop - Routines\BASE_4_5\System\EmptyGameLoop.asm
    Extra SFX - Routines\BASE_4_5\Game\Subroutines\ExtraSFX.asm
BankAssignments
    Bank 00 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 01 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 02 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 03 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 04 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 05 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 06 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 07 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 08 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 09 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0A - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0B - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0C - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0D - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0E - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0F - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 10 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 11 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 12 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 13 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 14 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 15 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 16 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 17 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 18 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 19 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 1A - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 1B - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 1C - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 1D - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 1E - Routines\BASE_4_5\Game\Subroutines\blank.asm
```

Opening the Base script define will show that it is a handful of includes, most of which are currently blank scripts.



```
ROOT - Routines\BASE_4_5\
Base - Routines\BASE_4_5\System\EmptyBase.asm
System
    Header - Routines\BASE_4_5\System\Header.asm
    Assign Banks - Routines\BASE_4_5\System\AssignBanks.asm
    NMI - Routines\BASE_4_5\System\NMI_Blank.asm
    Vectors - Routines\BASE_4_5\System\Vectors.asm
    Reset - Routines\BASE_4_5\System\Reset.asm
    Game Loop - Routines\BASE_4_5\System\EmptyGameLoop.asm
    Extra SFX - Routines\BASE_4_5\Game\Subroutines\ExtraSFX.asm
BankAssignments
    Bank 00 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 01 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 02 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 03 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 04 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 05 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 06 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 07 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 08 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 09 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0A - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0B - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0C - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0D - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0E - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0F - Routines\BASE_4_5\Game\Subroutines\blank.asm
```

C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\EmptyBase.asm - Notepad++

```
;;;; This is what is needed for a completely empty base.
;;=========================
    .include SCR_HEADER
    .enum $0000
        ;;; this variable needed for blank reset routine.
        soft2001    .dsb 1
    .ende
;;=========================
    .include SCR_ASSIGN_BANKS
;;=========================
    .include SCR_RESET
;;=========================
    .include SCR_GAME_LOOP
;;=========================
    .include SCR_NMI
;;=========================
    .include SCR_VECTORS
```

STEP 2: Creating a RAM MAP

The next thing I suggest doing is creating a RAM map. NES games have 2kb of internal RAM at addresses $0000-$07FF. A lot of times with NESmaker, we don't think about memory addresses so much. We think about these RAM slots as variables. We might make a variable called PosX to define an X coordinate. Really, that variable name PosX is just a label that defines a shorthand for a memory address. So maybe I place PosX at the third byte of RAM, which would be address $0002. When I create a variable like PosX, I am setting a common name I can remember to be associated with $0002. At that point, reading or writing to $0002 or reading or writing to PosX would mean exactly the same thing.

If that's too abstract, think about memory addresses like a wall of PO Boxes at the post office. Each of those boxes has some number assigned to them, but each also belongs to an individual with a name. When someone wants to establish a PO Box, the post office associates that PO Box with their name. So Mr. Smith's PO Box is box #1139. Whether I'm getting mail from Mr. Smith's PO Box or PO Box #1139, it means the same thing. The box numbered #1139 is always there no matter what I name it. But it's easier to remember whose box is whose if I put a name on the box. That's a bit like how memory addresses work. The memory addresses $0000-$07ff are always present whether we assign names to them or not. But we'll give labels to those memory addresses, and aligning with the parlance of modern game development, we'll think of these labels as variables and the byte stored in those memory locations as the value of the variable.

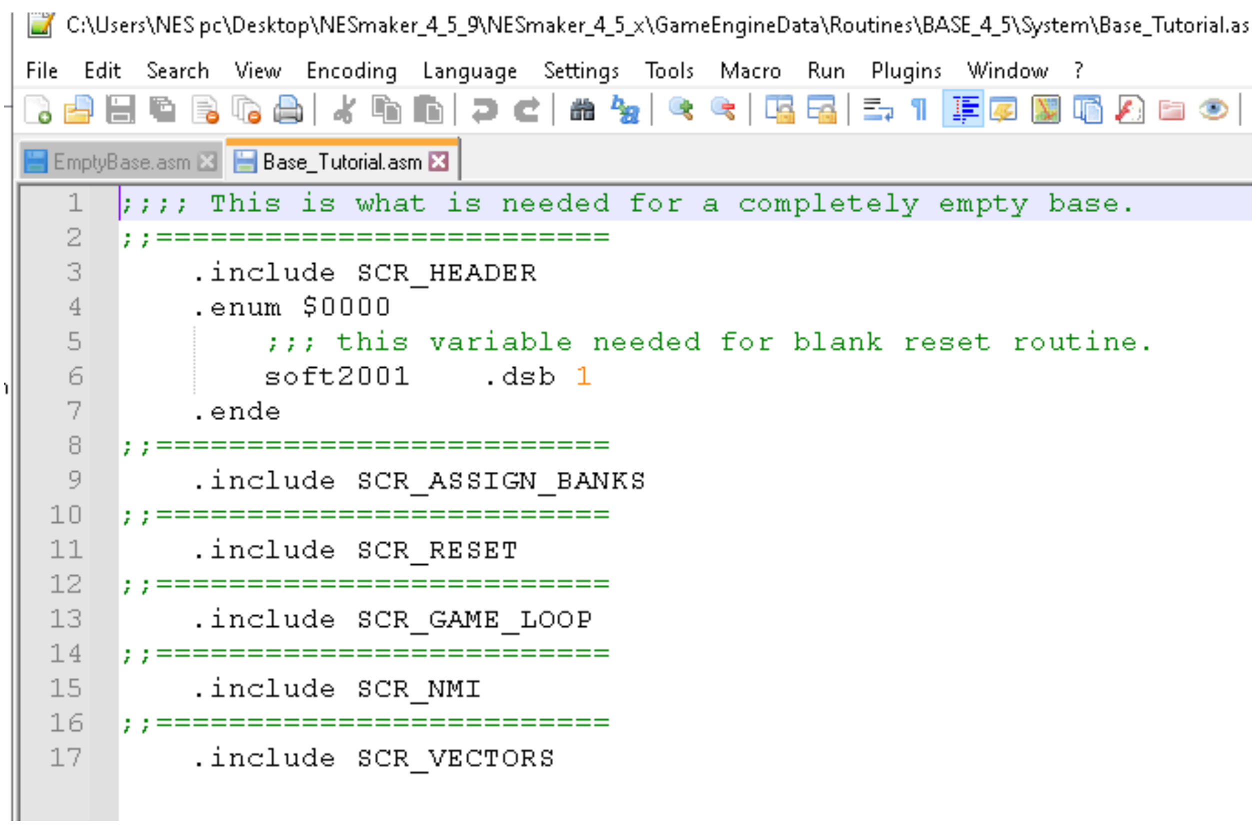
Creating a memory map is the first step at assigning some names to these proverbial mailboxes. Some of these memory locations are already predefined. Some can be used by the developer in any way that makes sense. If you were to load up different NESmaker modules, you might find that memory maps differ. A scrolling game may have more space reserved for collision data, as it may be important to always have two full screens worth of collision data loaded into RAM at a time, where this would not be important for a screen-by-screen adventure game. To get up and running, we're going to create a rather vanilla memory map.

```
;;;; 0000 - 00ff = zero page - this is already determined by the 6502 architecture
;;;; 0100 - 01ff = the stack - this is already determined by the 6502 architecture
;;;; 0200 - 02ff = sprite ram
;;;; 0300 - 03ff = collision ram
;;;; 0400 - 05ff = OBJECT DATA *Note, two pages worth, 512 values total
;;;; 0600 - 06ff = extra user variables
;;;; 0700 - 07ff = currently undefined
```

Now, in order to set this up, we will need to place it in a script. Basically, we need to place it in the main body of our game at the very top, before we hit our main game loop. If you were writing this all out in a massive text file to compile, this would appear near the top of the code. But we're not using a massive text file. Instead, we're using an asset organizer that allows us to replace small bits of code without having to scour through a massive text file. So here, we're going to learn how to iterate a script, re-set a script define, create a new script define, write a simple script for our memory map, and make sure our code includes this script in the appropriate place.

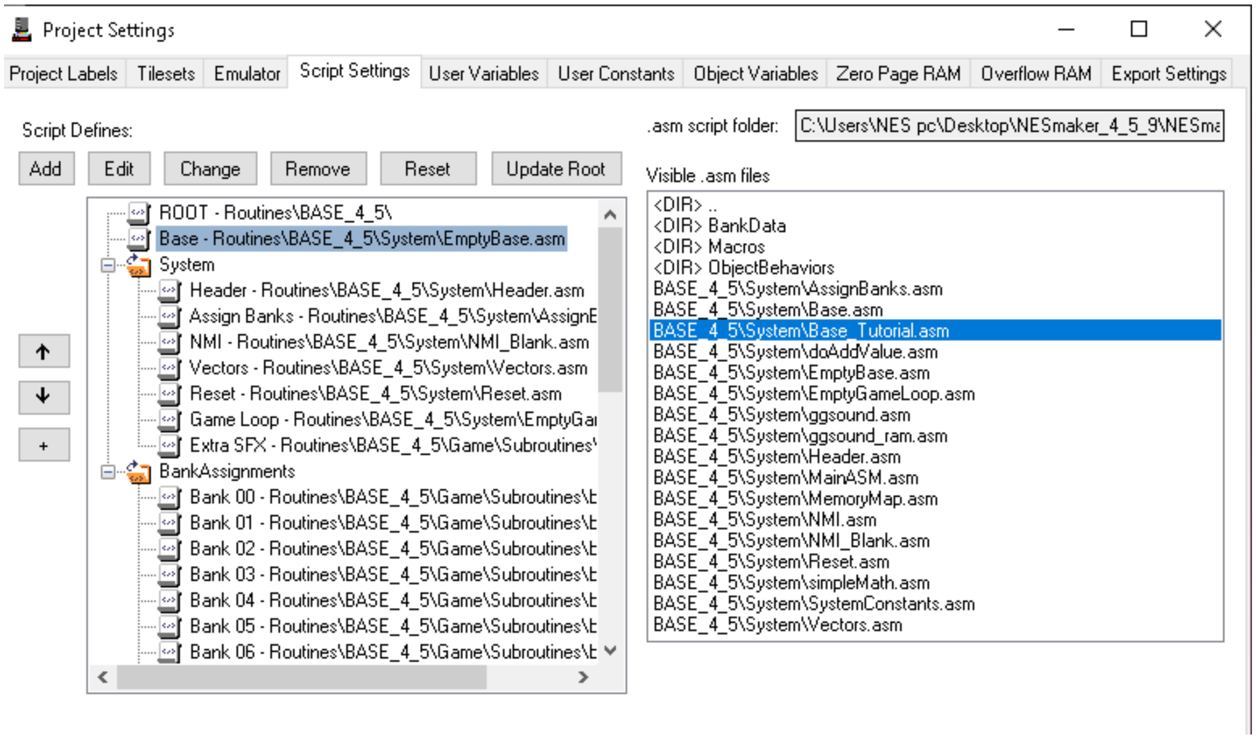
STEP 3: Iterate our EmptyBase script.
We do not want to modify our EmptyBase.asm, because we may want to utilize an EmptyBase script again in the future and start from this point. Instead, we'll iterate the script and give it a new name.

I hit Save As and called this file Base_Tutorial.asm. Here, I am showing the two tabs side by side. Currently, our empty module is still set to point to the EmptyBase file, so now we have to re-define our base script to the new one.


C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\Base_Tutorial.as
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm

```
;;;; This is what is needed for a completely empty base.
;;==========================
    .include SCR_HEADER
    .enum $0000
        ;;; this variable needed for blank reset routine.
        soft2001    .dsb 1
    .ende
;;==========================
    .include SCR_ASSIGN_BANKS
;;==========================
    .include SCR_RESET
;;==========================
    .include SCR_GAME_LOOP
;;==========================
    .include SCR_NMI
;;==========================
    .include SCR_VECTORS
```

STEP 4: Redefine the Base script.
Go to Project Settings and click on the Script Settings tab. There, you'll see a script define called Base, and you'll see it points to EmptyBase.asm. Click on this script define, then in the script finder, navigate to root \ System and double click on our newly created Base_Tutorial.asm. This will replace the Base script definition. Now, when it gets to the part of the script that includes the base file, it's the new file we just created that will be reference.



```
C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESma
```

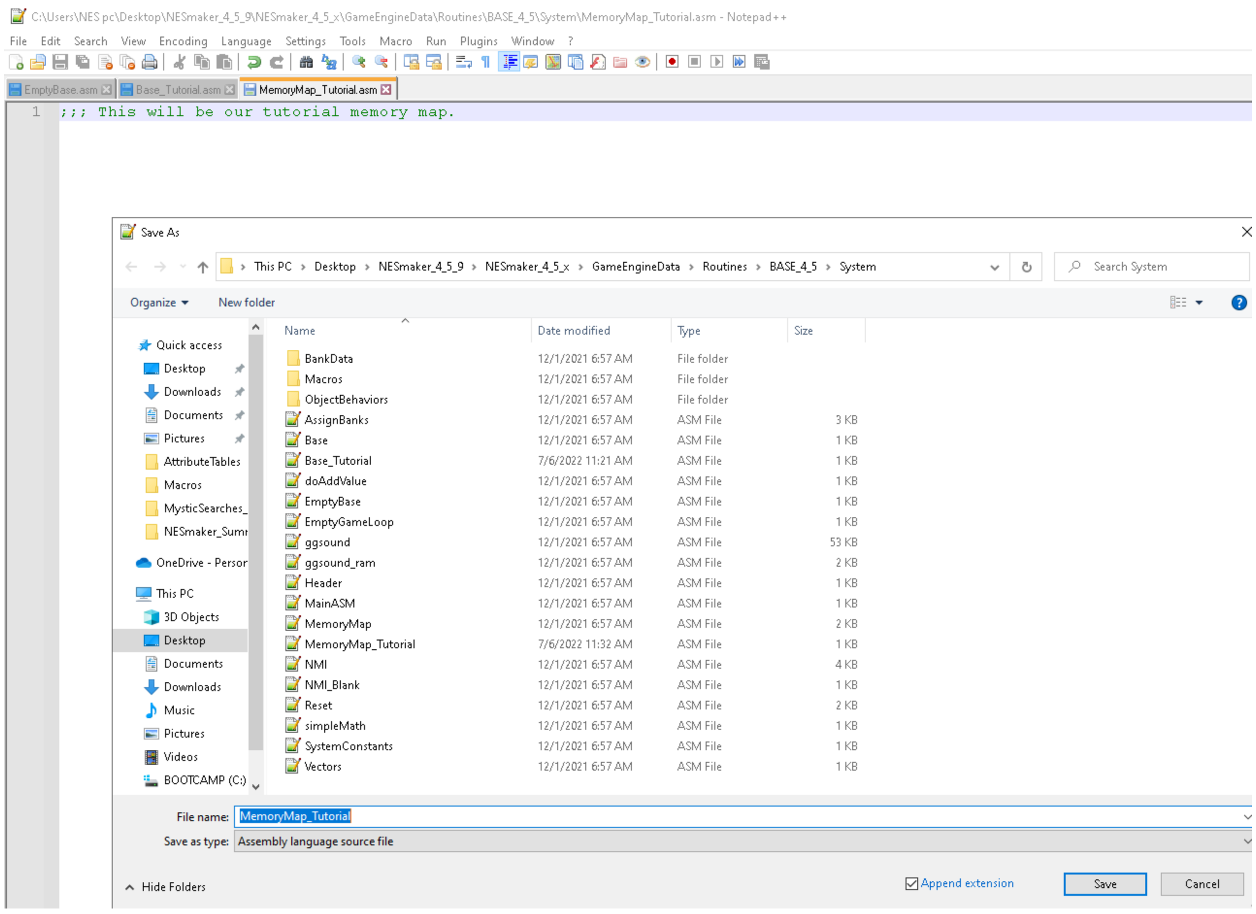
```
ROOT - Routines\BASE_4_5\
Base - Routines\BASE_4_5\System\EmptyBase.asm
System
    Header - Routines\BASE_4_5\System\Header.asm
    Assign Banks - Routines\BASE_4_5\System\AssignE
    NMI - Routines\BASE_4_5\System\NMI_Blank.asm
    Vectors - Routines\BASE_4_5\System\Vectors.asm
    Reset - Routines\BASE_4_5\System\Reset.asm
    Game Loop - Routines\BASE_4_5\System\EmptyGa
    Extra SFX - Routines\BASE_4_5\Game\Subroutines\
BankAssignments
    Bank 00 - Routines\BASE_4_5\Game\Subroutines\E
    Bank 01 - Routines\BASE_4_5\Game\Subroutines\E
    Bank 02 - Routines\BASE_4_5\Game\Subroutines\E
    Bank 03 - Routines\BASE_4_5\Game\Subroutines\E
    Bank 04 - Routines\BASE_4_5\Game\Subroutines\E
    Bank 05 - Routines\BASE_4_5\Game\Subroutines\E
    Bank 06 - Routines\BASE_4_5\Game\Subroutines\E
```

```
<DIR> ..
<DIR> BankData
<DIR> Macros
<DIR> ObjectBehaviors
BASE_4_5\System\AssignBanks.asm
BASE_4_5\System\Base.asm
BASE_4_5\System\Base_Tutorial.asm
BASE_4_5\System\doAddValue.asm
BASE_4_5\System\EmptyBase.asm
BASE_4_5\System\EmptyGameLoop.asm
BASE_4_5\System\ggsound.asm
BASE_4_5\System\ggsound_ram.asm
BASE_4_5\System\Header.asm
BASE_4_5\System\MainASM.asm
BASE_4_5\System\MemoryMap.asm
BASE_4_5\System\NMI.asm
BASE_4_5\System\NMI_Blank.asm
BASE_4_5\System\Reset.asm
BASE_4_5\System\simpleMath.asm
BASE_4_5\System\SystemConstants.asm
BASE_4_5\System\Vectors.asm
```

STEP 5: Create a new script for our Memory Map.
We could conceivably type up our entire memory map right in the Base_Tutorial.asm file if we wanted, but we're going to make an organized collection of interlinking scripts so that it's easy to find, edit, or replace different parts of our code. This is the real power of what NESmaker can do, and what is at the heart of each module.

Create a new script as a placeholder. In it, type ;;Thi will be our tutorial memory map. Save it into your NESmaker System folder as a file called MemoryMap_tutorial, and make sure to save as type Assembly Language Source FIle.

In order to save this, we will have to write something in the file. Just add at least one semicolon and write a few words. The semicolon will cause the assembler to ignore anything in the lone that comes after it. You can see use of this in the other scripts - it is a great way to add comments to yourself that don't get compiled with the actual game.



C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\MemoryMap_Tutorial.asm - Notepad++

```
;;; This will be our tutorial memory map.
```

Name	Date modified	Type	Size
BankData	12/1/2021 6:57 AM	File folder	
Macros	12/1/2021 6:57 AM	File folder	
ObjectBehaviors	12/1/2021 6:57 AM	File folder	
AssignBanks	12/1/2021 6:57 AM	ASM File	3 KB
Base	12/1/2021 6:57 AM	ASM File	1 KB
Base_Tutorial	7/6/2022 11:21 AM	ASM File	1 KB
doAddValue	12/1/2021 6:57 AM	ASM File	1 KB
EmptyBase	12/1/2021 6:57 AM	ASM File	1 KB
EmptyGameLoop	12/1/2021 6:57 AM	ASM File	1 KB
ggsound	12/1/2021 6:57 AM	ASM File	53 KB
ggsound_ram	12/1/2021 6:57 AM	ASM File	2 KB
Header	12/1/2021 6:57 AM	ASM File	1 KB
MainASM	12/1/2021 6:57 AM	ASM File	1 KB
MemoryMap	12/1/2021 6:57 AM	ASM File	2 KB
MemoryMap_Tutorial	7/6/2022 11:32 AM	ASM File	1 KB
NMI	12/1/2021 6:57 AM	ASM File	4 KB
NMI_Blank	12/1/2021 6:57 AM	ASM File	1 KB
Reset	12/1/2021 6:57 AM	ASM File	2 KB
simpleMath	12/1/2021 6:57 AM	ASM File	1 KB
SystemConstants	12/1/2021 6:57 AM	ASM File	1 KB
Vectors	12/1/2021 6:57 AM	ASM File	1 KB

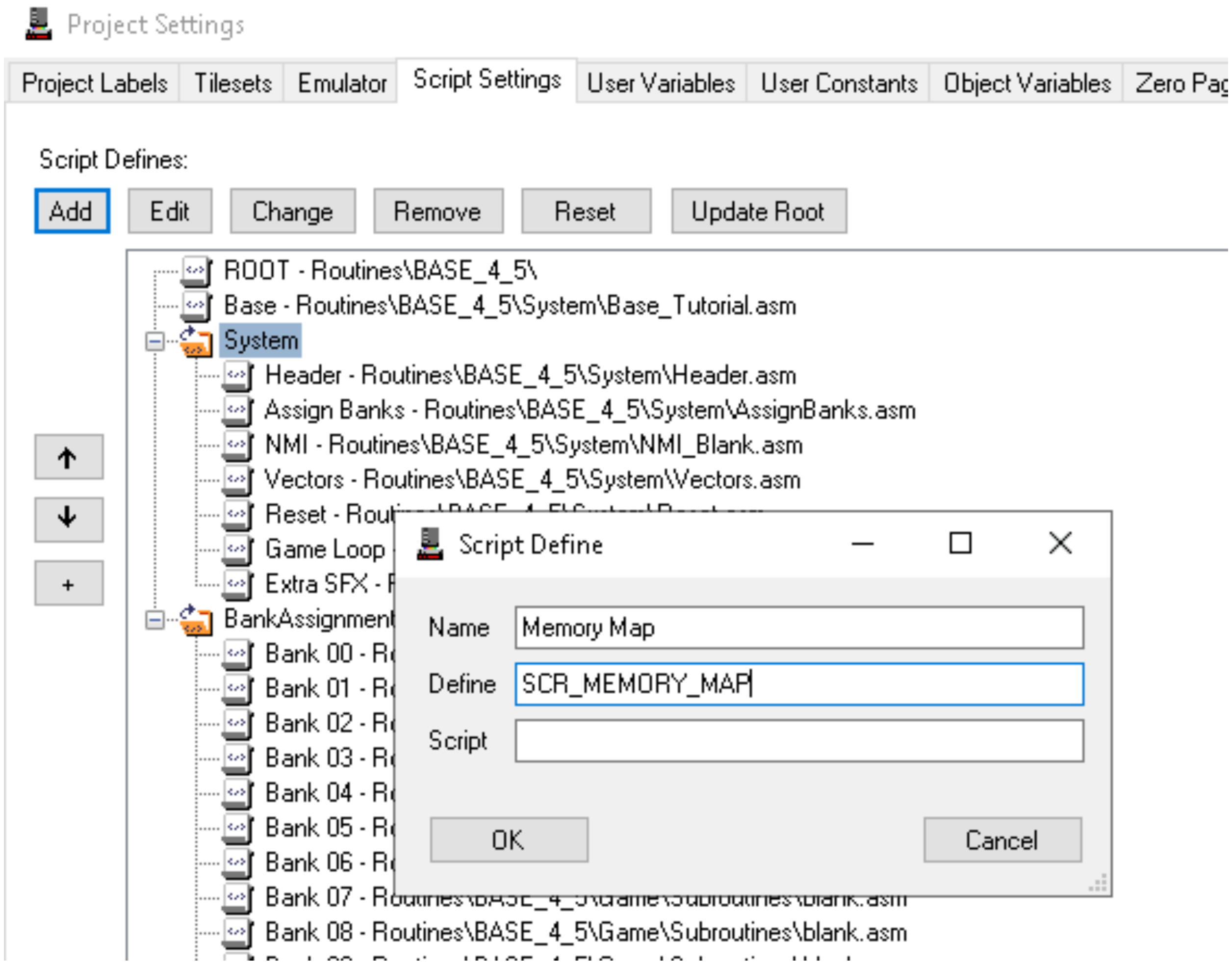
File name: MemoryMap_Tutorial

Save as type: Assembly language source file

STEP 6: Make our own script define.

In our Script Settings, click on the System node, and then hit the Add button. The add button will create a new script define in whichever node you have selected. You can also move scripts around to various nodes with the up and down arrows to the left side, or add new nodes or sub nodes with the small plus button under those arrows. For now, we want this in the System node, although placement here is irrelevant to the actual game - it is just for our own organization. This is something that has to do with our game's system, so we'll put it in system.

We need to give this a common name and a unique definition name. The common name can be anything descriptive that will read as its function. For this, I'll call it Memory Map. The define can also be anything, but I tend to start all of my script defines with SCR_ and use all caps, making them very easy to spot in a block of code. I will name this script definition SCR_MEMORY_MAP. Leave the script blank and press ok.



Project Settings

Project Labels | Tilesets | Emulator | Script Settings | User Variables | User Constants | Object Variables | Zero Pag

Script Defines:

Add | Edit | Change | Remove | Reset | Update Root

```
ROOT - Routines\BASE_4_5\
Base - Routines\BASE_4_5\System\Base_Tutorial.asm
System
    Header - Routines\BASE_4_5\System\Header.asm
    Assign Banks - Routines\BASE_4_5\System\AssignBanks.asm
    NMI - Routines\BASE_4_5\System\NMI_Blank.asm
    Vectors - Routines\BASE_4_5\System\Vectors.asm
    Reset - Rou
    Game Loop
    Extra SFX - F
BankAssignment
    Bank 00 - R
    Bank 01 - R
    Bank 02 - R
    Bank 03 - R
    Bank 04 - R
    Bank 05 - R
    Bank 06 - R
    Bank 07 - R
    Bank 08 - Routines\BASE_4_5\Game\Subroutines\blank.asm
```

Script Define

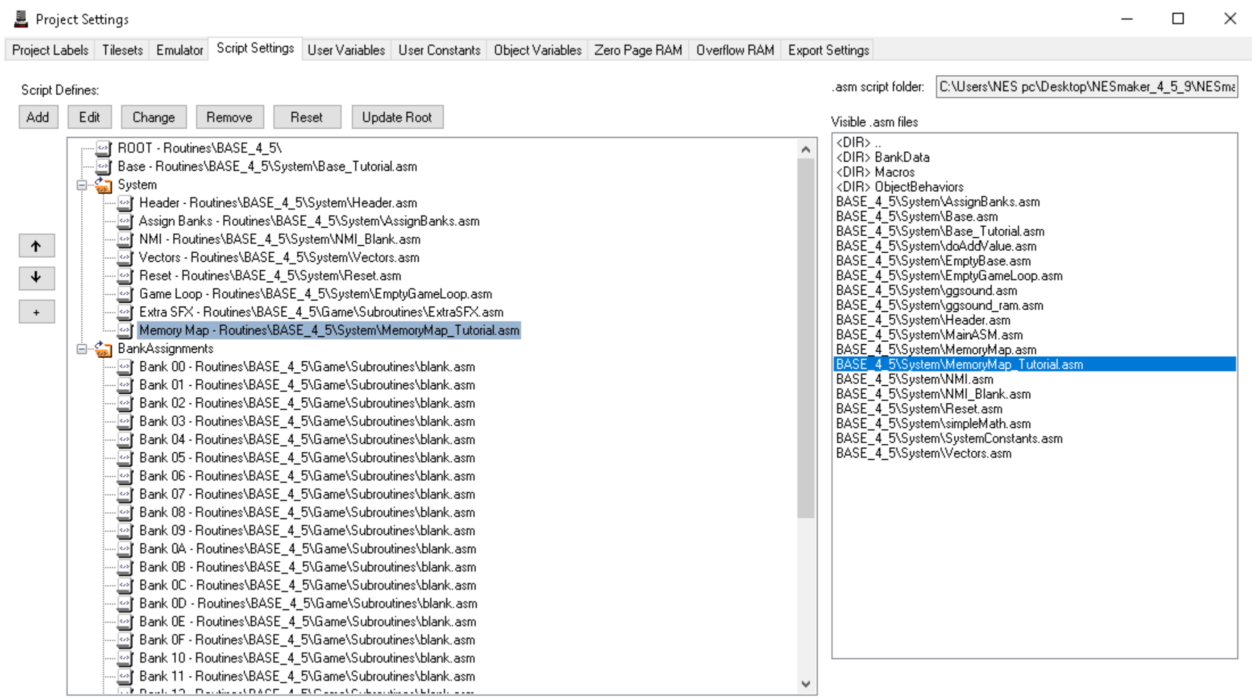
Name: Memory Map

Define: SCR_MEMORY_MAP

Script:

OK | Cancel

STEP 7: Link our new dummy script to this script define.
Click on our newly created script define, navigate to the System folder in our script finder and double click on the script we creed, called MemoryMap_Tutorial.asm.



.asm script folder: C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESma

```
ROOT - Routines\BASE_4_5\
Base - Routines\BASE_4_5\System\Base_Tutorial.asm
System
    Header - Routines\BASE_4_5\System\Header.asm
    Assign Banks - Routines\BASE_4_5\System\AssignBanks.asm
    NMI - Routines\BASE_4_5\System\NMI_Blank.asm
    Vectors - Routines\BASE_4_5\System\Vectors.asm
    Reset - Routines\BASE_4_5\System\Reset.asm
    Game Loop - Routines\BASE_4_5\System\EmptyGameLoop.asm
    Extra SFX - Routines\BASE_4_5\Game\Subroutines\ExtraSFX.asm
    Memory Map - Routines\BASE_4_5\System\MemoryMap_Tutorial.asm
BankAssignments
    Bank 00 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 01 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 02 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 03 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 04 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 05 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 06 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 07 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 08 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 09 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0A - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0B - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0C - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0D - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0E - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 0F - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 10 - Routines\BASE_4_5\Game\Subroutines\blank.asm
    Bank 11 - Routines\BASE_4_5\Game\Subroutines\blank.asm
```

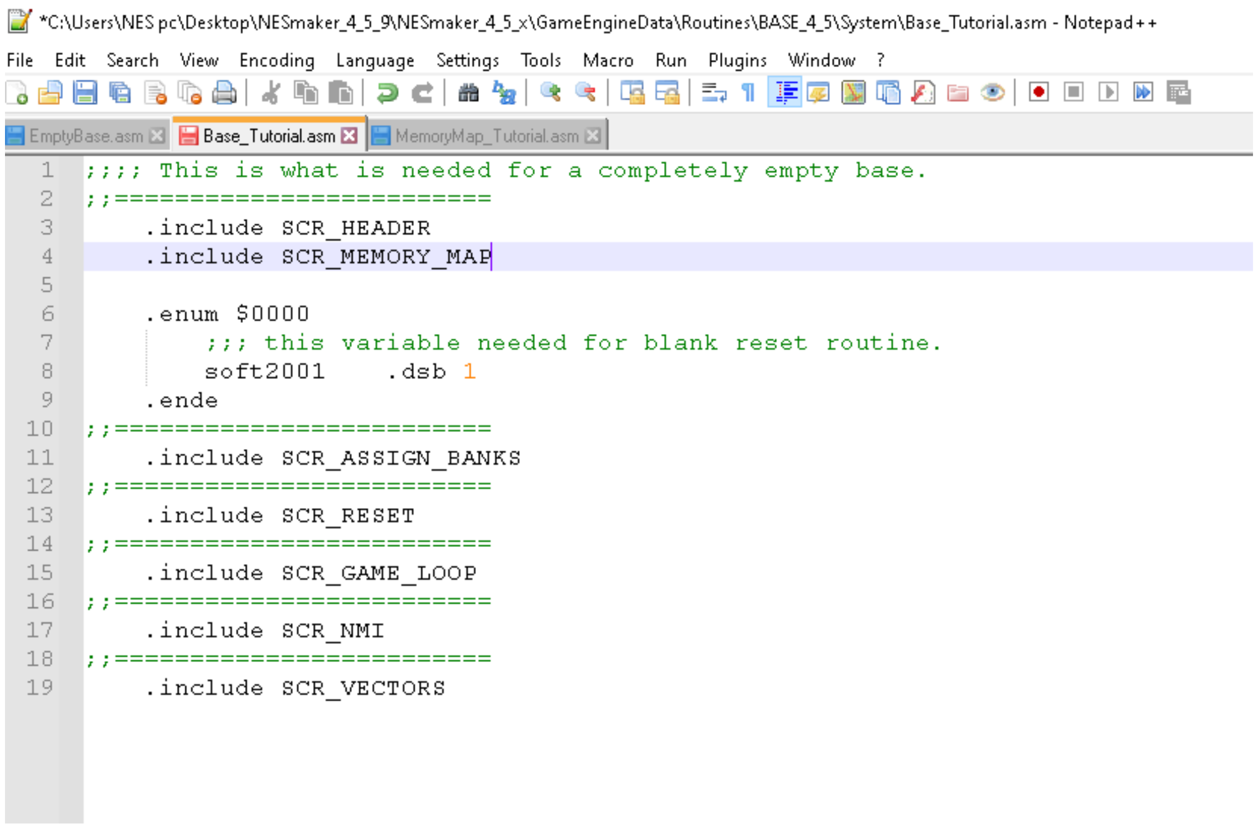
Visible .asm files

```
<DIR> ..
<DIR> BankData
<DIR> Macros
<DIR> ObjectBehaviors
BASE_4_5\System\AssignBanks.asm
BASE_4_5\System\Base.asm
BASE_4_5\System\Base_Tutorial.asm
BASE_4_5\System\doAddValue.asm
BASE_4_5\System\EmptyBase.asm
BASE_4_5\System\EmptyGameLoop.asm
BASE_4_5\System\ggsound.asm
BASE_4_5\System\ggsound_ram.asm
BASE_4_5\System\Header.asm
BASE_4_5\System\MainASM.asm
BASE_4_5\System\MemoryMap.asm
BASE_4_5\System\MemoryMap_Tutorial.asm
BASE_4_5\System\NMI.asm
BASE_4_5\System\NMI_Blank.asm
BASE_4_5\System\Reset.asm
BASE_4_5\System\simpleMath.asm
BASE_4_5\System\SystemConstants.asm
BASE_4_5\System\Vectors.asm
```

This script is still not actually present in our game. But now, what we can do is we can tell our game to include SCR_MEMORY_MAP, the actual definition, wherever we want to put this memory map file. It will then place in that spot whatever script we attached to this SCR_MEMORY_MAP script define.

STEP 8: Include our SCR_MEMORY_MAP into our project.
In order to include this into our code, open up the Base_Tutorial.asm script by clicking on it in our Script Settings and clicking the edit button at the top of the screen.

Just below the header, add an include command, followed by SCR_MEMORY_MAP.


*C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\Base_Tutorial.asm - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm
MemoryMap_Tutorial.asm

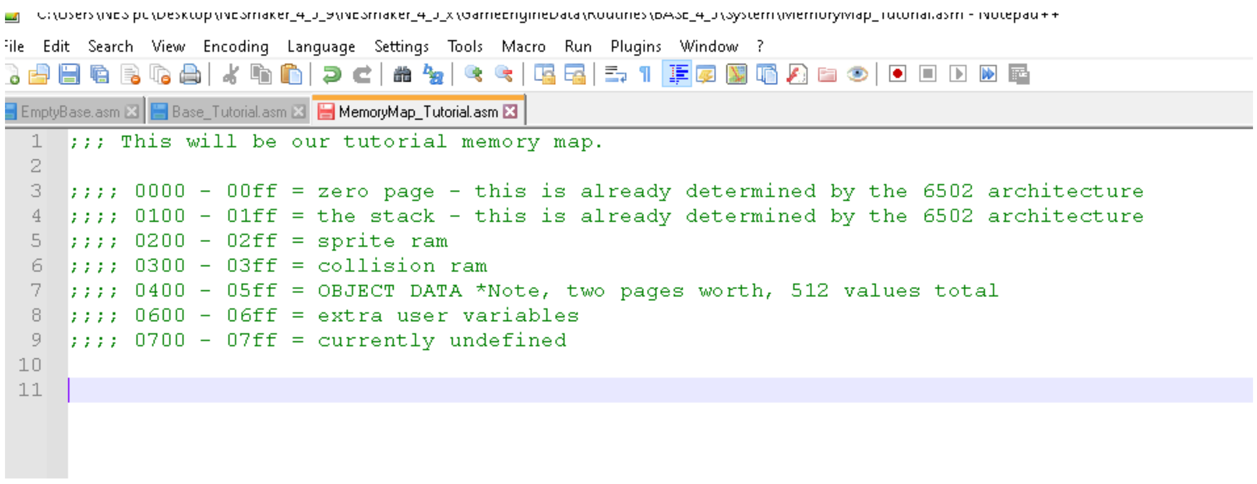
```
;;;; This is what is needed for a completely empty base.
;;========================
    .include SCR_HEADER
    .include SCR_MEMORY_MAP

    .enum $0000
        ;;; this variable needed for blank reset routine.
        soft2001    .dsb 1
    .ende
;;========================
    .include SCR_ASSIGN_BANKS
;;========================
    .include SCR_RESET
;;========================
    .include SCR_GAME_LOOP
;;========================
    .include SCR_NMI
;;========================
    .include SCR_VECTORS
```

Make sure to save this file. Now, whatever we code in the MemoryMap_Tutorial.asm will be added to our base code. And if we change where SCR_MEMORY_MAP points, it will insert whatever script we point it to in that spot instead.

STEP 9: Writing a logical memory map.

Open our memory map file. Right now, it should be blank other than the commented out verbiage at the top.

I'm going to start by adding comments about how my memory map will be set up. This is just reference for me to understand what is going where.


```
;;; This will be our tutorial memory map.

;;;; 0000 - 00ff = zero page - this is already determined by the 6502 architecture
;;;; 0100 - 01ff = the stack - this is already determined by the 6502 architecture
;;;; 0200 - 02ff = sprite ram
;;;; 0300 - 03ff = collision ram
;;;; 0400 - 05ff = OBJECT DATA *Note, two pages worth, 512 values total
;;;; 0600 - 06ff = extra user variables
;;;; 0700 - 07ff = currently undefined

```

Step 10: Set up a few CONSTANTS.

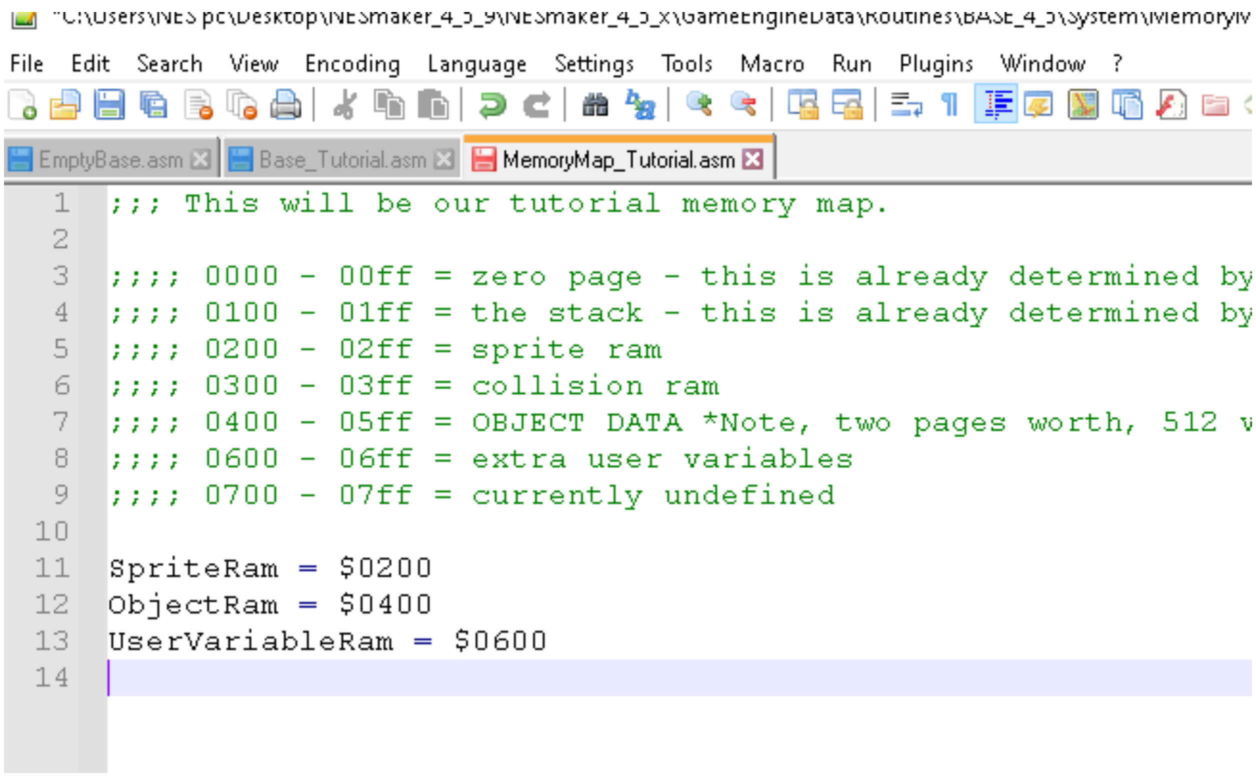
If you're unfamiliar with constants, they're pretty easy to understand. We already talked about RAM addresses and variables using labels. Variables are things that change over the course of the game - things like the position of an object, what screen is being show, what sound effect is playing, what tileset is selected, how much ammy a player has, what inputs are pressed, etc. We hae those proverbial mailboxes set up so that way when the game needs to know what input is coming from the gamepad, it can quickly look at the right mailbox, see what's currently in there, and consequently proceed.

Constants are similar to variable in that we will use common name labels to define them, but they are not actually included in our game's RAM. Instead, they are transformed into whatever they are equal to at runtime. Think of it like an autofill. If you were filling out an application on line, it might ask you at the top to enter your name. It would then apply your name to every line that required your name to be present. It's not variable once you have entered it - it will apply to every field. If you change it in the top field, it will change it in all subsequent fields. This is kind of like how constants work.

If I had a declaration to read that said "I, (state your name) promise to muscle through all these tutorials even when it gets hard, or my name isn't (state your name)", I'm inserting what my name equals into those parentheticals wherever they appear. Unlike variable values, they wouldn't change depending on the state of the declaration I am reading.

One reason this is great is that they don't take up any memory, but it's easy to make quick changes. For instance, if I wanted all objects to move at a speed of 5, always, I could make a constant called OBJECT_SPEED and set it to 5. Now, I might have to use that speed value in a hundred places in code. Without the constant, I would just be writing 5 in all those places. If I then decided I wanted to change the object speeds, I would have to find every single place in code that referenced the speed and change the 5 to a 3. But if instead I used OBJECT_SPEED in all the places where speed is referenced, all I would have to do is change OBJECT_SPEED where it is set from 5 to 3, and when the game compiles, the number 3 would get inserted into all of the places that reference OBJECT_SPEED.

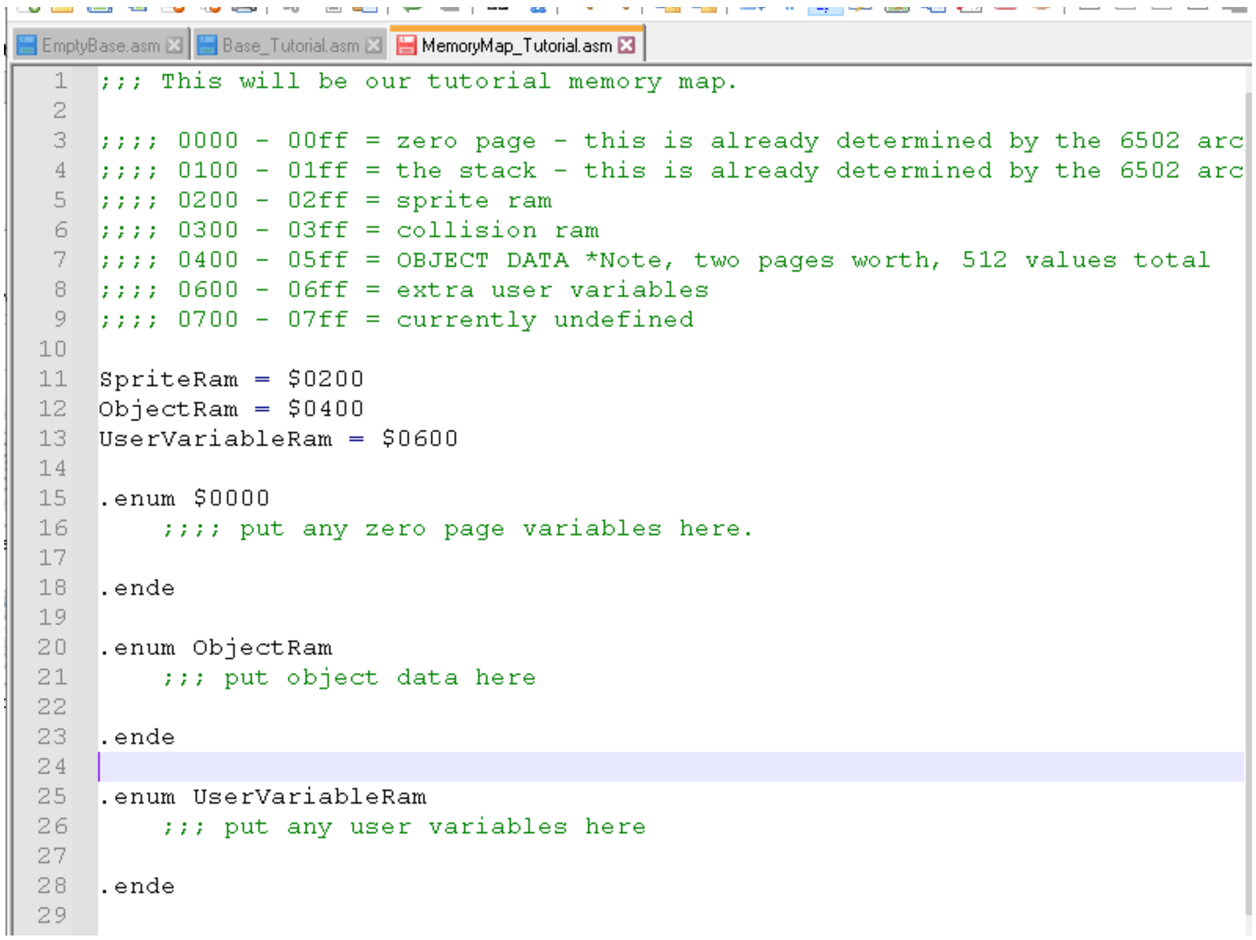
We are going to create a few constants to give common names to our memory pages. We'll use a label called SpriteRam, a label called ObjectRam, and a label called UserVariableRam. If we look back at our commented memory map, we can see that we want our SpriteRam to start at $0200, our ObjectRam to start at $0400, and our UserVariableRam to start at $0600. We'll set that up in our memory map.


*C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\Memory
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm
MemoryMap_Tutorial.asm

```
;;; This will be our tutorial memory map.

;;;; 0000 - 00ff = zero page - this is already determined by
;;;; 0100 - 01ff = the stack - this is already determined by
;;;; 0200 - 02ff = sprite ram
;;;; 0300 - 03ff = collision ram
;;;; 0400 - 05ff = OBJECT DATA *Note, two pages worth, 512 v
;;;; 0600 - 06ff = extra user variables
;;;; 0700 - 07ff = currently undefined

SpriteRam = $0200
ObjectRam = $0400
UserVariableRam = $0600

```

This currently doesn't mean anything to our game yet, but we're starting to set up a foundation for what will live where in memory. Next we can start to put some things into these places.

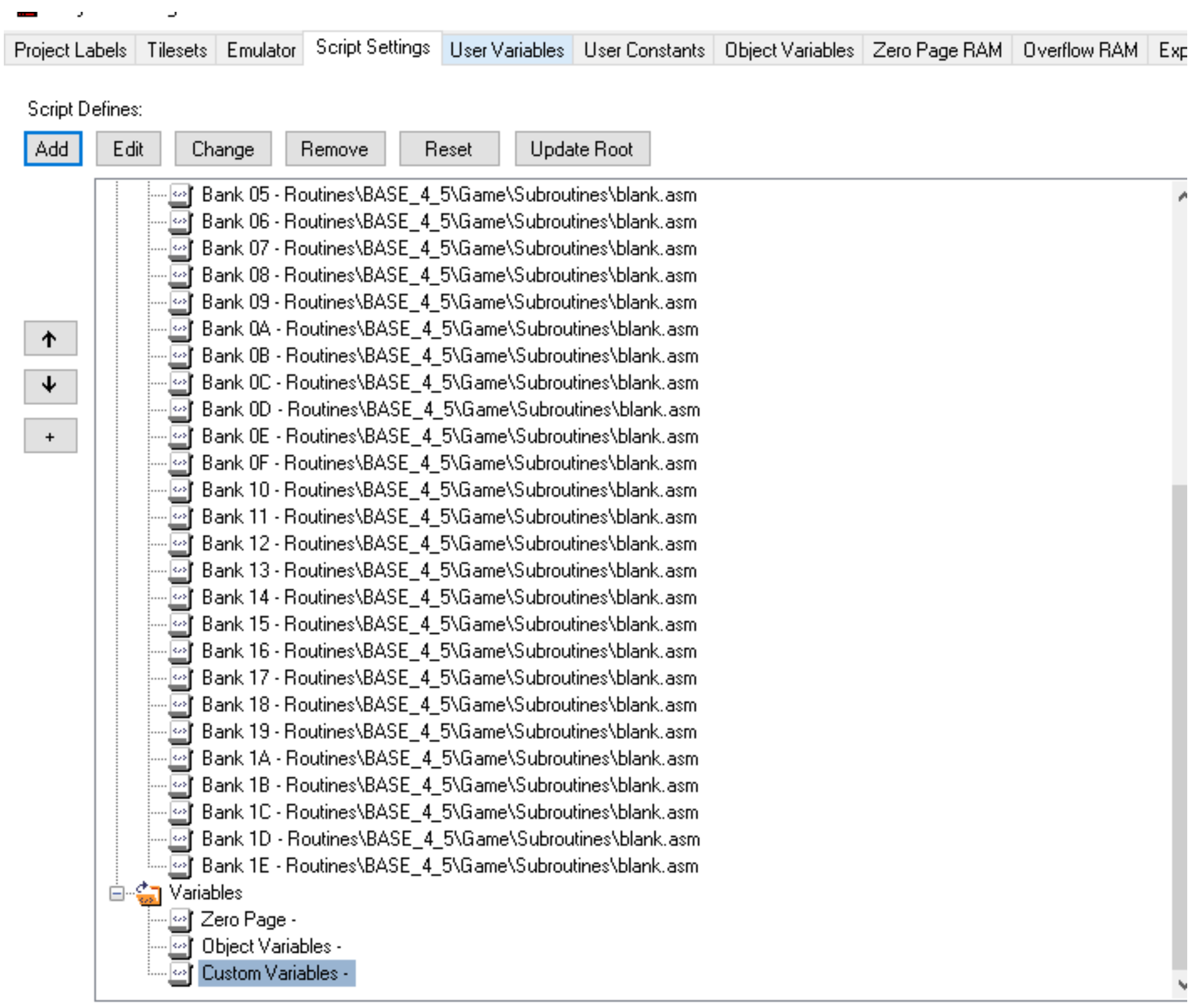
Step 11: Using enumeration to place different scripts into their proper places. We're going to use an ASM function called Enum, which means to start an enumeration. We'll keep it easy and not get too technical. For a basic understanding of what is happening here, we're basically telling the game to turn the page to address 0000 and start writing things from there forward. This is how we're going to place certain chunks of code or certain variables at defined places. We're going to point to address 0000, which is where we're going to put zero page data. We're going to pont to our ObjectRam position, which is $0400, and we're going to include our file that deals with object data. We're going to point to our UserVariableRam position, which is $0600, and we're going to include our user variables. We'll close each enumeration with the function .ende like what is shown in the image.



```
;;; This will be our tutorial memory map.

;;;; 0000 - 00ff = zero page - this is already determined by the 6502 arc
;;;; 0100 - 01ff = the stack - this is already determined by the 6502 arc
;;;; 0200 - 02ff = sprite ram
;;;; 0300 - 03ff = collision ram
;;;; 0400 - 05ff = OBJECT DATA *Note, two pages worth, 512 values total
;;;; 0600 - 06ff = extra user variables
;;;; 0700 - 07ff = currently undefined

SpriteRam = $0200
ObjectRam = $0400
UserVariableRam = $0600

.enum $0000
    ;;;; put any zero page variables here.

.ende

.enum ObjectRam
    ;;; put object data here

.ende

.enum UserVariableRam
    ;;; put any user variables here

.ende

```

Our game currently has no zero page variables, no object ram data, and no user variables. NESmaker has easy GUI interfaces to deal with these. It allows you to create and organize variables in the software, and then it spits that data out into the GameData folder. Ordinarily what you would see here in each section is an include of those files in each section. But because we're determined to be as from-scratch as we can to get a better understanding of what's under the hood, we''l make our own files instead of using the built in GUI.

Step 12: Make a script node for variables and three script defines with it.
In our Script Settings, press the small plus sign on the left of our defines. This will add a new group node at the bottom of the list. Right click on that new folder node and rename it Variables. Then, with that Variables node selected, click Add, and make a Zero Page define pointing to SCR_ZERO_PAGE, a Object Variables define pointing to SCR_OBJECT_VARS, and a Custom Variables pointing to SCR_CUSTOM_VARS. We don't yet have scripts for these.

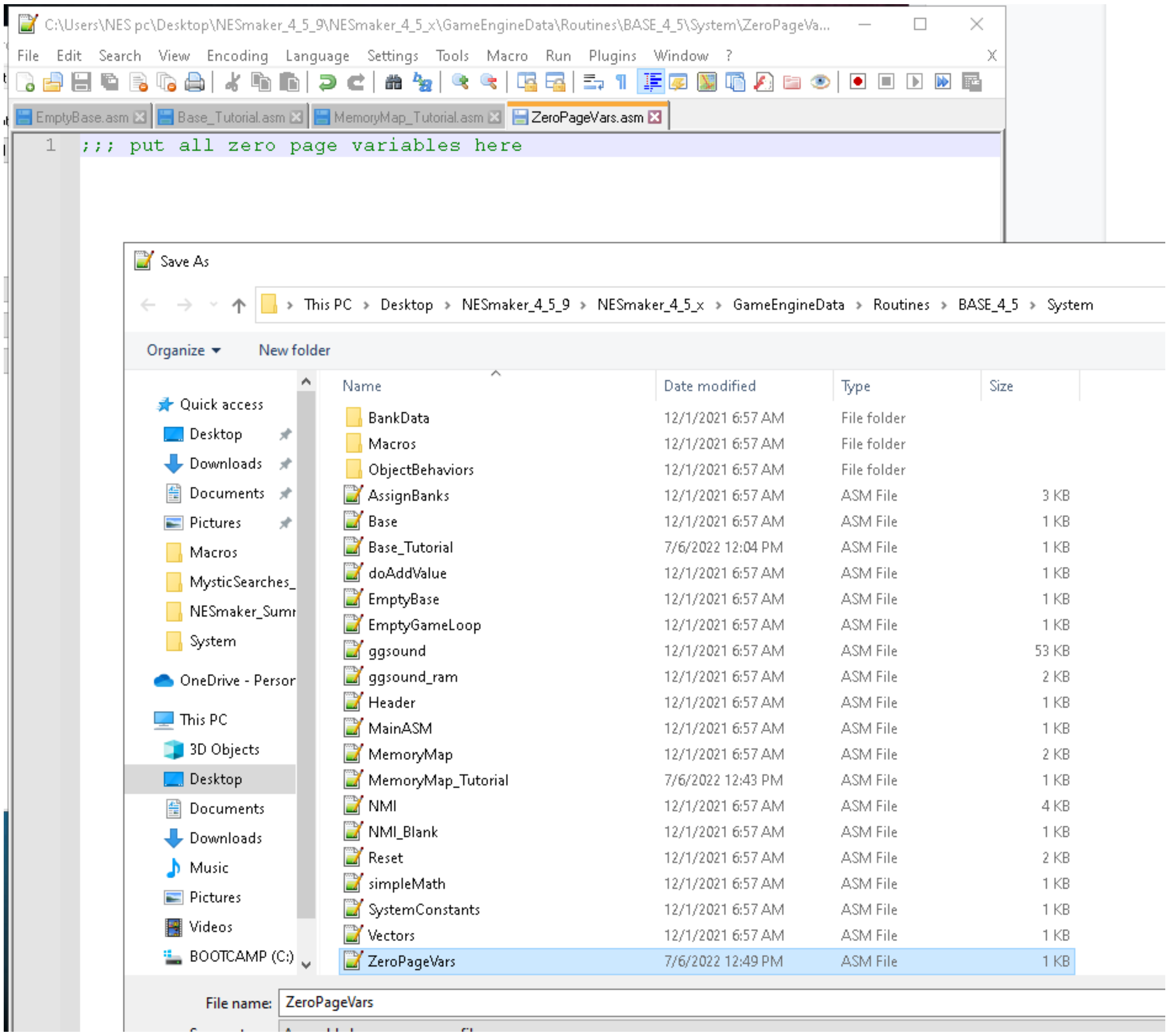

Project Labels | Tilesets | Emulator | Script Settings | User Variables | User Constants | Object Variables | Zero Page RAM | Overflow RAM | Exp

Script Defines:

Add | Edit | Change | Remove | Reset | Update Root

```
Bank 05 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 06 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 07 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 08 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 09 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0A - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0B - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0C - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0D - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0E - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0F - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 10 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 11 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 12 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 13 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 14 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 15 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 16 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 17 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 18 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 19 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1A - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1B - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1C - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1D - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1E - Routines\BASE_4_5\Game\Subroutines\blank.asm
Variables
    Zero Page -
    Object Variables -
    Custom Variables -
```

Step 13: Make a script for each of the variable types we just defined.
Call it ZeroPageVars, make sure you're saving it as an assembly source file type, and save it to the System folder.


C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\ZeroPageVa...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm
MemoryMap_Tutorial.asm
ZeroPageVars.asm
Save As
This PC › Desktop › NESmaker_4_5_9 › NESmaker_4_5_x › GameEngineData › Routines › BASE_4_5 › System
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
Macros
MysticSearches_
NESmaker_Sumr
System
OneDrive - Persor
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
BOOTCAMP (C:)

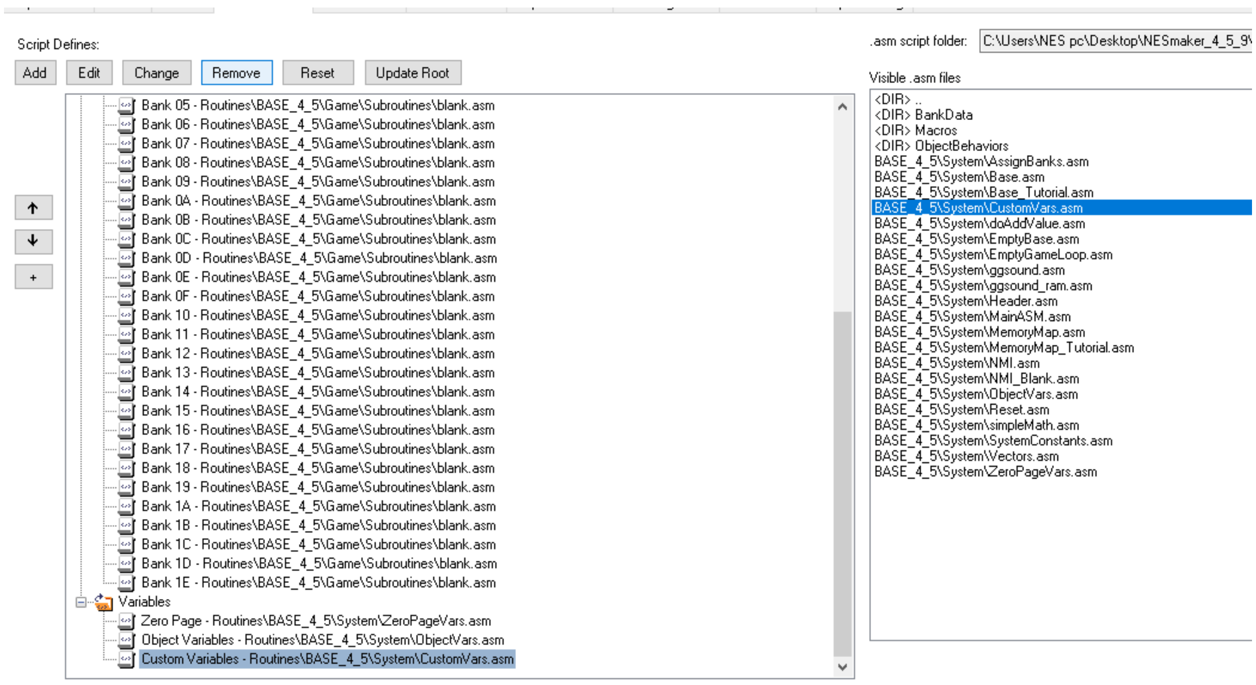
```
;;; put all zero page variables here
```

Name	Date modified	Type	Size
BankData	12/1/2021 6:57 AM	File folder	
Macros	12/1/2021 6:57 AM	File folder	
ObjectBehaviors	12/1/2021 6:57 AM	File folder	
AssignBanks	12/1/2021 6:57 AM	ASM File	3 KB
Base	12/1/2021 6:57 AM	ASM File	1 KB
Base_Tutorial	7/6/2022 12:04 PM	ASM File	1 KB
doAddValue	12/1/2021 6:57 AM	ASM File	1 KB
EmptyBase	12/1/2021 6:57 AM	ASM File	1 KB
EmptyGameLoop	12/1/2021 6:57 AM	ASM File	1 KB
ggsound	12/1/2021 6:57 AM	ASM File	53 KB
ggsound_ram	12/1/2021 6:57 AM	ASM File	2 KB
Header	12/1/2021 6:57 AM	ASM File	1 KB
MainASM	12/1/2021 6:57 AM	ASM File	1 KB
MemoryMap	12/1/2021 6:57 AM	ASM File	2 KB
MemoryMap_Tutorial	7/6/2022 12:43 PM	ASM File	1 KB
NMI	12/1/2021 6:57 AM	ASM File	4 KB
NMI_Blank	12/1/2021 6:57 AM	ASM File	1 KB
Reset	12/1/2021 6:57 AM	ASM File	2 KB
simpleMath	12/1/2021 6:57 AM	ASM File	1 KB
SystemConstants	12/1/2021 6:57 AM	ASM File	1 KB
Vectors	12/1/2021 6:57 AM	ASM File	1 KB
ZeroPageVars	7/6/2022 12:49 PM	ASM File	1 KB

File name: ZeroPageVars

Similarly, make files for ObjectVars and CustomVars.

Step 14: In script settings, attach these new blank scripts to our new variable script defines.

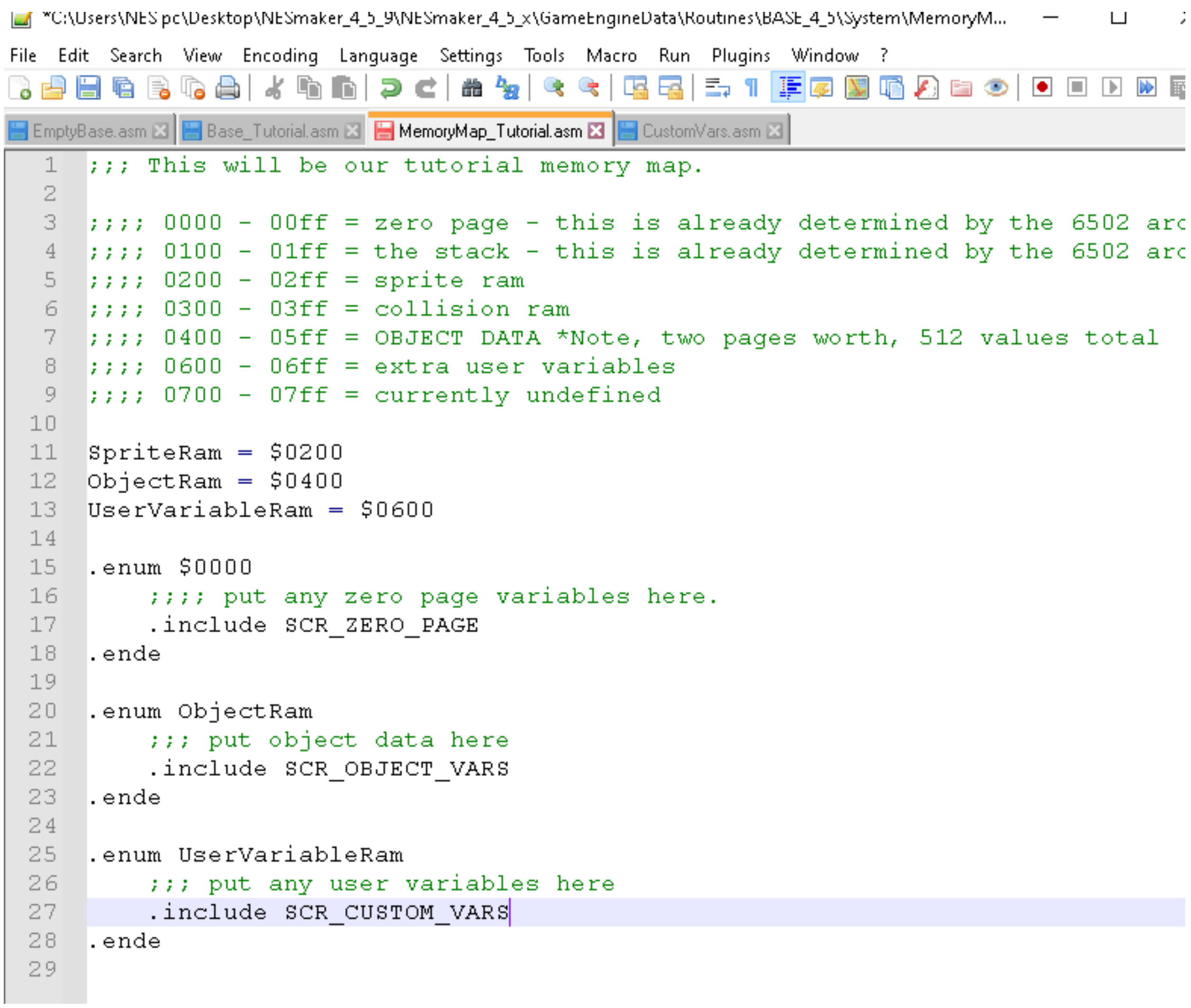


C:\Users\NES pc\Desktop\NESmaker_4_5_9\

```
Bank 05 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 06 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 07 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 08 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 09 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0A - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0B - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0C - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0D - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0E - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 0F - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 10 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 11 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 12 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 13 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 14 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 15 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 16 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 17 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 18 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 19 - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1A - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1B - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1C - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1D - Routines\BASE_4_5\Game\Subroutines\blank.asm
Bank 1E - Routines\BASE_4_5\Game\Subroutines\blank.asm
Variables
  Zero Page - Routines\BASE_4_5\System\ZeroPageVars.asm
  Object Variables - Routines\BASE_4_5\System\ObjectVars.asm
  Custom Variables - Routines\BASE_4_5\System\CustomVars.asm
```

```
<DIR> ..
<DIR> BankData
<DIR> Macros
<DIR> ObjectBehaviors
BASE_4_5\System\AssignBanks.asm
BASE_4_5\System\Base.asm
BASE_4_5\System\Base_Tutorial.asm
BASE_4_5\System\CustomVars.asm
BASE_4_5\System\doAddValue.asm
BASE_4_5\System\EmptyBase.asm
BASE_4_5\System\EmptyGameLoop.asm
BASE_4_5\System\ggsound.asm
BASE_4_5\System\ggsound_ram.asm
BASE_4_5\System\Header.asm
BASE_4_5\System\MainASM.asm
BASE_4_5\System\MemoryMap.asm
BASE_4_5\System\MemoryMap_Tutorial.asm
BASE_4_5\System\NMI.asm
BASE_4_5\System\NMI_Blank.asm
BASE_4_5\System\ObjectVars.asm
BASE_4_5\System\Reset.asm
BASE_4_5\System\simpleMath.asm
BASE_4_5\System\SystemConstants.asm
BASE_4_5\System\Vectors.asm
BASE_4_5\System\ZeroPageVars.asm
```

Step 15: Include these scripts in their proper places in our Memory Map file.
Don’t forget to save the files as you go.

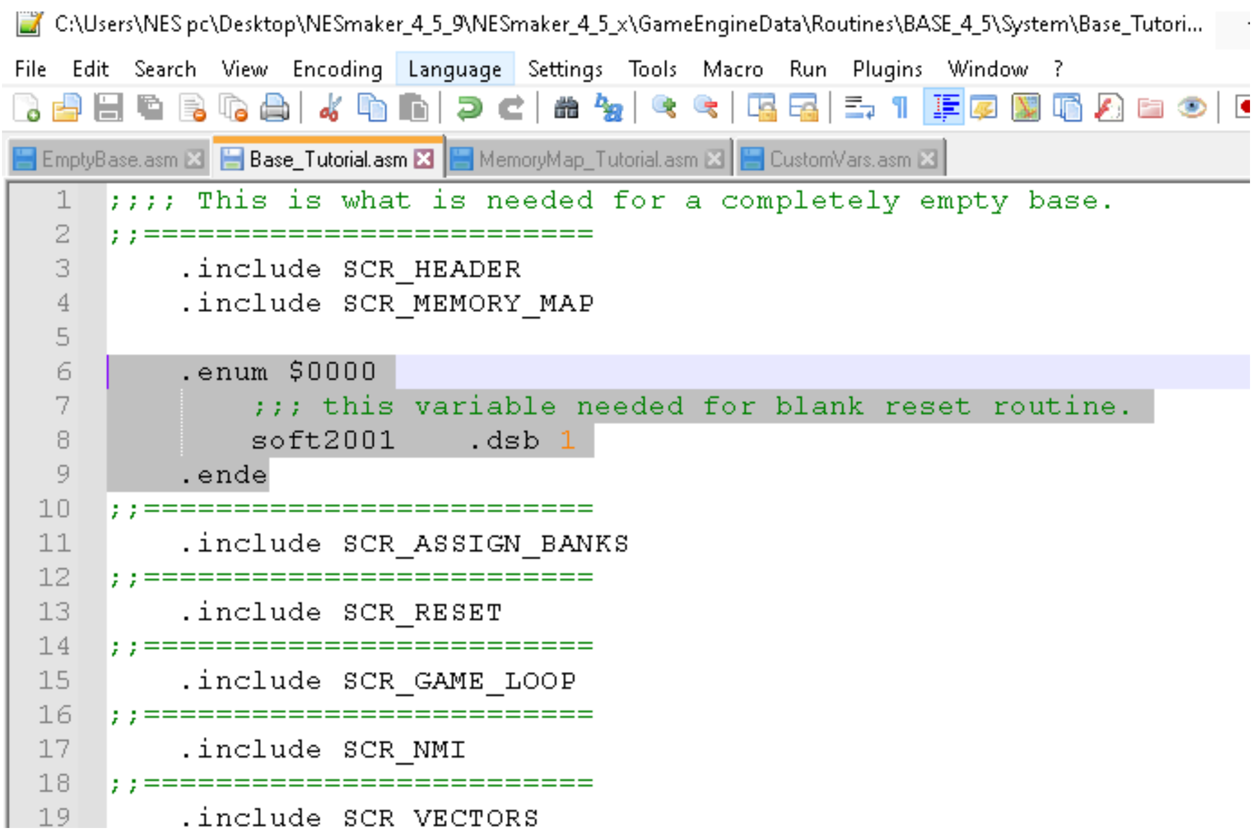

*C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\MemoryM...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm
MemoryMap_Tutorial.asm
CustomVars.asm

```
;;; This will be our tutorial memory map.

;;;; 0000 - 00ff = zero page - this is already determined by the 6502 arc
;;;; 0100 - 01ff = the stack - this is already determined by the 6502 arc
;;;; 0200 - 02ff = sprite ram
;;;; 0300 - 03ff = collision ram
;;;; 0400 - 05ff = OBJECT DATA *Note, two pages worth, 512 values total
;;;; 0600 - 06ff = extra user variables
;;;; 0700 - 07ff = currently undefined

SpriteRam = $0200
ObjectRam = $0400
UserVariableRam = $0600

.enum $0000
    ;;;; put any zero page variables here.
    .include SCR_ZERO_PAGE
.ende

.enum ObjectRam
    ;;; put object data here
    .include SCR_OBJECT_VARS
.ende

.enum UserVariableRam
    ;;; put any user variables here
    .include SCR_CUSTOM_VARS
.ende

```

Now, whatever you put in the SCR_ZERO_PAGE script will be placed starting at memory address $0000, everything you put in the SCR_OBJECT_VARS script will be placed starting at memory address $0400, and everything you put in the SCR_CUSTOM_VARS will be placed starting at memory position $0600.

We've now completely bypassed NESmaker's default GUI based variable handling system. If we add variables to our GUI variable managers in our project settings, it will still generate a file that it will put in our GameData folder, however our game will never see it since it does not include that file from the GameData folder into the project. Instead, we'll manually manage variables in these scripts, and they'll always be very easy to find - too add or make changes all we will have to do is go to our script settings, scroll to the variable node, and press edit.

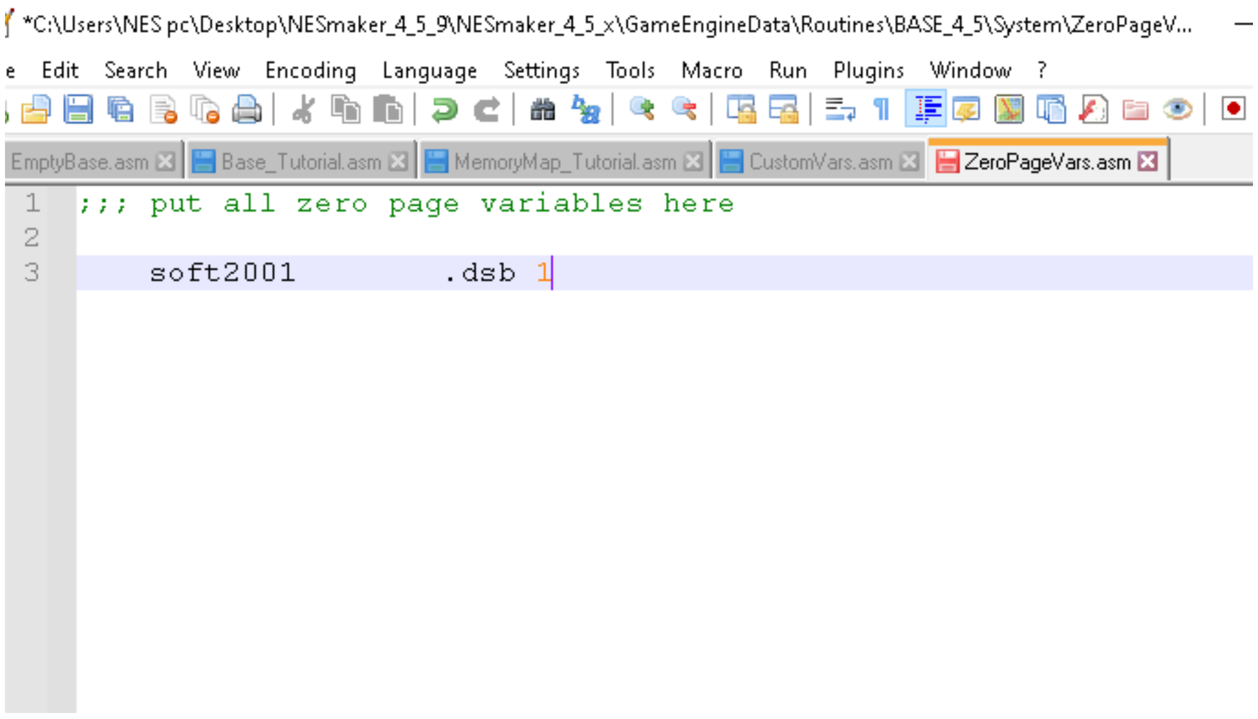
In fact, something that already exists in our code that we haven't looked at will make a bit more sense. Open our Base_Tutorial script by going to Script Settings, clicking on Base, and pressing edit. What you'll see is that there is a enumerator that is pointing to position $0000. We know that we're using that space for our zero page variables. This variable, soft2001, is a one byte variable that was required for our blank game to compile and run. But now, it would be better to keep this variable with the rest of the zero page variables. We are going to remove the entire enumerator from this file, and place soft2001 in our zero page variable script.


C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\Base_Tutori...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm
MemoryMap_Tutorial.asm
CustomVars.asm

```
;;;; This is what is needed for a completely empty base.
;;==========================
    .include SCR_HEADER
    .include SCR_MEMORY_MAP

    .enum $0000
        ;;; this variable needed for blank reset routine.
        soft2001    .dsb 1
    .ende
;;==========================
    .include SCR_ASSIGN_BANKS
;;==========================
    .include SCR_RESET
;;==========================
    .include SCR_GAME_LOOP
;;==========================
    .include SCR_NMI
;;==========================
    .include SCR VECTORS
```

Step 16: Remove the selection from the previous image and save our Base_Tutorial without that section.

Then, open the script for your Zero Page Variables by going to script settings, clicking on Zero Page, and pressing edit. Add the variable soft2001 there, and after the variable write .dsb 1. For our purposes this is easy to understand. It will let your game know that soft2001, which is a variable that our current engine needs in order to properly compile, exists and is a one byte variable. Make sure to save.


*C:\Users\NES pc\Desktop\NESmaker_4_5_9\NESmaker_4_5_x\GameEngineData\Routines\BASE_4_5\System\ZeroPageV...
e Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
EmptyBase.asm
Base_Tutorial.asm
MemoryMap_Tutorial.asm
CustomVars.asm
ZeroPageVars.asm

```
;;; put all zero page variables here

    soft2001        .dsb 1
```

If you have done everything right to this point, you should be able to return to NESmaker, click on export and test, and compile your game. Literally, the game is still a whole lot of nothing, but it should assemble and open a blank screen in the emulator. If it does not open in the emulator, or returns an error in the assembler window, check to see the first error thrown. Check your spellings, check your cases as everything is case sensitive. Worst case scenario, walk through the above steps to see if you can track down the problem.

END OF LESSON 1

From the NESmaker FIle menu, make sure to save the project. I'm going to call mine Practice_FromScratch.

If you're someone who is used to the GUI driven method of using NESmaker, it may seem like we just put in a lot of work to see very little result. These formative setup steps are the sort of things that all games more or less have in common, and that you can't really get visual feedback to determine if you've done them correctly. That's why most modules already have all these things established in NESmaker so that you can dive straight into the creative part rather than spending all your time setting up these skeletons. But this is an advanced tutorial that is intended to show how NESmaker can be utilized in a from-scratch workflow, and to give users more comfort in understanding how to manipulate what's under the hood. That means progress will seem slow, especially compared to just jumping in and designing screens within a few minutes! But hopefully you'll learn a lot, and your games will be better for it.

By this point, you should have a pretty good grasp on how the Script Settings tab works; how to set up new script defines, make new scripts, and include those scripts into your project. You should understand the general idea of a memory map, of variables and labels and constants. We will continue to revisit these, so if the concepts are murky, definitely ready through this section again before moving forward. Hopefully, as with anything else, the more we practice the clearer it will become.