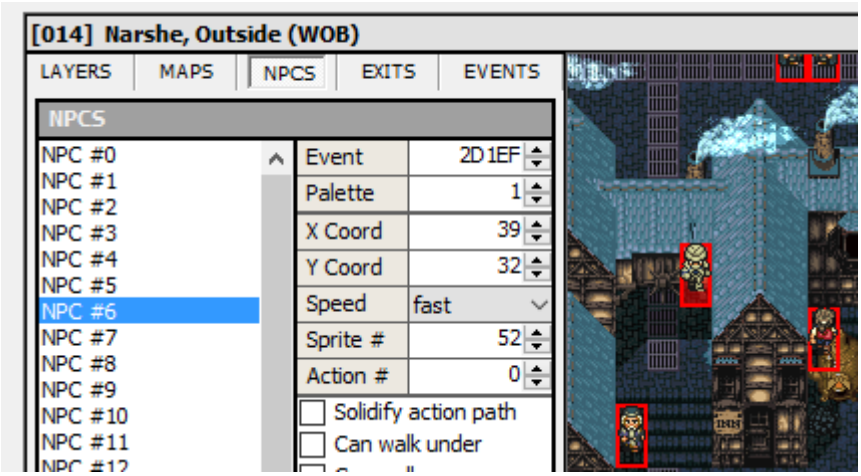
Action Queues

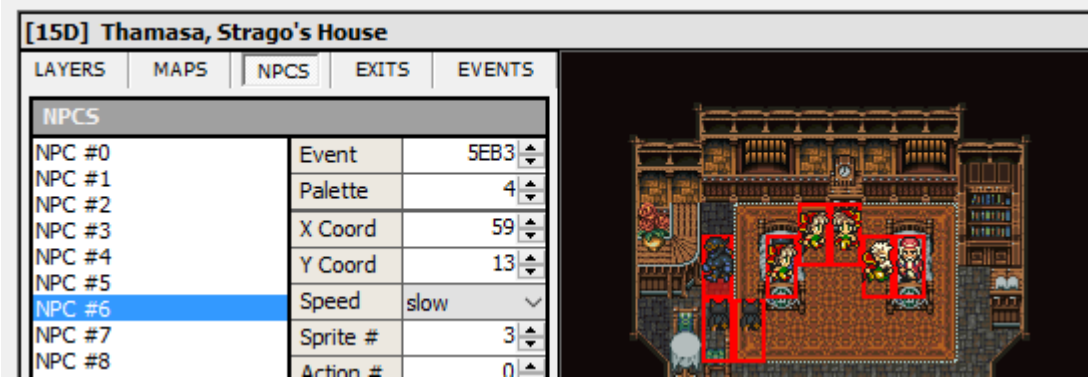
One question that you may have been asking yourself is “How do I make the characters move”? Action queues contain a set of actions that you want a character to perform. For example, you could queue up the actions “walk one step up”, “turn left”, and “nod head left”. If you set the action queues up correctly, you can have multiple characters executing a complex series of actions at the same time. If you've been following the tutorial so far, action queues will be fairly straightforward to learn.

Starting an Action Queue

Starting an action queue works just like executing any other event command, although the number of parameters can vary depending on how many actions you want the character to take. Any command between $00 and $34 will start an action queue for one character. The Event Command Document has some information on which character will take action. This character will fall into one of the following categories:

- $00 - $0F: Player characters. For example, $00 begins an action queue for Terra, and $01 begins one for Locke. $0E and $0F are special in that they begin action queues for temporary characters like Banon and Leo. Be careful, as the game often uses NPC lookalikes that resemble the player characters; those fall into the next category.
- $10 - $2F: Non-player characters. Different NPCs will take action depending on what map is loaded. You can cross-reference the Event Commands Document and FF6LE to figure out which command corresponds to the NPC that you want to move. According to the Event Commands Document, the $16 command will begin an action queue for NPC 6 in FF6LE. If you open the LE and select “[014] Narshe, Outside (WOB)”, you can see that NPC 6 is a guard on this map. If you select “[15D] Thamasa, Strago's House”, you can see that NPC 6 is a lookalike of Shadow on this map.





- $30: The camera. You can make the camera “walk” around the map, just like any other character. If you tell the camera to move upwards eight steps, the player's view will shift eight tiles upwards.
- $31 - $34: The characters in the party. Often, the player has the opportunity to pick a party of their choosing, so you won't know what characters are in the party during your event. By using the $31 command, you can move whomever happens to be leading the party. Figuring out which character is in the lead (and then moving them manually) would be a lot more difficult.

The Second Byte

The main purpose of the second byte is to tell the game how long the action queue will be. Most of the time, you should decide what actions you want to put in the queue before filling in this byte. You can have up to $7F (127 in decimal) actions in a queue, but it's unlikely that you will need anywhere near that many.

This byte also allows you to tell the game whether you want to finish the character's actions before moving on with the event. If you add $80 to this byte (after figuring out how long you want the queue to be), the game will wait for a character's actions to finish before moving on. Otherwise, the game will not wait. The latter is useful if you want to make multiple characters move at the same time, among other things.

The Actions

This is the main meat of the action queue, containing the actions that you want the character to perform. There is a list of actions here. The vast majority of these are self-explanatory. Many just tell the character to strike a pose or walk somewhere. The commands from $C0-$C4 allow you to change how fast the character walks when they move. Ignore the commands related to branching or setting event bits for now; these will be covered in later sections of the tutorial.

By default, a character will actually take footsteps when they are told to move somewhere. If you execute the $80 command (Move character up 1 tile), then the $82 command (Move character down 1 tile), the character will take one step upwards and one step downwards. If your intention was to make a character float up and down while facing forwards, these footsteps would need to be disabled by placing a $C7 command in the character's action queue. In other words, the middle of your action queue might contain the following commands: CE C7 80 82.

You should also take note of the E0 command. Delays tend to be very important in action queues because it takes a very short amount of time for characters to strike poses. Let's say that you want a character to shake their head back and forth. You may be tempted to alternate between $23 and $63. Unfortunately, the game will switch between $23 and $63 so quickly that the player won't be able to see it! Inserting a short delay between these poses will fix the issue. Instead of using 23 63 23 63, you might want to use 23 E0 01 63 E0 01 23 E0 01 63. Unfortunately, even the shortest delay ($01) is too long for some purposes. To create a very tiny delay, we can create four tiny action queues in a row, instead of creating one action queue with all four head-shaking poses in it. You will see an example of this below.

All action queues need to be ended with the $FF command. Once you have finished the body of your queue, don't forget to count how long it is and fill in the second byte. The $FF command at the end should also be included in this count.

Our First Action Queue

It's finally time to fix our problem from earlier. In order to prevent the event from repeating, we want the party to step off the event tile before the event ends. We could use $00 to begin an action queue for Terra, but we don't know who will be leading the party when the player steps on the event tile. If the player were to switch Biggs to the front of the party, our event would fall apart. Therefore, we should begin the queue with the $31 command.

We'll come back to the second byte later.

The body of this queue is very straightforward. We just want Terra to take a single step upwards. Looking at the Movement Action Commands, we can see that the $80 command will accomplish this. Finally, we need to complete the queue with $FF.

Now we need to fill in the second byte. Our action queue only contains two bytes ($80 and $FF), so we'll start with $02. We also want to wait for the party's movement to end before continuing the event, so we need to add $80, giving us a total of $82.

And that's it! The resulting action queue is 31 82 80 FF. As of now, this is the whole event:

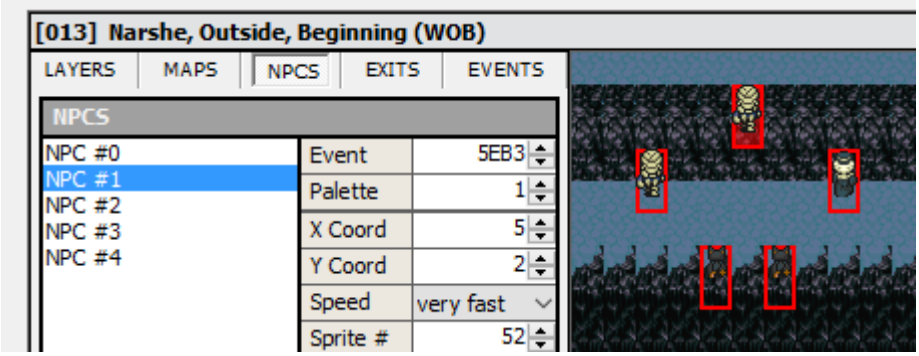
```
000C9B00  FE F4 D1 55 80 94 D2 B5 D2 33 D2 BF 4B 0A 00 B6  þôÑU€"ÒµÒ3Ò¿K..¶
000C9B10  18 9B 02 B3 5E 00 3A FE 4B D4 06 3A FE 4B 0C 00  .›.³^.:þKÔ.:þK..
000C9B20  55 20 92 4D 01 3F 96 31 82 80 FF FE FD FD FD FD  U 'M.?–1‚€ÿþýýýý
```

A Larger Example

Right now, the event uses a red flash to indicate that the guards are approaching. To improve on this, we can script some guards to physically approach the party. Since the battle includes two guards, we want two guards to be part of the event as well.

As before, our first goal is to determine which action queue to begin. The guards are NPCs, so we definitely need to use a command between $10 and $2F. To narrow it down further, we need to look in FF6LE. It might not be obvious at first glance, but the map that we need to look at is “[013] Narshe, Outside, Beginning (WOB)”. This map is the one loaded at the beginning of the game, and it is distinct from the map that is usually loaded when you enter Narshe.

The NPCs aren't easily noticed because they are all in the top-left hand corner. Spreading them out, we can see that there are two guards, two dogs, and an old man whose purpose isn't obvious. Since there are only two guards, it appears that they are reused for all the cutscenes in this area. We can follow the lead of the vanilla designers and do the same. Clicking on the guards, we can see that they are NPC #0 and NPC #1.



Now we can cross-reference this with the Event Commands Document, which states that we can use $10 to move NPC #0 and $11 to use NPC #1. We will need to create a separate action queue for each of the two guards.

First of all, we need to move the two guards to the bottom of the screen, which can be accomplished by using the $D5 command in the action queue. Playing around with the LE and the emulator, I determined that I want to place the guards at (37, 58) and (39, 58). Then I want the guards to walk onscreen from the south. You can use the LE to see the tiles that these coordinates point to. These coordinates become ($25, $3A) and ($27, $3A) when converted to hexadecimal. The final commands are D5 25 3A and D5 27 3A.

Presumably, the guards would be running towards the Magitek Armour, ready for attack. We should set their movement speed to “fast” with the $C3 command.

NPC #0

Let us say that NPC #0 is the one who will appear at (37, 58). This guard won't be able to approach all the way to the Magitek Armor because there is a steam wheel in his path. This means that we will need to break down his movements into several actions:

1. Take four steps upwards, approaching the steam wheel
2. Take one diagonal step up/right, in order to avoid the steam wheel
3. Take one more step upwards to get closer to the party.

The sequence of commands that can accomplish this is 8C A0 80.

Finally, we need to calculate the second byte. After some counting, we can determine that the action queue is eight bytes long. In this case, we do **not** want to add $80 to the second byte. We should not wait for this guard to finish walking because we want the second guard to move at the same time.

The entire action queue for the first guard is 10 08 D5 25 3A C3 8C A0 80 FF.

NPC #1

This guard could simply run up towards the party and stop somewhere. However, that would be boring, so let's spice things up a little. I want this guard to glance around furtively, as if he is looking around for more danger. This guard will follow these movements:

1. Walk up three steps
2. Turn left
3. Delay for a moment with the E0 command (so that the player can see the guard looking left)
4. Turn right
5. Delay for a moment
6. Walk up four steps

That's quite a few steps, but it's easy to piece together with the Movement Action Commands page as a reference. This sequence of actions is 88 CF E0 01 CD E0 01 8C. We can now determine the second byte to be $8D (we will wait for this guard's actions to complete before initiating the battle). The action queue for the second guard is 11 8D D5 27 3A C3 88 CF E0 01 CD E0 01 8C FF.

Finishing up