Creating Mods for ECWolf

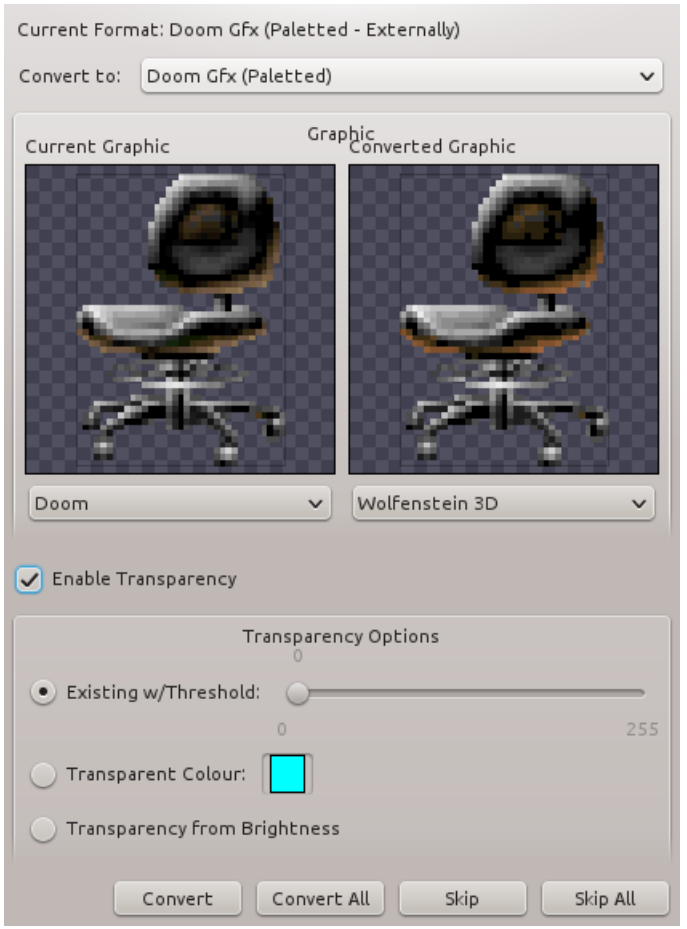
Part 2
- Working with sprites
- DECORATE scripts
- SNDINFO scripts

Sprites

- To add or replace sprites in ECWolf, simply place images named according to convention in the sprites/ directory of your PK3
  - If you are using a WAD file the sprites go between S_START/S_END markers
  - In a PK3, sprites can be organized into subdirectories freely. ECWolf also ignores the extension of files, so you may leave the .png, .lmp, etc intact

Sprites

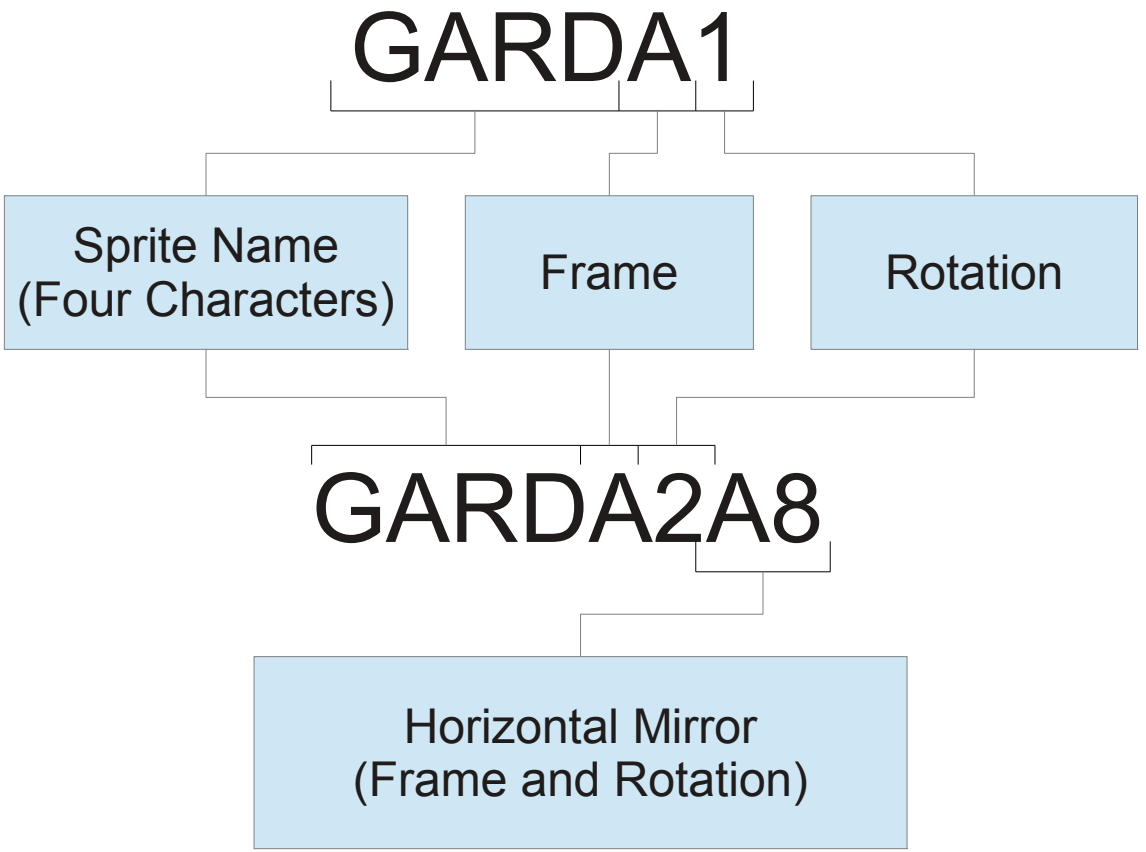
- Sprites in ECWolf can be of any size and may be in a variety of graphic formats. Usually either PNG or the Doom graphics format is used (Slade can convert other formats, such as BMP, to either of these)
  - PNGs may be truecolor, however, ECWolf uses an 8-bit renderer so the graphic will be matched to the palette at run time
  - Because of this, it may be advisable to use the Doom graphics format as after PK3 compression they will be smaller
- To convert and image, right click graphic entry and go to Gfx->Convert To



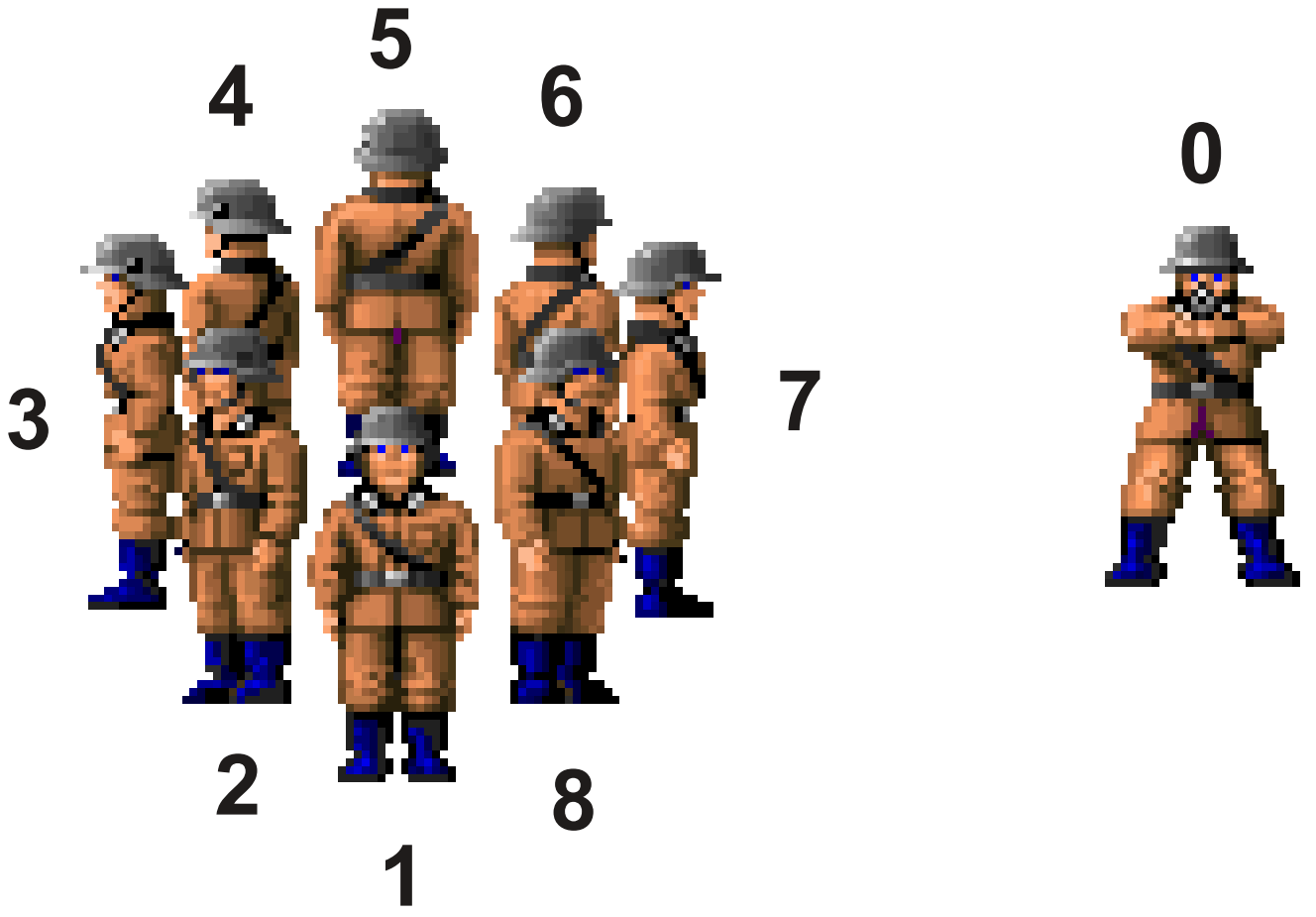
Sprite Names

- All sprites must have a four character name (Example: GARD)
- The name will be followed by a frame letter which can be A-Z
  - Frames '[', '\', and ']' are also usable, but their use is not recommended as they complicate things a bit
- The frame is followed by a number for the rotation 1-8 or the number 0 if the sprite should be used for all angles
- Another frame/rotation pair can be provided which will be used for automatically producing a horizontally mirrored sprite

Sprite Names

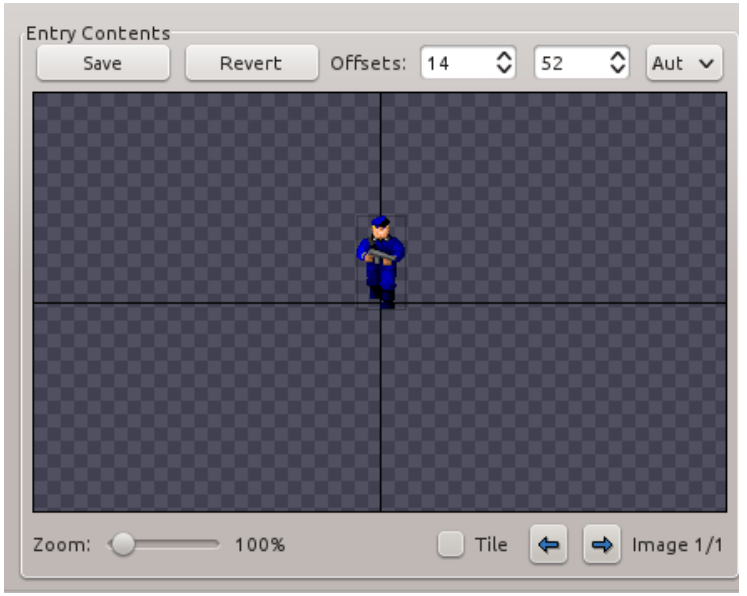


Sprite Angles



Sprite Offsets

- Since ECWolf imposes no restrictions on sprite size, the sprite needs to be offset in order to appear correctly
- Slade has a feature to automatically offset sprites, but they may need some manual tweaking after the fact
  - Right click sprite, Gfx->Modify Gfx Offsets
- Proper alignment is to have the monster's feet either slightly below the center line or right on it



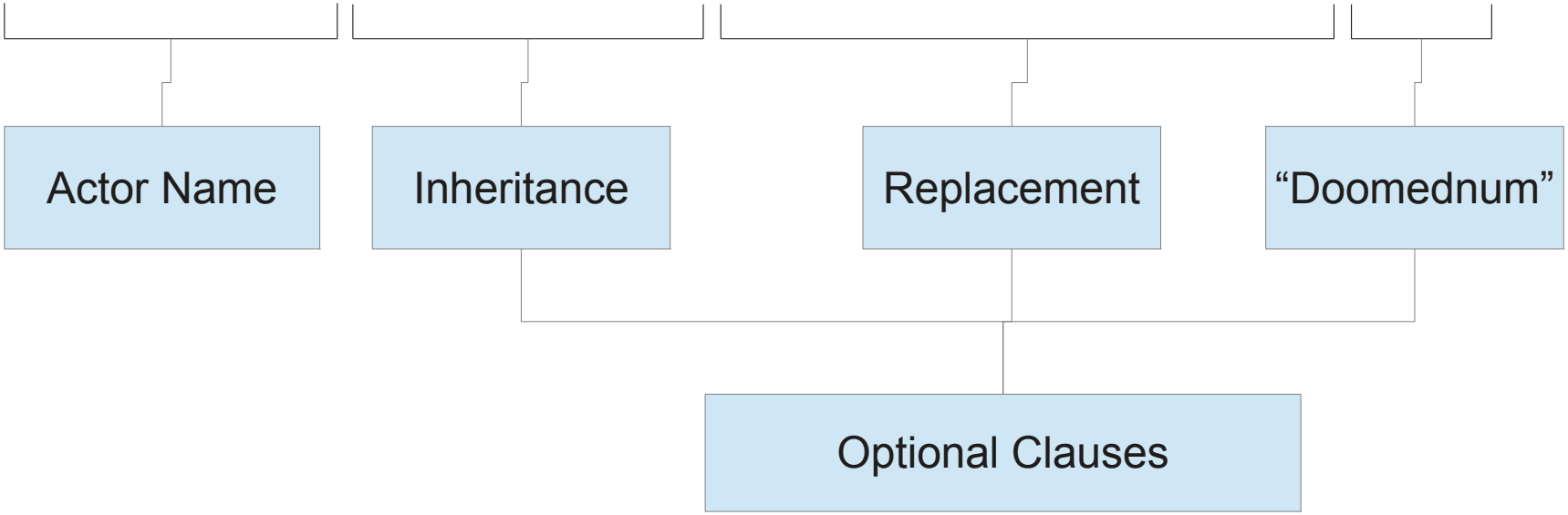
Intro to DECORATE

- DECORATE is an actor definition language
  - For our PK3, we'll want to create a decorate.txt
- The definition of an actor is somewhat loose as it includes decorations, enemies, inventory items, weapons, and more
- An actor consists of a set of properties, flags, and states of animation

Actor Syntax

- An actor starts with the following header. Do note that some parts are optional

actor MyIdentifer : OtherActor replaces AnotherActor 1234 { }



Actor Syntax

- Within the actor block we will provide a list of properties and flags
  - http://maniacsvault.net/ecwolf/wiki/Actor_properties
  - http://maniacsvault.net/ecwolf/wiki/Actor_flags
- A property takes a list of comma separated parameters
  - Syntax: propertyname param1, param2, …
  - Example: radius 32
- A flag is prefixed by either a '+' or '-' indicating whether to set or unset the flag
  - Example: +SOLID
  - There exist two flag combinations keywords which are actually parameterless properties. These are MONSTER and PROJECTILE. Their use and effects will be discussed later

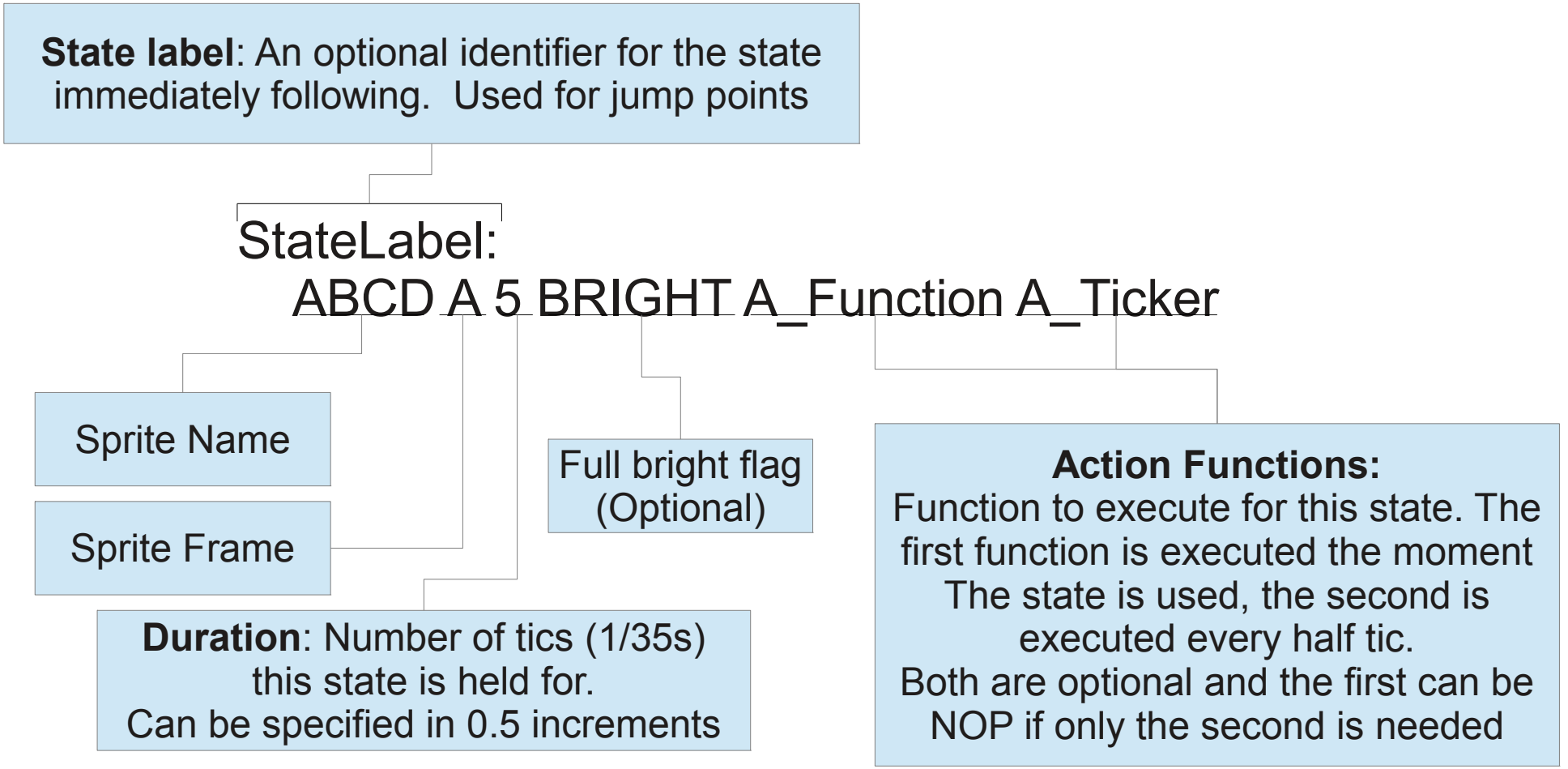
Actor Syntax

- The final part to an actor definition is the states (animation)
- To define them we use a states block inside the actor

```
actor MyActor {
    // Properties and flags
    states {
        // States go here
    }
}
```

State Syntax

- States use the following syntax. Again be aware that some parts are optional



State Syntax

- After a state's duration has been reached, the next state on the list will be shown unless otherwise specified
- The jumps available are:
  - loop
    - Go back to the previous state label
  - wait
    - Repeat the current state
  - goto StateLabel
    - Go directly to the specified label
  - stop
    - Remove the actor from play
    - Also used to remove states when inheriting from an actor (Label: stop)

State Syntax

- If a series of states vary only by the frame letter, the states can be shortened by simply listing the frames

Example:

```
ACTR A 4 NOP A_Chase
ACTR B 4 NOP A_Chase
ACTR C 4 NOP A_Chase
ACTR D 4 NOP A_Chase
loop
```

Example:

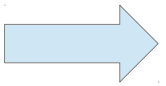
```
ACTR ABCD 4 NOP A_Chase
loop
```

- Note that this short hand still applies even if the 'wait' jump is used

Speaking of Shorthand

- In a way the second (ticker) function is a short hand. The following behave the same, although, unlike the previous example aren't completely identical

Example:
ACTR AAAAAAAAAA 0.5 A_Chase
loop



Example:
ACTR A 5 NOP A_Chase
loop

State Durations

- As stated before, ECWolf supports durations in 0.5 tic intervals (1/70th of a second)
- A duration of -1 signifies that the state should be held indefinitely
- A duration of 0 indicates that next state should be run *immediately*
  - Note that the action function is executed at the moment the state is entered so this can be used to combine multiple function calls
  - Beware of loops with 0 duration states as an infinite loop will cause a crash
  - Ticker functions will not be run on a 0 duration state as no tick passes

Basic States

- A few state labels have a special meaning assigned to them
  - Spawn
    - This is the state that will be shown immediately when the actor is spawned
    - Note: The action function for the first state will not be executed. This only applies immediately when the actor is spawned, if you use goto or loop the function will execute as expected! This can be worked around by making the first state have a 0 duration
  - See
    - State entered by monsters after targeting the player
  - Death
    - State entered when a monster reaches 0 health or a projectile hits a wall
  - XDeath
    - Entered when a monster has been overkilled or a projectile hits a bleeding actor

Basic States

- Pain
  - Entered when a monster has been hit and the painchance has been satisfied
- Missile
  - Monster's ranged attack sequence
- Melee
  - Monster's melee attack sequence

Putting It Together

- We will now create a destroyable decoration which replaces the brown plant in Wolfenstein 3D (1st solid decoration you see in E1L1)
- You will need to:
  - Create two sprites DCORA0 and DCORB0
  - Input the code on the next slide into your DECORATE lump

Putting It Together

```
actor MyDecoration replaces BrownPlant {
    radius 32
    health 100
    +SOLID
    +SHOOTABLE
    states {
        Spawn:
            DCOR A -1
            /* It has been something of a common practice to use stop after a -1
               duration even though it will never be reached */
            stop
        Death:
            // Use A_Fall to make the decoration become non-solid
            DCOR B -1 A_Fall
            stop
    }
}
```

Putting It Together

Putting It Together

- Additional challenges for this exercise:
  - Use 8 directional sprites for the decoration
  - Use additional states of animation for the actor's Death (or even an idle animation on Spawn)
- While I will be covering these topics can you figure them out on your own?
  - Give the thing a death sound (even an already existing sound will do) Hint: You will need to call A_Scream
  - Look at the code for a guard in Wolf3D, can you find the differences and make your decoration move?

    http://maniacsvault.net/ecwolf/wiki/Classes:Guard

Sound

- We have created an object which can be destroyed, but at the moment it just animates
- There are multiple ways to get an actor to play sounds, but first we must define the sounds for ECWolf
- ECWolf supports LPCM WAV files as well as Vorbis (OGG) for sound effects
- Sounds go in the sounds/ directory in your PK3

SNDINFO

- Just placing the sound file in the sounds directory is not enough, we need to assign a logical name to the sound
  - To add a little confusion here, logical sound names usually resemble paths
- Fortunately the syntax for SNDINFO (sndinfo.txt) is simple: logicalname lumpname
  - Example: mydecoration/death DSDECDTH
  - This would reference your dsdecdth file in the sounds directory
- The actors defined by ECWolf generally use unique logical names for everything so SNDINFO by itself can be used to give sounds to actors that reused sounds!

SNDINFO

- There are, however, more advanced things that can be done with SNDINFO
  - $alias newlogicalname otherlogicalname
    - Defines a new sound as the same as an old sound, this is useful so that you can give multiple logical names to a single lump either allowing for more specific replacements later or reducing the amount of work if you need to rename a sound file
  - $random logicalname { logical1 logical2 … }
    - Plays one of the sounds listed at random. A sound can be specified multiple times to give one sound weight

Assigning Sounds

- The most basic way to have an actor play a sound is to simply use the action function A_PlaySound(“logicalname”)
- There are, however, properties such as deathsound, seesound, painsound, etc that can be used
  - The properties allow the engine to play sounds on certain events automatically, but often they use their own action pointers to play them
    - For example A_Scream is used to play the deathsound!

Modifying the Example

- First get a sound called dsdecdth.wav in your sounds directory
- Create a sndinfo.txt file in the root
- Add the following to the file:
  mydecoration/death DSDECDTH
- Change our example actor to the code on the following slide

Modifying the Example

```
actor MyDecoration replaces BrownPlant {
    radius 32
    health 100
    deathsound “mydecoration/death”
    +SOLID
    +SHOOTABLE
    states {
        Spawn:
            DCOR A -1
            /* It has been something of a common practice to use stop after a -1
               duration even though it will never be reached */
            stop
        Death:
            // We use a 0 duration state to call this at the same time as A_Fall
            DCOR B 0 A_Scream
            // Use A_Fall to make the decoration become non-solid
            DCOR B -1 A_Fall
            stop
    }
}
```

Monsters

- Now that we've made some static objects, lets take a look at monsters
- Since we haven't looked at inheritance yet, we're going to create Doom like monsters for now. These are a little simpler to make, but the movement won't be as smooth
- Here I'll be using sprites from the original game, but feel free to substitute your own. Just be sure the sprite names and frames are adjusted accordingly!
- Keep the action function reference on hand since I'll be referring to a lot of them. The wiki explains the usage of all of them
  - http://maniacsvault.net/ecwolf/wiki/Action_functions

Monsters

- There isn't anything particularly special about a monster. In general they have a certain set of flags that should be set (shootable, solid, counts for kills, etc) so the MONSTER keyword will set them for you
- From there we will use the Spawn state to look out for the player with A_Look
- The See state will then be used to chase after the player with A_Chase
- For an attack we can create a Missile or Melee state (you can even define both!)
  - At the end of this state sequence, you will want to use 'goto See' in order to resume the chase sequence
  - A typical ranged attack uses A_WolfAttack or A_CustomMissile
  - The Melee attack will often use A_MeleeAttack
- Optionally the monster can drop an item (dropitem property) for the player and/or give them points (points property)

Example Monster

```
actor ExampleMonster replaces DeadGuard { // Replaces the dead guard at the start of E1L1
    points 100
    health 50
    painchance 127 // Go to the pain state on about half of the hits
    dropitem “Clip”, 200, 25 // Drop 25 bullets most of the time
    deathsound “guard/death”
    seesound “guard/sight”
    activesound “guard/sight”
    painsound “guard/death”
    attacksound “guard/attack”
    MONSTER
    states {
        Spawn:
            GARD A 4 A_Look
            wait
        See:
            GARD BBCCDDEE 4 A_Chase
            loop
        Missile:
            // We use A_FaceTarget here in case we have directional sprites
            GARD FG 10 A_FaceTarget
            GARD H 10 BRIGHT A_WolfAttack
            GARD G 10 A_FaceTarget
            GARD H 10 BRIGHT A_WolfAttack
            goto See
        Pain:
            GARD I 5 A_Pain
            goto See
        Death:
            GARD K 7 A_Fall
            GARD L 7 A_Scream
            GARD M 7
            GARD N -1
            stop
    }
}
```

Conclusion

- We have looked at how to define static objects and monsters in DECORATE, but there is much more left to look at
  - Creating inventory items, including weapons
  - Creating new player classes
  - And more
- Mostly from here out we'll have to use inheritance to access new actor behaviors
- But, we also haven't looked at how to create new things without replacing old ones. Until a UWMF compatible map editor gets created we'll have to learn how to write map translators