Repeater Cannon

By Ivan Mogilko, 25 July 2006

The most easy way to modify a weapon in VavoomC – is to change some of its attributes, damage for example. But this way is not *simple*, it is ***primitive***. And I wish to show some nice coding here, that will be rather simple and effective at the same time.

After minutes of thinking I decided to create a Repeater Cannon.

In this tutorial following files will be modified:

- progs\doom\game1\WeaponChaingun.vc
- progs\doom\game1\WeaponMissile.vc

Task overview

First of all, as always, we state our goal. In this case it would be creation of a weapon for Doom/Doom2 game that will fire rockets at chaingun's firerate. There are two possible and most obvious ways to do this and I am going to demonstrate both. At first, I'll create this crazy weapon basing on Chaingun code, secondly – starting with Rocket Launcher code. Results will be similar, but there'll be difference in code we'll have to add.

Method One: Modifying Chaingun

Let's open 'WeaponChaingun.vc'. It contains the **WeaponChaingun** class which is a child of the **Weapon** class, and gives chaingun all the functionality needed. It has only one function – **A_FireCGun()**, that does the single shot.

There is something specific in how chaingun shoots. As you can remember from game, chaingun fires automatically while 'fire' key is pressed.

Let's look on the "__states__" section of the class description. It shows all the states this weapon has. If you slip your view a bit lower, into "defaultproperties" section, you'll notice following string:

```
AttackState = S_CHAIN1;
```

This means that when commanded to attack (fire) chaingun will be set to S_CHAIN1 state.

```
S_CHAIN1('CHGG', 0, 'weapons/v_chain.md2', 3, 4.0 / 35.0, S_CHAIN2) { A_FireCGun(); }
S_CHAIN2('CHGG', 1, 'weapons/v_chain.md2', 4, 4.0 / 35.0, S_CHAIN3) { A_FireCGun(); }
S_CHAIN3('CHGG', 1, 'weapons/v_chain.md2', 4, 0.0 / 35.0, S_CHAIN) { A_ReFire(); }
```

What is important for understanding chaingun's fire process, is that being set to S_CHAIN1 state chaingun runs A_FireCGun function and automatically sets itself to S_CHAIN2 state, which runs A_FireCGun function once more and sets chaingun to S_CHAIN3 state, which finally runs **A_ReFire()** function.

NOTE: for more precise information about states read the States Tutorial.

A_Refire function is a member of Weapon class, a parent class for all weapons that player can handle. Let's look into its source ('progs\doom\game\Weapon.vc') and find this function.

```
void A_ReFire()
```

```
{
  // check for fire
  // (if a weaponchange is pending, let it go through instead)
  if ((Player.Buttons & BT_ATTACK) && Player(Player).PendingWeapon == DoomDefs::wp_nochange && Player.Health)
  {
    Player(Player).Refire++; Player(Player).FireWeapon();
  }
  else
  {
    Player(Player).Refire = 0; Player(Player).CheckAmmo();
  }
}
```

Generally, as you may see, this function checks whether 'fire' key is still pressed by user. If 'fire' key is pressed, then A_Refire calls **Player::FireWeapon()** function which sets current weapon to its **AttackState**. If 'fire' key is not pressed any more, then shooting is over.

In other words, while user helds 'fire' key pressed, those three states above (S_CHAIN1, S_CHAIN2 and S_CHAIN3) will repeat again and again infinitely (that's in theory, in practice – until all ammo is spent). Smart person may also notice, that on single command chaingun will always make two shots.

Now, when we know how chaingun is shooting, let's modify *what* does it shoot. In A_FireCGun function following line does real shot:

```
GunShot(Actor(Player.MO), !Player(Player).Refire, dir);
```

The way this **GunShot()** works is not matter of concern now, because bullets are fired somehow different than rockets. Thus we should simply remove it. Plus we should remove following lines here since they will be unnecessary:

```
TVec dir;
```

```
dir = Actor(Player.MO).Aim(16.0 * 64.0);
```

All missiles, like rockets in our case, are shot using Player::SpawnPlayerMissile() function. It takes only one parameter – a classid (class name) of an object we want to spawn. Since we want our chaingun to fire rockets, call it in the place where GunShot was earlier:

```
if (State == AttackState)
  Player(Player).SetPsprite(ps_flash, FlashState);
else
  Player(Player).SetPsprite(ps_flash, S_CHAINFLASH2);
Player(Player).SpawnPlayerMissile(Rocket); // <--- our new function call
Player.MO.Effects |= DoomDefs::EF_DL_MUZZLEFLASH;
```

Now chaingun will fire rockets instead of bullets.

Save your work, compile progs, run Vavoom and test your modified chaingun.

And... err... take care.

Method Two: Modifying Rocket Launcher

Let us open 'WeaponMissile.vc' now. WeaponMissile class works for rocket launcher and its "shooting" function is A_FireMissile(). It is very simple and has only two lines of code.

```
void A_FireMissile()
{
  Player(Player).Ammo[Ammo]--;
  Player(Player).SpawnPlayerMissile(Rocket);
}
```

I suppose it is obvious that first line of code decreases quantity of rockets player carries by 1 and second line “shoots” a rocket.

```
S_MISSILE1('MISG', 1, 'weapons/v_launch.md2', 3, 8.0 / 35.0, S_MISSILE2) { A_GunFlash(); }
S_MISSILE2('MISG', 1, 'weapons/v_launch.md2', 4, 12.0 / 35.0, S_MISSILE3) { A_FireMissile(); }
S_MISSILE3('MISG', 1, 'weapons/v_launch.md2', 5, 0.0 / 35.0, S_MISSILE) { A_ReFire(); }
```

AttackState variable is set to S_MISSILE1 in “defaultproperties” section. As you already know, this means that when weapon is ordered to fire, S_MISSILE1 state is set. S_MISSILE1 state calls **A_GunFlash()** function, then state S_MISSILE2 starts, which calls A_FireMissile and sets S_MISSILE3 state which call A_Refire. If player is still holding ‘fire’ key, next state is S_MISSILE1 again, otherwise it is S_MISSILE state, which simply re-readies the weapon.

All seem very similar to WeaponChaingun class, except for A_GunFlash function which actually is not important for us now. (If you wish to know what is it doing, check ‘Weapon.vc’.) So, what is the real difference in shooting process? Answer is – fire rate, that is frequency by which states are shifting. Each state has its delay parameter which tells how much this state last, it is the fifth parameter in state’s round brackets. Delay is usually defined as a fraction of 35.0. Delay equal to 1.0 (one) will last one second. Delay defined as 17.5 / 35.0 will last about half of the second, and so on.

To make rocket launcher shoot as fast as chaingun, we must

1. Replace A_GunFlash function call at S_MISSILE1 state with extra A_FireMissile call,
2. Modify ‘delay’ parameters of all three states mentioned correspondingly.

Resulting code should be looking like this:

```
S_MISSILE1('MISG', 1, 'weapons/v_launch.md2', 3, 4.0 / 35.0, S_MISSILE2) { A_FireMissile(); }
S_MISSILE2('MISG', 1, 'weapons/v_launch.md2', 4, 4.0 / 35.0, S_MISSILE3) { A_FireMissile(); }
S_MISSILE3('MISG', 1, 'weapons/v_launch.md2', 5, 4.0 / 35.0, S_MISSILE) { A_ReFire(); }
```

And that is enough for our current task.

Adjusting proper ammunition usage

If you tested Method One above, you probably noticed that although chaingun is shooting rockets, it still uses bullets ammunition. To correct this, let’s open ‘WeaponChaingun.vc’ once again and look onto “defaultproperties” section. There is a **Ammo** variable being set a value:

```
Ammo = DoomDefs::am_clip;
```

Ammo variable is an index of ammunition the weapon is using. It must be set to one of the presets, declared in ‘progs\doom\doomdefs.vc’.

```
// Ammunition types defined.
enum
{
  am_clip, // Pistol / chaingun ammo.
```

```
  am_shell, // Shotgun / double barreled shotgun.
  am_cell, // Plasma rifle, BFG.
  am_misl, // Missile launcher.
  #ifdef DDF am_pellet, am_nail, am_grenade, am_gas,
  #endif
  NUMAMMO,
  am_noammo // Unlimited for chainsaw / fist.
};
```

We should change it to value, that rocket launcher uses:

```
Ammo = DoomDefs::am_misl;
```

Now our chaingun not only fires rockets, but uses rocket ammo.