David J. Sushil davidjsushil@gmail.com

Simple AI in the Torque Game Engine

Torque excels at creating first-person shooter games. And where would any respectable FPS be without deadly accurate AI to challenge the player? This tutorial presents a very basic example of how artificial intelligence can be utilized in the Torque Game Engine. It takes a "least amount of code" approach to AI programming, a topic which is covered in much greater detail in any number of excellent books and websites.

Overview

The code we will work on in this exercise is only one possible way of adding AI to your project. I state this only to alleviate any future frustrations you might encounter in using the code provided below. It's a start – a jumping off point for your future masterpieces – but it's not a total solution. In many cases though, it should work in a pinch.

What we want to create is a system where AI bots can be spawned with a single line of script, which, for example, could be called when the mission begins, or when the player passes through a trigger.

We also want to establish a system that allows for multiple AI styles. That is, if all bots acted the same way, the game would become boring. This tutorial will demonstrate how we can control (and even switch) AI styles.

Finally, we need a way to periodically update our AI. Thankfully, Torque makes this easy with the schedule function, which we will utilize in this example.

Adding a Bot

Torque comes with preset dummy characters known as PlayerBodies. These are represented by Torque's poster boy, Kork the Orc (quite a sense of humor there at GarageGames, eh?). Much of the PlayerBody functionality can be found in aiPlayer.cs, a file located in the demo/server/scripts/ folder. I recommend looking over this file for additional help with activites such as pathfinding. Since our AI bots are more of a deterministic variety, we'll forego much of this file for a more simplistic approach.

To add a bot, use the following code:

```
%bot = new AIPlayer(){
      datablock = PlayerBody;
      position = "100 100 100";
};
```

In this way, the ID number of the bot will be stored in %bot, we inherit characteristics from the PlayerBody datablock, and place the bot at 100, 100, 100 on the map. Of course, that position needs to be updated to reflect your particular level. This will spawn an instance of Kork in the level, however he won't do much. He just kinda starts there, all awkward and stuff.

Let's take this code a bit further and add it to a custom function, called addBot. AddBot takes four arguments, an AI style, and position information.

```
function addBot(%style, %x, %y, %z){
      %bot = new AIPlayer(){
            datablock = PlayerBody;
            position = %x @ " " @ %y @ " " @ %z;
            botID = 0;
            style = %style;
      };
}
```

Now, if you want to add a bot to your level, you can simply call the addBot function with a style and position for each bot, like so:

```
addBot(0, 100, 100, 100);
```

Still, no action on the part of Kork. But we have added two new properties. The first is *botID*, which we will use throughout the exercise. Currently, it's set to zero. After the new block that creates our bot, add the following line:

```
%bot.botID = %bot;
```

Now, we can more easily keep track of the bot.

The second property we've added is *style*, which we will use to control how the bot behaves.

Finally, let's give Kork a crossbow. Below the new block that creates the bot, add the following line:

```
%bot.mountImage(CrossBowImage, 0);
```

Still, no Kork just sits there. That's where the updateBot function comes in handy. We'll write the function in the next section, but for now, add the following line:

```
updateBot(%bot.botID);
```

Updating the AI

The updateBot function takes one argument, the ID number of an AI bot. In this function, we'll make sure the agent always has ammunition for the crossbow. Also, we'll determine how far the bot is from the player. Then, we'll execute an IB behavior based on style. Finally, we'll schedule the updateBot function to be called sometime in the near future.

First, let's create a new function called updateBot, like so:

```
function updateBot(%id){
      // CODE GOES HERE
}
```

Within the function, you can give the bot ammunition for the crossbow by calling:

```
%id.incInventory(CrossbowAmmo, 1);
```

This ensures at least one shot per update, however depending on your implementation, you may want to adjust the number of shots between executions.

Next, we need to determine how far away the player and the bot are from each other. This is sadly more complicated than it needs to be in Torque. The engine stores position information as a single string, instead of three separate numerical values. To parse out each x-, y-, and z-coordinate, we have to use the getword function, like so:

```
    %botX = getword(%id.getTransform(),0);
    %botY = getword(%id.getTransform(),1);
    %botZ = getword(%id.getTransform(),2);
    %playerX = getword($gspPlayer.getTransform(),0);
    %playerY = getword($gspPlayer.getTransform(),1);
    %playerZ = getword($gspPlayer.getTransform(),2);
```

Finally, we can use the values stored here to determine a distance:

```
    %distance = mSqrt((%playerX - %botX)*(%playerX - %botX) + (%playerY
- %botY)*(%playerY - %botY));
```

For the moment, we'll skip executing AI behaviors, and instead look at how we can recall this function automatically. We do so by using the schedule function, like so:

```
schedule(200, 0, updateBot, %id);
```

The first argument in this function represents how many milliseconds to wait before calling the third argument, which in our case is the name of our updateBot function. Any arguments after the third represent arguments for the function to be called (hence %id, which is the sole argument for updateBot). So, a translation of this statement in plain English would read "In 200 milliseconds, call updateBot for the bot indicated by %id."

So now we have a function that gives the bot ammo, determines its distance from the player, and schedules another update in the future. In the next section, we'll explore AI behaviors.

Styles

After we determine a distance from the player, we will steer the behavior of the bot according to its style. We will add three styles of AI to this example – a grunt, which pursues the player, and when in close range, stops to fire; a sniper, which remains stationary and fires if the player is within a wide range; and a berserker, who always pursues the player, firing as it goes regardless of distance.

Within the updateBot function, we will execute these behaviors using a switch statement, which looks like this:

```
switch (%id.style){
      // CASES GO HERE
}
```

Within the switch statement, the property %id.style will be evaluated, and a case will be selected. Each case represents a particular style of bot AI. Going in reverse order, the berserker's case looks like this:

```
case 2: // BERSERKER AI
      %id.fire(true);
```

```
      %id.setMoveDestination($gspPlayer.getTransform());
      %id.setAimObject($gspPlayer);
```

The fire function, which takes a Boolean value for an argument, determines whether the bot is firing or not. It is not a request for a single shot, but rather a toggle switch that sets the bot to fire indefinitely, or remain passive. The setMoveDestination function takes the player's position as an argument, and tells the bot to move towards that spot on the map. Finally, the setAimObject function tells the bot to always aim towards the player.

This is a very simple AI bot, which always follows the player, and always shoots. Though simply, he's pretty darn lethal, and you might want to consider going easy on the number of berserkers in your game!

A more realistic opponent is the grunt:

```
case 0: // GRUNT AI
      if (%distance < 15){
            %id.fire(true);
            %id.setMoveDestination(%id.getTransform());
      }else if (%distance > 30){
            %id.fire(false);
            %id.setMoveDestination($gspPlayer.getTransform());
      }
      %id.setAimObject($gspPlayer);
```

As you can see, here we are utilizing the distance variable we created previously. If the player is more than 30 units away, put your gun away and chase him; if he is less than 15 units away, stop and shoot.

Finally, we have the sniper, who would do well to stay out of range of the player – perhaps in on the top of a building somewhere:

```
case 1: // SNIPER AI
      if (%distance < 100){
            %id.fire(true);
      }else{
            %id.fire(false);
      }
      %id.setAimObject($gspPlayer);
```

This case tells the AI, if the player is within 100 units, fire; otherwise, put the gun away and wait.

The Player's ID

Finally, you may have noticed a global variable in there called $gspPlayer. We need to add this variable ourselves. Although there are other ways of tracking the player's ID, I find this method most useful.

Open the demo/server/scripts/ folder, and edit the file named game.cs. Find the createPlayer function, and add the following boldfaced line, just below the new Player command, and before the call to MissionCleanup:

```
// Create the player object
%player = new Player() {
```

```
      dataBlock = PlayerBody;
      client = %this;
};
$gspPlayer = %player;
MissionCleanup.add(%player);
```

Now, whenever you want to refer to the player, you can use the global $gspPlayer variable.

Complete Code

Because we're dealing with quite a bit of code, here's the total package. I would recommend placing it in a single .cs file, and adding a reference to it in the onServerCreated function, within demo/server/scripts/game.cs.

```
// FUNCTION FOR ADDING A SINGLE BOT AT (X,Y,Z) ON THE MAP
function addBot(%style, %x, %y, %z){
      %bot = new AIPlayer(){
            datablock = PlayerBody;
            position = %x @ " " @ %y @ " " @ %z;
            botID = 0;
            style = %style;
      };
      %bot.mountImage(CrossBowImage, 0);
      %bot.botID = %bot;
      updateBot(%bot.botID);
}

// UPDATES THE BOT AI
function updateBot(%id){
    %id.incInventory(CrossbowAmmo, 1);
    %botX = getword(%id.getTransform(),0);
    %botY = getword(%id.getTransform(),1);
    %botZ = getword(%id.getTransform(),2);
    %playerX = getword($gspPlayer.getTransform(),0);
    %playerY = getword($gspPlayer.getTransform(),1);
    %playerZ = getword($gspPlayer.getTransform(),2);
    %distance = mSqrt((%playerX - %botX)*(%playerX - %botX) + (%playerY
- %botY)*(%playerY - %botY));
    switch (%id.style){
        case 0: // GRUNT AI
            if (%distance < 15){
                %id.fire(true);
                %id.setMoveDestination(%id.getTransform());
            }else if (%distance > 30){
                %id.fire(false);
                %id.setMoveDestination($gspPlayer.getTransform());
            }
            %id.setAimObject($gspPlayer);
        case 1: // SNIPER AI
            if (%distance < 100){
                %id.fire(true);
            }else{
                %id.fire(false);
            }
            %id.setAimObject($gspPlayer);
        case 2: // BERSERKER AI
            %id.fire(true);
```

```
            %id.setMoveDestination($gspPlayer.getTransform());
            %id.setAimObject($gspPlayer);
    }
    schedule(200, 0, updateBot, %id);
}
```