

Tutorial 6: Single-Use TimerTasks

Contributed by D1st0rt
 Saturday, 25 February 2006
 Last Updated Saturday, 25 February 2006

This is the first of two tutorials to demonstrate the use of TimerTasks. In this one we will be dealing with Single-Use TimerTasks, meaning tasks that execute only once some time in the future. TWCore provides an easy way to manage these TimerTasks through the BotAction class, which I will attempt to illustrate in this tutorial with our good friend mybot. Our mybot module welcomes players whenever they enter the arena, but we only want it to greet players that have entered the arena for the first time in the last 10 minutes. Since parts of the mybot module stay the same, I'm only going to show changed sections of code in this tutorial.

```

//Removed some of the older comments to make the code cleaner
//now that you should know the reasoning behind it all
package twcore.bots.twbot;

import java.util.HashSet;
import java.util.Vector;
import java.util.TimerTask;
import twcore.core.*;

public class twbotmybot extends TWBotExtension
{
    /** Keeps track of the recent players to enter */
    private HashSet<String> recentPlayers;
    /** Holds onto all of the current tasks running */
    private Vector<RemovePlayer> activeTasks;
    /** Module Help message */
    private final String[] helpMsg = {
        "[mybot]",
        "!welcome - welcomes people"
    };

    /**
     * Creates a new instance of the mybot module
     */
    public twbotmybot()
    {
        recentPlayers = new HashSet<String>();
        activeTasks = new Vector<RemovePlayer>();
    }

    /**
     * Stops any running tasks so the module can be unloaded.
     */
    public void cancel()
    {
        for(RemovePlayer task : activeTasks)
            task.cancel();
    }
}

```

There are some new imports for this module, being the HashSet and Vector data structures from the java.util package. These will allow us to easily store some of the dynamic data we need for this module. I also added the TimerTask import for obvious reasons. In the constructor, I instantiated my HashSet and Vector. Because they are objects and not primitive types like an int or a char, they must be created before they can be used. The cancel method is important to have in a module because if you do not cancel all of your tasks they will keep running even after the module is unloaded, which can have some undesired effects.

```

/**
 * Event: PlayerEntered
 * Called when a player enters the arena
 * If the player enters the arena for the first time in the past 10 minutes,
 * welcome them to the arena.

```

```

*/
public void handleEvent(PlayerEntered event)
{
    String name = event.getPlayerName();
    if(!recentPlayers.contains(name))
    {
        m_botAction.sendPrivateMessage(name,"Welcome!");
        recentPlayers.add(name);
        RemovePlayer task = new RemovePlayer(name);
        activeTasks.add(task);
        m_botAction.scheduleTask(task, 600000);
    }
}
}

```

Here is our new and improved PlayerEntered event handling. Once we get the player's name, we check to see if they have been in the arena in the last 10 minutes by whether they are contained in our set of recent players. If they aren't, we need to welcome them, add them to our recent players, and start a task to remove them from the recent players set after 10 minutes. We don't do anything if they are in the recent player set because that means they previously entered the arena less than 10 minutes ago.

```

/**
 * This class represents a task to remove a player from the list of
 * players recently entering the arena after 10 minutes
 */
private class RemovePlayer extends TimerTask
{
    /** The name of the player */
    private String name;

    /**
     * Creates a new instance of the RemovePlayer task
     * @param name the name of the player
     */
    public RemovePlayer(String name)
    {
        this.name = name;
    }

    /**
     * Execute this task: remove the player from the recent players list
     */
    public void run()
    {
        recentPlayers.remove(name);
        activeTasks.remove(this);
    }
}
}

```

Whenever you want to use a TimerTask, having a task to execute is usually fairly important. We put our class RemovePlayer that extends TimerTask inside of our module class so that it can access the recent player set, and we make it private so that no other class has access to it. When we create a new RemovePlayer task, we specify the name of the player we want to remove when the task executes. The run method is what is called when it is time for the task to do it's thing. In this case, it removes the player from the recent players and removes this task from the list of active tasks. Because a TimerTask runs on a separate thread from the bot, it is a good idea to use the Vector class to store tasks over the ArrayList class because the latter is not thread-safe.