Build The HTML Page

```
<!DOCTYPE HTML PUBLIC "-
//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/str
ict.dtd">
```

```
<html>
<head>

        <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">

        <title>Artificial intelligence : OpenKore source
code documentation</title>

        <link rel="stylesheet" type="text/css"
href="openkore.css">

        <!-- Fix broken PNG transparency for IE/Win5-6+ --
>

        <!--[if gte IE 5.5000]>

        <script type="text/javascript"
src="pngfix.js"></script>

        <![endif]-->


        <style type="text/css">

        <!--

        .example {
         margin: 0.3cm;
         margin-left: 0.5cm;

        }

        .comment {

                font-style: italic;
        }
```

```
        .term {
         border-bottom: 1px dotted black;

        }

        .cstr {
         color: #007700;

        }

        -->

        </style>

</head>


<body>


<div id="title">OpenKore source code documentation</div>


<div id="navigation">

        <ul>

        <li><a
href="http://openkore.sourceforge.net/">Main
website</a></li>

         <li><a href="index.html">Table of
contents</a></li>

        <li><b>Artificial intelligence</b></li>

        </ul>

</div>


<div id="main">
```

```
<h1>How the AI subsystem is designed</h1>


The AI subsystem isn't really complex, but it could take a
while to understand it's design.


<p>

All "intelligence" is handled inside the
<code>AI()</code> function (right now it's one big
function but
```
we hope to split it in the future).
```
As explained in the <a>Main loop &amp; initialization</a>
page, the <code>AI()</code> function only runs less than
a fraction of a second.


<p>

Basically, the AI tells Kore to do certain things based on
the current situation. I'll try to explain it with some
examples.




<a name="ex1"></a>

<h2>Example 1: Random walk</h2>


You're probably familiar with Kore's random walk feature.

If there are no monsters and Kore isn't doing anything, it
will walk to a random spot on the map, and attack any
monsters it encounters.

The following piece of code (within the <code>AI()</code>
function makes Kore walk to a random spot if it isn't
doing anything:

<pre class="example">
```

```perl
1          <span class="comment">##### RANDOM WALK
           #####</span>
2          <b>if</b> ($config{'route_randomWalk'} &&
           $ai_seq[0]
<b>eq</b> "" && @{$field{'field'}} > 1 &&
!$cities_lut{$field{'name'}.'.rsw'}) {

3          <span class="comment"># Find a random block
           on the map that we can walk on</span>

4          <b>do</b> {

5          $ai_v{'temp'}{'randX'} = int(rand() *
($field{'width'} - 1));

6          $ai_v{'temp'}{'randY'} = int(rand() *
($field{'height'} - 1));

7          } <b>while</b>
($field{'field'}[$ai_v{'temp'}{'randY'}*$field{'width'} +
$ai_v{'temp'}{'randX'}]);

8

9      <span class="comment"># Move to that block</span>

10     message <span class="cstr">"Calculating random
       route to:
$maps_lut{$field{'name'}.'.rsw'}($field{'name'}):
$ai_v{'temp'}{'randX'}, $ai_v{'temp'}{'randY'}\n"</span>,
<span class="cstr">"route"</span>;

11     ai_route(\%{$ai_v{'temp'}{'returnHash'}},

12         $ai_v{'temp'}{'randX'},

13         $ai_v{'temp'}{'randY'},

14         $field{'name'},

15         0,

16         $config{'route_randomWalk_maxRouteTime'},

17         2,

18         undef,

19         undef,

20         1);
```

```
21  }
```

```
</pre>
We call this block of code an <em class="term">AI code
block</em>.
```

In other words, an AI code block is an entire block of
   code which deals with a certain part of the AI.

```
<h3>Situation check</h3>
```

```
In line 1, it checks:
```

```
<ol>
```

```
<li>whether the configuration option
<code>route_randomWalk</code> is on</li>
```

```
<li>whether there are currently no other active <em
class="term">AI sequences</em> (see below)</li>
<li>whether we're currently NOT in a city </li>
```

```
</ol>
```

```
If all of the above is true, then Kore will run the code
inside the brackets.
```

```
<p>
```

```
What is an <em class="term">AI sequence</em>? It is a
value within the <code>@ai_seq</code> array.
```

```
This array is a <em>command queue</em>.
```

```
<p>
```

```
AI code blocks prepend values into this array so they can
know when it's their turn to do something.
```

```
When an AI code block is done with it's task, it will
remove that value from the array.
```

So, if `<code>@ai_seq</code>` is empty, then that means all
AI code blocks have finished and Kore isn't doing
anything else.

And this is when the random walk AI code block jumps in.

<p>

There is also the `<code>@ai_seq_args</code>` array, used to
store temporary variables used by the current AI code
block.

If a value is prepended into `<code>@ai_seq</code>`, then a
value must also be prepended into
`<code>@ai_seq_args</code>`.

More on this later.

<h3>Finding a random position to walk to</h3>

Line 4-7 tries to find a random position in the map that
you can walk on.

(`<code>$field{field}</code>` is a reference to an array
which contains information about which blocks you can and
can't walk on.

But that's not important in this example. You just have to
understand what this block does.)

<p>

The result coordinate is put into these two variables:

`<code>$ai_v{temp}{randX}</code>`
`<code>$ai_v{temp}{randY}</code>`

<small>(In case you didn't know, <code>$foo{bar}</code> is the same as <code>$foo{'bar'}</code>.)</small>

Moving

Line 11-20 is the code which tells Kore to move to the random position.

It tells <code>ai_route()</code> where it wants to go to.

<code>ai_route()</code> prepends a <code>"route"</code> AI sequence in <code>@ai_seq</code>, and arguments in a hash

(which is then prepended into <code>@ai_seq_args</code> and immediately returns.

Shortly after this, the entire <code>AI()</code> function returns.

The point is, <code>ai_route()</code> is not synchronous.

<p>

In less than a fraction of a second, the <code>AI()</code> function is called again.

Because the <code>@ai_seq</code> variable is not empty anymore, the random walk AI code block is never activated

(the expression <code>'$ai_seq[0] eq ""'</code> is false).

<p>

The AI code block that handles routing is elsewhere in the <code>AI()</code> function.

It sees that the first value in <code>@ai_seq</code> is <code>"route"</code>, and thinks "hey, now it's my turn to do something!".

(The route AI code block is very complex so I'm not going to explain what it does, but you get the idea.)

When the route AI code block has finished, it will remove the first item from <code>@ai_seq</code>.

If <code>@ai_seq</code> is empty, then the random route AI code block is activated again.

<h2>Example 2: Attacking monsters while walking to a random spot</h2>

You might want to wonder how Kore is able to determine whether to attack monsters when it's walking.

Let's take a look at a small piece of it's source code:

<pre class="example">

    ##### AUTO-ATTACK #####

    if (($ai_seq[0] eq "" || $ai_seq[0] eq "route" || $ai_seq[0] eq "route_getRoute" || $ai_seq[0] eq "route_getMapRoute" || $ai_seq[0] eq "follow"

        || $ai_seq[0] eq "sitAuto" || $ai_seq[0] eq "take" || $ai_seq[0] eq "items_gather" || $ai_seq[0] eq "items_take")

        ...

</pre>

As you can see here, the auto-attack AI code block is run if any of the above AI sequences are active.

So when Kore is walking (<code>$ai_seq_args[0]</code> is "route"), Kore continues to check for monsters to attack.

<p>

But as you may know, if you manually type "move WhateEverMapName" in the console, Kore will move to that map without attacking monsters (yes, this is intentional behavior). Why is that?

<p>

As seen in example 1, the <code>ai_route()</code> function initializes the route AI sequence.

That function accepts a parameter called "attackOnRoute". <code>$ai_seq_args[0]{attackOnRoute}</code> is set to the same value as this parameter.

Kore will only attack monsters while moving, if that parameter is set to 1.

When you type "move" in the console, that parameter is set to 0. The random walk AI code block however sets that parameter to 1.

<p>

Inside the auto-attack AI code block, Kore checks whether the argument hash that's associated with the "route" AI sequence has a

'attackOnRoute' key, and whether the value is 1.

<pre class="example">

    ...
    $ai_v{'temp'}{'ai_route_index'} = binFind(\@ai_seq,
"route");

```
    <b>if</b> ($ai_v{'temp'}{'ai_route_index'} ne <span
class="cstr">""</span>) {

        $ai_v{'temp'}{'ai_route_attackOnRoute'} =
$ai_seq_args[$ai_v{'temp'}{'ai_route_index'}]{'attackOnRo
ute'};

    }

    ...

    <span class="comment"># Somewhere else in the
autoattack AI code block, Kore checks whether

    # $ai_v{'temp'}{'ai_route_attackOnRoute'} is set to
1.</span>

</pre>
```

`<h2>Timeouts: To wait a while before doing something</h2>`

In certain cases you may want the program to wait a while before doing anything else.

For example, you may want to send a "talk to NPC" packet to the server, then send a "choose NPC menu item 2" packet 2 seconds later.

`<p>`

The first thing you would think of is probably to use the `<code>sleep()</code>` function.

However, that is a bad idea. `<code>sleep()</code>` blocks the entire program. During the sleep, nothing else can be performed.

User command input will not work, other AI sequences are not run, network data is not received, etc.
`<p>`

The right thing to do is to use the <a
href="Utils.html#timeOut"><code>timeOut()</code>
function.

The API documentation entry for that function has two
examples. Here's another example, demonstrating how
you can use the timeOut() function in an AI sequence. This example initializes a
conversation with NPC 1337 (a
Kapra NPC).

Then two seconds later, it sends a "choose NPC menu item
2" packet.

<pre class="example">

The AI() function is run in the
main loop

sub AI {

        ...

        if ($somethingHappened) {

                my %args;

                $args{stage} = 'Just
started';

                unshift @ai_seq, "NpcExample";

                unshift @ai_seq_args, \%args;

                $somethingHappened = 0;

        }

        if ($ai_seq[0] eq "NpcExample") {
                if ($ai_seq_args[0]{stage}
eq 'Just started') {

```
                                    <span class="comment"># This AI
        sequence just started

                                    # Initialize a conversation with
            NPC 1337</span>
sendTalk($net, 1337);


                                    <span class="comment"># Store the
        current time in a variable</span>


        $ai_seq_args[0]{waitTwoSecs}{time} = <b>time</b>;

                                    <span class="comment"># We want to
        wait two seconds</span>


        $ai_seq_args[0]{waitTwoSecs}{timeout} = 2;


                                    $ai_seq_args[0]{stage} = <span
        class="cstr">'Initialized conversation'</span>;


                        } <b>elsif</b> ($ai_seq_args[0]{stage}
        <b>eq</b> <span class="cstr">'Initialized
        conversation'</span>

                                    <span class="comment"># This 'if'
        statement is only true if two seconds have passed

                                    # since
        $ai_seq_args[0]{waitTwoSecs}{time} is set</span>

                                    && timeOut(
        $ai_seq_args[0]{waitTwoSecs} )

                        ) {
                                    <span class="comment"># Two
        seconds have now passed</span>

                                    sendTalkResponse($net, 1337, 2);
                                    <span class="comment"># We're
        done; remove this AI sequence</span>
```

```
                         <b>shift</b> @ai_seq;

                         <b>shift</b> @ai_seq_args;

              }

     }

     ...

}

</pre>
```

<h2>Conclusion & summary</h2>

The entire AI subsystem is kept together by these two variables:

<code>@ai_seq</code> : a queue which contains AI sequence names.

Usually, AI code blocks are run based on the value of the first item in the queue

(though this doesn't have to be true; it depends on how the AI code block is programmed).

<code>@ai_seq_args</code> : contains arguments that's associated with current AI sequence.

The design is pretty simple. This allows the system to be very flexible:
you can do pretty much anything you want. There aren't many real limitations

(but that's just my opinion).

```
<p>

The <code>AI()</code> function runs only very shortly. So
AI code blocks shouldn't do anything that can block the
function for a long time.


<h3>Glossary</h3>

<ul>

<li>An <em class="term">AI code block</em> is an entire
block of code which deals with a certain part of the
AI.</li>

<li>An <em class="term">AI sequence</em> is a value within
the <code>@ai_seq</code> queue (and an associated value
inside the <code>@ai_seq_args</code> array).</li>

</ul>




<p><hr><p>


<div id="footer">

        <ul>

        <li><a
href="http://validator.w3.org/check?uri=referer"
title="Valid HTML 4.01!"><img
src="http://www.w3.org/Icons/valid-html401" alt="Valid
HTML 4.01!" height="31" width="88"></a></li>

        <li><a
href="http://www.mozilla.org/products/firefox/" title="Get
Firefox - Take Back the Web"><img width="104" height="32"
src="http://www.mozilla.org/products/firefox/buttons/getf
irefox_small.png" alt="Get Firefox - Take Back the
Web"></a></li>

        <li><a
```

```
href="http://www.mozilla.org/products/firefox/" title="If
you were looking at this page in any browser but
Microsoft Internet Explorer, it would look and run better
and faster"><img width="45" height="45"
src="http://linuxart.com/img/noIE-small.png" alt="If you
were looking at this page in any browser but Microsoft
Internet Explorer, it would look and run better and
faster"></a></li>

        </ul>

</div>


</div>


</body>
</html>
```