

# BuildTheHTMLPage TeamID:PNT2022TMID07674

```
<!DOCTYPEHTMLPUBLIC"/W3C//DTDHTML4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

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

    <title>Artificialintelligence:OpenKore
    sourcecodedocumentation</title>

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

    <!--FixbrokenPNGtransparencyforIE/Win5-6+-
->

    <!--[ifgteIE5.5000]>

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

    <![endif]-->

    <styletype="text/css">

    <!--

    .example{margin:

        0.3cm;marginleft:0.5cm;

    }

    .comment{font-

        style:italic;

    }

    .term{border-bottom:1px dottedblack;

    }

    .cstr{color:
```

```

        #007700;

    }

    -->

</style>

</head>

<body>

<div="title">OpenKore source coded documentation</div>

<div="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="main">

<h1>How the AI subsystem is designed</h1>
The AI subsystem isn't really complex, but it could take a while to understand its 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 & 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">

```
1          <span class="comment">#####RANDOM
           WALK#####</span>

2          <b>if</b>(<code>$config{'route_randomWalk'}</code>&&
           $ai_seq[0]

           <b>eq</b>""&&@{<code>$field{'field'}</code>>1&& !<code>$cities_lut{$field{'name'}}.rsw</code>){

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

4          <b>do</b>{

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

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

7          }

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

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

### Situation check

Inline 1, it

checks:



- whether the configuration option `route_randomWalk` is on

- whether there are currently no other active **AI sequences** (see below)

- whether we're currently NOT in a city

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

<p>

What is an *Al sequence*? It is a value within the `@ai_seq` array.

This array is a *command queue*.

<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 its task, it will remove that value from the array.

So, if `@ai_seq` 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 `@ai_seq_args` array, used to store temporary variables used by the current AI code block.

If a value is prepended into `@ai_seq`, then a value must also be prepended into `@ai_seq_args`. More on this later.

### Finding a random position to walk to

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

(`$field{field}` 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:

```

<ul>

<li><code>$ai_v{temp}{randX}</code></li>

<li><code>$ai_v{temp}{randY}</code></li>

</ul>

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

```

```

<h3>Moving</h3>

```

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

It tells `ai_route()` where it wants to go to.

`ai_route()` prepends a `"route"` AI sequence in `@ai_seq`, and arguments in a hash

(which is then prepended into `@ai_seq_args` and immediately returns.

Shortly after this, the entire `AI()` function returns.

The point is, `ai_route()` is *not* synchronous.

```

<p>

```

In less than a fraction of a second, the `AI()` function is called again.

Because the `@ai_seq` variable is not empty anymore, the random walk AI code block is never activated (the expression `'$ai_seq[0]eq'''` is false).

```

<p>

```

The AI code block that handles routing is elsewhere in the `AI()` function.

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

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

When the routeAlcode block has finished, it will remove the first item from `@ai_seq`.

If `@ai_seq` is empty, then the random routeAlcode block is activated again.

## Example 2: Attacking monsters while walking to a random spot

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 its source code:

```
<pre class="example">
```

```
    <span class="comment">#####AUTO-ATTACK#####</span>
```

```
    <b>if</b> (($ai_seq[0]<b>eq</b>
<span class="cstr">""</span> | | $ai_seq[0]<b>eq</b>
<span class="cstr">"route"</span> | | $ai_seq[0]<b>eq</b>
<span class="cstr">"route_getRoute"</span> | | $ai_seq[0]
<b>eq</b> <span class="cstr">"route_getMapRoute"</span>
| | $ai_seq[0]<b>eq</b> <span class="cstr">"follow"</spa
n>
        | |          $ai_seq[0]          <b>eq</b>
        <span class="cstr">"sitAuto"</span> | | $ai_seq[0]<b>eq</b>
        <span class="cstr">"take"</span> | | $ai_seq[0]<b>eq</b>
<span class="cstr">"items_gather"</span> | | $ai_seq[0]
<b>eq</b> <span class="cstr">"items_take"</span>) ...
```

```
</pre>
```

As you can see here, the auto-attackAlcode block is run if any of the above sequences are reactive.

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

```
<p>
```

But as you may know, if you manually type "moveWhateverMapName" 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

`ai_route()` function initializes the route sequence.

That function accepts a parameter called "attackOnRoute".

`$ai_seq_args[0]{attackOnRoute}` 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 has that 's' associated with the "route" sequence has a

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

<pre>class="example"> ...

```
$ai_v{'temp'}{'ai_route_index'}=binFind(\@ai_seq,  
<span>class="cstr">"route"</span>); <b>if</b>(<b>$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'}]{attackOnRoute};  
  
}
```

...

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

#<span>class="comment">\$ai\_v{'temp'}{'ai\_route\_attackOnRoute'}isset 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 `sleep()` function.

However, that is a bad idea. `sleep()` 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 [<code>timeOut\(\)</a> function.](Utils.html#timeOut)

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 NPC1337 (a Ka pra NPC).

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

<pre class="example">

<span class="comment"># The AI() function is run in the main loop</span>

<b>sub</b> AI { ...

    <b>if</b> { \$somethingHappened {

        <b>my</b> %args;

        \$args{stage} = <span class="cstr">'Just started'</span>;

        <b>unshift</b> @ai\_seq,

        <span class="cstr">"NPCExample"</span>;

<b>unshift</b>@ai\_seq\_args,\%args;

\$somethingHappened=0;

}

<b>if</b>(\$ai\_seq[0]<b>eq</b>

<spanclass="cstr">"NpcExample"</span>){

<b>if</b>(\$ai\_seq\_args[0]{stage}

<b>eq</b><spanclass="cstr">'Juststarted'</span>){

<spanclass="comment">#ThisAI

sequencejuststarted

#Initializeaconversationwith

NPC1337</span>

sendTalk(\$net,1337);

<spanclass="comment">#Store

thecurrenttimeinavariablenbsp>

\$ai\_seq\_args[0]{waitTwoSecs}{time}=<b>time</b>;

<spanclass="comment">#We

wanttowaittwoseconds</span>

\$ai\_seq\_args[0]{waitTwoSecs}{timeout}=2;

\$ai\_seq\_args[0]{stage}=

<spanclass="cstr">'Initializedconversation'</span>;

<b>elsif</b>(\$ai\_seq\_args[0]{stage}

<b>eq</b><span

class="cstr">'Initializedconversation'</span>

<spanclass="comment">#This

'if'statementisonlytrueiftwosecondshavepassed

```

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

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

                                ){

                                <spanclass="comment">#
Twosecondshavenowpassed</span>

                                sendTalkResponse($net,1337,2);

                                <spanclass="comment">#
We'redone;removethisAlsequence</span> <b>shift</b>@ai_seq;

                                <b>shift</b>@ai_seq_args;

                                }}

                                ...

                                }

</pre>

```

## Conclusion&summary</h2>

TheentireAlsubsystemiskepttogetherbythese twovariables:



- <code>@ai\_seq</code>:aqueuewhichcontains Alsequencenames.

Usually,Alcodeblocksarerunbasedonthevalueofthefirst iteminthequeue

(thoughthisdoesn'thavetobetrue;itdependsonhowtheAl  
codeblockisprogrammed).</li>

- <code>@ai\_seq\_args</code>:containsarguments  
that'sassociatedwithcurrentAlsequence.</li>



Thedesignisprettysimple.Thisallowsthesystemto beveryflexible:

you can do pretty much anything you want. There aren't many real limitations (but that's just my opinion).

<p>

The `AI()` 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 *AI code block* is an entire block of code which deals with a certain part of the AI.</li>

<li>An *AI sequence* is a value within the `@ai_seq` queue (and an associated value inside the `@ai_seq_args` array).</li>

</ul>

<p><hr><p>

<div id="footer">

<ul>

<li><a href="http://validator.w3.org/check?uri=referer" title="ValidHTML 4.01!"></a></li>

<li><a href="http://www.mozilla.org/products/firefox/" title="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"></a></li>

Explorer,itwouldlookandrunebetterandfaster"></a></li>

</ul>

</div>

</div>

</body>

</html>