DATE	05 November 2022
TERM ID	PNT2022TMID28501
PROJECT NAME	Real-Time Communication System Powered by AI for Specially Able

OpenKore source code

documentation Main website

Table of contents

Artificial intelligence

How the AI subsystem is designed

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

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

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

Example 1: Random walk

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 AI() function makes Kore walk to a random spot if it isn't doing anything:

```
##### RANDOM WALK #####
if ($config{'route_randomWalk'} && $ai_seq[0] eq "" && @{$field{'field'}} > 1
&&!$cities_lut{$field{'name'}.'.rsw'}) {
# Find a random block on the map that we can walk on
do {
$ai_v{'temp'}{'randX'} = int(rand() * ($field{'width'} - 1));
$ai_v{'temp'}{'randY'} = int(rand() * ($field{'height'} - 1));
} while ($field{'field'}[$ai_v{'temp'}{'randY'}*$field{'width'} + $ai_v{'temp'}{'randX'}]);

# Move to that block
message "Calculating random route to:
$maps_lut{$field{'name'}.'.rsw'}($field{'name'}):
$ai_v{'temp'}{'randX'}, $ai_v{'temp'}{'randY'}\n", "route";
ai_route(\%{$ai_v{'temp'}{'returnHash'}},
$ai_v{'temp'}{'randX'},
$ai_v{'temp'}{'randY'},
$field{'name'},
0,
$config{'route_randomWalk_maxRouteTime'},
2,
undef,
undef,
1);
}
```

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

In line 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.

What is an AI sequence? It is a value within the @ai_seq array. This array is a command queue.

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 @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.

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.)

The result coordinate is put into these two variables:

$ai_v{temp}{randX}

$ai_v{temp}{randY}

(In case you didn't know, $foo{bar} is the same as $foo{'bar'}.) Moving

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

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).

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 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 @ai_seq. If @ai_seq is empty, then the random route AI code 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 it's source code:

```
##### 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")

...
```

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

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?

As seen in example 1, the ai_route() function initializes the route AI 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.

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.

```
...

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

if ($ai_v{'temp'}{'ai_route_index'} ne "") {


$ai_v{'temp'}{'ai_route_attackOnRoute'} =

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

}



...


# Somewhere else in the auto- attack AI code block, Kore checks

whether # $ai_v{'temp'}{'ai_route_attackOnRoute'} is set to 1.
```

Timeouts: To wait a while before doing something

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.

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.

The right thing to do is to use the timeOut() 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.

```
# 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') {

# This AI

sequence just started
```

```
 # Initialize a conversation
 with NPC 1337
 sendTalk($net, 1337);


 # Store the current time in a variable


 $ai_seq_args[0]{waitTwoSecs}{time} =
 time; # We want to wait two seconds
$ai_seq_args[0]{waitTwoSecs}{timeout} = 2;


 $ai_seq_args[0]{stage} = 'Initialized conversation';


 } elsif
 ($ai_seq_args[0]{stage} eq
 'Initialized conversation'
 # This 'if' statement is only true if two seconds have
 passed # since


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

)


{
```

```
# Two seconds have now passed
sendTalkResponse($net, 1337, 2);


# We're

done; remove this AI
sequence shift @ai_seq;
shift @ai_seq_args;

}


}

...


}
```

Conclusion & summary

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

@ai_seq : 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).

@ai_seq_args : 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).

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

Glossary

An AI code block is an entire block of code which deals with a certain part of the AI.

An AI sequence is a value within the @ai_seq queue (and an associated value inside the @ai_seq_args array).