

## Making AI more intelligent

As a young tearaway, I was always in trouble. My main problem was the instruction "Whatever you do, don't go *<insert dangerous place here>*". Even though I'm the owner of at least a dozen X-rays of various limbs and broken bones, I still think it was healthy to be inquisitive at that age. My inquisitive nature got the better of me when I first started playing with the Dark AI plugin. And to my delight, I discovered I could change the intelligence of the system to my advantage. Just like my own parents, I found that it's possible to say "Don't go there!" whilst still allowing my characters to venture into dangerous territory when it's absolutely necessary, or just too exciting to resist.

In this tutorial, we'll use a realistic example. We'll set up a scene with a small number of obstacles, and a hill between 2 camps. The challenge will be to persuade the character to take the easy route and walk around the hill, but at the same time allow him to climb the hill in times of peril or attack.



### Step 1 – Making the terrain

Firstly, we need a setting. This example uses Advanced Terrain to create the ground, which is part of the standard DarkBASIC Professional command set. It generates a large hill in the centre, and also encloses the playing area with a raised area, a useful technique for removing unsightly edges and implying that the land is larger than it actually is. I have used TreeMagik and Plant Life to add a small number of obstacles, including the rocks that define the 2 camps. This is carried out in function *makeTerrain()*. Take this opportunity to download, install and run Step 1. You can use the arrow keys to fly over the terrain and picture the task ahead.

At this stage, the first steps of the AI have also been introduced.

- The rocks have been added directly to the AI system as obstacles.
- The bases of the trees have been added by placing a cube in the same position and using this as the obstacle template. We cannot use the tree directly, because Dark AI would assume that the overhead foliage was part of the obstructed area.
- Several other waypoints have also been manually added at the top of the hill and around the base (function *createWaypoints()*). This is to ensure that the character we introduce later has various options available to him.

## Step 2 – Adding the Character

Now it's time to see how a character reacts to our newly created environment. This part is ridiculously simple. After loading the character and setting a few basic parameters, he is added to the AI system (function *makeCharacter()*). A path is created between the start and destination, and the character is instructed to use this path as his Patrol. You can compile and run Step 2 now, and watch what happens after implementing just a few lines of code.

We have some success in that our soldier is walking around under the control of the AI system. However, things are a little strange to say the least. Firstly, the soldier has no interest in the intermediate waypoints, you will notice that they are ignored. Secondly, he has taken a liking to the trees in the scene, and takes quite a detour on his way to the destination. So what is happening?

The soldiers preferred path between 2 points is a straight line. However, the rocks in the scene prevent this. After looking around for an alternative path, the soldier determines that the best waypoints to use are those that define the tree border. Despite the presence of a large hill and intermediate waypoints, the AI system allows the soldier to "see" the tree and use this as the best option.

The soldier is currently achieving his aim and reaching the second camp, but the situation is not ideal. We need to reduce his range of sight, give him more options and finally persuade him to walk around the hill. You will have noticed that the code causes the soldier to slow down when ascending hills. In reality, it would be natural for him to walk around the hill if it was the faster, easier option.

### Step 3 – More waypoints, more choices

In the next step, we will add more waypoints, reduce the “look-ahead” ability, and the result should be closer to our desired effects.

Firstly, adding more waypoints is a simple way to add more options for our characters. Waypoints are created automatically by obstacles, but in scenarios like ours they are few and far between. In extreme cases, the entities can actually suffer from agoraphobia and refuse to move from their position. Taking performance into consideration, several waypoints are added in this version of the program.

Waypoints have been added roughly every 100 units, with a random factor to prevent clinical straight-line paths. After adding them, updating the visibility is essential, and this is also where performance is carefully controlled. Here is the command as it is used in our tutorial:

```
AI update waypoint visibility 0, 120
```

The final parameter dictates the maximum distance of the paths generated between points. This has been set so that adjacent waypoints will form paths, but those further afield are out of range. Without this parameter, every point would be joined to all others by a rats’ nest of paths. This would add a huge burden to the AI system, and be an unnecessary complication in our scenario.

At this stage, we have also added a similar parameter to the path-creating command:

```
AI make path between points 10, 0, x1#,z1#,x2#,z2#,300
```

The final parameter in this command ensures that a path can only be constructed from waypoints within 300 units of the previous one. This will stop our friend from seeing the tree in the distance and using it as part of his path. He must include more waypoints in the path to find his way from one end to the other.

Compile and run the code again, and check the results.

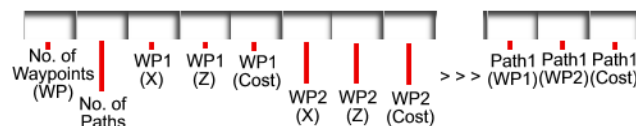
## Step 4 – The finishing touches

The results of the previous stage speak for themselves. Our soldier can successfully find his way from A to B without taking obscure detours. The final step is to ensure he doesn't climb that rather large, steep hill in the centre which slows him down so much.

The obvious solution is to make the hill an obstacle. Then he is forced to find an alternative route because it is impassable. But then it's also out of bounds permanently. You can't flee up the hill, attack up the hill, or mount an unexpected assault on your enemy from above. You have effectively built a wall around a large section of your landscape.

Dark AI has a fantastic feature that we can take advantage of to accomplish this task. The waypoints and paths can be copied into a memblock for us to manipulate. First, it's important to understand how pathfinding works. In it's simplest form, the AI module looks at all of the possible paths and works out which is the shortest. It does this by looking at the 'cost' of each edge between waypoints, which is actually it's distance. The waypoints themselves also have a cost, although this is zero by default.

In the tutorial source, the waypoint and path data has been copied into a memblock (*updateWaypoints()*). The waypoints occupy 12 bytes each, from position 8 in the memblock. The float value at Position 0 tells us how many waypoints there are. Incidentally, position 4 indicates the number of paths, and they also occupy 12 bytes each, continuing in the memblock at the point where the waypoints end.



The waypoint data is in 3 parts; the X position, the Z position, and the cost. In our example, any waypoint in a position on the map that is higher than 60 units has been modified. The cost of travelling through the waypoint has been updated to 2000, a value way above the potential cost of walking around the hill.

This is starting to sound complicated, but I assure you it isn't. Let's look at the pseudo-code, which explains the source code that you can also view in the downloaded files:

- Load waypoints into memblock
- Get waypoint count (Position 0)
- For each waypoint in memblock
  - Get X and Z positions
  - Get ground height from map
  - If ground > 60, add 2000 to waypoint cost
- Make waypoints from memblock

Compile and run the last example; it's time to see if the not-so-hard work has paid off.

### **The final word, and an important little trick**

Our soldier now makes the intelligent decision to walk around the hill. But at the same time, the hill is still available to hide in, attack from and run away from the enemy. In fact, because the return path of a patrol is directly to the start point, our soldier will return back over the hill to prove that it is still accessible.

One thing to notice is that after updating the cost of the waypoints, we did not need to update the waypoint visibility. Why not? Well, it's because all of the waypoint edges are still in tact, it's only the cost of passing through them that has changed. And why is this an important little trick? It means that you can interactively change the way your characters move around the map without the overhead of rebuilding the visibility, which is a costly exercise. For example, you could prevent an entity from returning down a corridor by changing the cost of the waypoint directly behind him. You could burn down a building, and make the paths leading to it too costly to consider. My favourite use of this technique is to randomly increase the cost of waypoints in an arena to make the AI system less predictable, and ensure every game is different.

Until next time,  
Happy coding!

Steve Vink.