DARK A.I.

The DarkAI expansion pack provides a set of commands to create and control computer controlled characters in Dark Basic Professional. The system is almost completely automatic allowing you to drop in entities and obstacles and then letting them move about on their own. DarkAI includes several features that help you organise your world and give you a wide range of control over the AI system, including:

Path Finding
Flexible waypoint based path finding system that calculates a set of waypoints around all obstacles added to the AI system. Entities then use an A* algorithm to calculate a path between their position and destination. This means your entities should always use the shortest path between two points, no matter how complex the situation. You can also create a path between two specific points allowing you to use the path finding feature on its own.

Teams
Provides 3 teams (enemy, friendly and neutral teams) to organise entities in the AI world. Neutral entities run away from combat, friendlies attack the player's enemies whilst enemies attack the player and its allies.

Entity Commands
By default entities will automatically move about the world and react to events (e.g. Sounds) and other entities according to their team and aggressiveness. Set the entity aggressiveness and restrictions by using commands such as 'Defend Point' or 'Patrol Path', the entity will then act in accordance with these restrictions.

Manual Commands
Entities can also be moved around the world using a set of more specific commands such as setting the entity destination or look at point directly

with 'Set Destination' and 'Look At' or 'Look At Target', the entity will create a path automatically to its destination and inform you when it's ready to fire. Entities will detect nearby enemies but not act against them during manual control.

Zones
Add one or more zones to an entity that will trigger a response when an opposing team member enters it. Entities can be set to ignore people if they leave a zone or chase them until they can no longer see them.

Direct Integration
If using DarkBasic Professional 3D objects you can set the AI system to automatically move and turn your objects for you.

Containers
Areas of your world can be separated from each other using containers, such as different floors in a building. The player can freely move between containers whereas Entities can be restricted to the bounds of its container.

You can check out examples of all these features in the provided demos, along with detailed descriptions of each in the demo section of the help file. The help file also includes the full command list explaining everything you need to know about how the commands work and how to use them.

AI ADD CONTAINER

Creates a new container to the AI system to separate objects within it from other containers. A container must be added before you can add any objects to it. Container 0 is created at system start-up and is the default location for all AI data.

Syntax
AI Add Container *Container Number*

Parameters
Container Number, the id number you want to use to represent this container, must be a non-zero positive integer

Return
n/a

OBSTACLES

Obstacles represent the physical limitations on where entities can move and see, such as walls and scenery. Entities will automatically work their way around defined obstacles to reach their intended destination and use them as cover when appropriate. There are four main types of obstacle that can be added to the AI system:

- *Full Height* obstacles define an area that cannot been seen over or passed through at any time and entities must always move around it to see the other side.

- *Half Height* obstacles define an area that cannot be passed through but can be seen over whilst the entity is standing. Ducking entities cannot see over half height obstacles, and also cannot be seen by others.

- *Boundary* obstacles enclose an area beyond which an entity cannot see or move, such as the bounds of a level. Therefore entities can be restricted to the area within the boundary obstacle and cannot move out of it. Only one boundary per container should be added to prevent problems with multiple confining areas.

- *View Blocking* obstacles are a special type of obstacle that does not block movement, entities are free to move around as if the obstacle didn't exist, but entities cannot see through it. As such these obstacles do not affect the waypoint network and can be added and removed efficiently whilst the system is in motion, allowing you to use it to represent a closed door then remove it when the door is opened.

Obstacles can be added to separate containers to divide the world into enclosed sections. For example each floor of a building can be represented by a container and the obstacles of each floor are added to the relevant container, by default a single container (0) is created to hold all obstacles and entities.

When you have added all your obstacles you must call *AI Complete Obstacles* to complete the setup and create waypoint and collision data for all the obstacles. This does not include view blocking obstacles which can be added and removed at any time without needing this command.

AI ADD STATIC OBSTACLE

This command adds an obstacle to the AI system from a specified DarkBasic object. The AI system will create a convex shape from the object by looking top down on the object and joining the outside points in a circular fashion. Therefore this is best suited to objects that are already convex in nature such as cubes and spheres, and other more complex shapes should be made using *AI Start New Obstacle*. This command does not require the DarkBasic object to exist after it has been called and does not update the internal state if the object is moved or rotated.

Obstacles can be created as full height or half height, the only difference being that half height objects do not block an entity's view whilst it is standing, but do when it is ducking. Both block an entity's path from one point to another. The default is full height (1).

The container number allows you to add the obstacle to a different container, the default is 0.

For dynamic obstacles (that obstruct movement) it is recommended that you use an external collision or physics system and report such collisions to the entity using *AI Set Entity Collide*.

See *AI Add View Blocking Obstacle* for adding obstacles that do not obstruct movement.

You must call *AI Complete Obstacles* sometime after this command, after all static obstacles have been added, to complete the setup and create waypoint and collision data for all obstacles. If you add an obstacle after calling *AI Complete Obstacles* you must call it again to see any changes you make.

Syntax
AI Add Static Obstacle *Obstacle Number*
AI Add Static Obstacle *Obstacle Number, Height*
AI Add Static Obstacle *Obstacle Number, Height, Container Number*

Parameters
Obstacle Number, The id of the object you want to add
Height, (optional) 1 for full height, 0 for half height
Container Number, The id of the container you want to add the obstacle to.

Return
n/a

AI ADD VIEW BLOCKING OBSTACLE

This command adds an view blocking obstacle to the AI system from a specified DarkBasic object. The AI system will create a convex shape from the object by looking top down on the object and joining the outside points in a circular fashion. Therefore this is best suited to objects that are already convex in nature such as cubes and spheres, and other more complex shapes should be made using *AI Start New Obstacle*. This command does not require the DarkBasic object to exist after it has been called and does not update the internal state if the object is moved or rotated.

View blocking obstacles are a special type of obstacle that do not obstruct the movement of entities but still prevent entities from seeing through the edges of the obstacle. Unlike normal obstacles view blocking obstacles do not affect the waypoint network when added or removed, as such view blocking obstacle can be added or removed with very little impact on performance and can be done in real-time. This could be used to represent a door which prevents entities from seeing through it, but can be removed when the door is opened, and re-added when it is closed.

Obstacles can be created as full height or half height, the only difference being that half height objects do not block an entity's view whilst it is standing, but do when it is ducking. The default is full height (1). The container number allows you to add the obstacle to a different container, the default is 0.

For dynamic obstacles (that obstruct movement) it is recommended that you use an external collision or physics system and report such collisions to the entity using *AI Set Entity Collide*.

See *AI Add Static Obstacle* for adding obstacles that obstruct movement.

Syntax
AI Add View Blocking Obstacle *Obstacle Number*
AI Add View Blocking Obstacle *Obstacle Number, Height*

AI Add View Blocking Obstacle *Obstacle Number, Height, Container Number*

Parameters
Obstacle Number, The id of the object you want to add
Height, (optional) 1 for full height, 0 for half height
Container Number, The id of the container you want to add the obstacle to.

Return
n/a

AI REMOVE OBSTACLE

This command removes all obstacles from all containers that correspond to the specified obstacle number from the AI system. After this command you must call *AI Complete Obstacles* to update the waypoint can collision data which is a slow command and should not be done in your main loop, the exception being view blocking obstacles. This does not remove the DarkBasic object that represents the obstacle, you should remove that yourself.

This can be used to remove both normal and view blocking obstacles, when the obstacle number refers only to view blocking obstacles, and no normal obstacles are removed, this command does not need *AI Complete Obstacles* to be called and so can be called in your main loop to remove things like doors when opened.

Syntax
AI Remove Obstacle *Obstacle Number*

Parameters
Obstacle Number, The id of the obstacle you want to remove

Return
n/a

AI START NEW OBSTACLE

This command starts the creation of a new obstacle that is created by manually adding points that define the obstacle's outside edge. It must be used with the commands *AI Add obstacle Vertex* and *AI End New Obstacle* to finish defining the obstacle. You must end or discard a new obstacle before you can begin another one.

There is no restriction on the obstacle number, two obstacles can have the same id but would both be removed using *AI Remove* Obstacle. The default id is 0.

Syntax
AI Start New Obstacle
AI Start New Obstacle *Obstacle Number*

Parameters
Obstacle Number, (optional) The id you want to use to represent this obstacle.

Return
n/a

AI ADD OBSTACLE VERTEX

This command adds a point to the obstacle started with *AI Start New Obstacle*. These points will form the outside edge of the obstacle in the order that they are added, the last point will be joined with the first point to create an enclosed space.

Points should be defined in a clockwise order, from a top down view, to create an obstacle that entities will avoid, or anti-clockwise to create a boundary that will confine entities to an area. There is no limit to how many clockwise obstacles can be added per container but only one or less boundaries should be added per container.

Syntax
AI Add Obstacle Vertex *Position X#, Position Z#*

Parameters
Position X#, The X component of the position in world co-ordinates of the vertex you want to add
Position Z#, The Z component of the position in world co-ordinates of the vertex you want to add

Return
n/a

AI END NEW OBSTACLE

This command completes the new obstacle and adds it to the specified container. If you are not using different containers use the default value of 0. The height defines if it is a full height or half height obstacle, the difference being that half height objects do not obstruct an entity's view whilst standing, but do whilst ducking. Both block an entity's path when moving to a destination.

The obstacle type parameter allows you to choose between an obstacle that blocks both movement and view (normal) or an obstacle that only blocks view, meaning an entity is free to move around as if the obstacle didn't exist but cannot see through any edges of the obstacle. A bit like hanging cloth or a closed door (which can be opened). Unlike normal obstacles view blocking obstacles do not affect the waypoint network when added or removed, as such view blocking obstacle can be added or removed with very little impact on performance and can be done in the main loop.

You must call *AI Complete Obstacles* after this command if you use an obstacle type of 0 to see the changes in the AI system. This does not apply if you are creating view blocking obstacles (type 1).

Syntax
AI End New Obstacle *Container Number, Half Height, [Obstacle Type]*

Parameters
Container Number, The id of the container you want to add the obstacle to, must exist.
Half Height, 0 for a half height object 1 for a full height object.
Obstacle Type, (optional) 0 for a normal obstacle, 1 for a view blocking obstacle. Default is 0.

Return
n/a

AI DISCARD NEW OBSTACLE

This command deletes an obstacle that was started using *AI Start New Obstacle,* discards all information about it, and allows you to start a new one if you choose.

Syntax
AI Discard New Obstacle

Parameters
n/a

Return
n/a

AI ADD OBSTACLE FROM LEVEL

This command creates a set of obstacles from a mesh, for example an entire level, by using the polygons that make it up to work out where the walls are. This can be used to quickly and easily add all the collision data you need for large levels and then add the smaller obstacles afterwards. You can also make this command output a list of obstacles to a file so that you can change it and add it manually to your own code if one or more parts are not quite right, this will be put in the same folder as the .exe in a file called 'Obstacle Data.txt'. Any file that already exists with this name will be over written.

It does this by taking a flat plane, at the height you specify, and calculating all polygons that intersect it. This provides a rough idea of where the polygons that represent walls are located, it ignores any polygons that are orientated at less than 45 degrees or that are smaller in Y size than *Min Height*. The polygons create lines of intersection across the plane, any lines that are less that *Min Length* are ignored. Lines that are close to each other (within the radius defined with AI Set Radius) are merged and then used directly as obstacles with two vertices. These create double sided obstacles that will allow movement on both sides but not through it, as opposed to a boundary which will not allow movement outside of it. You can use *AI Debug Show Obstacle Bounds* to check the obstacles produced.

The list of obstacles are added to container 0 as full height obstacles with obstacle id 0. The code to create the list of obstacles produced can be output to a file by setting *Output To File* to 1, this will not add the obstacles to the AI system. You will still need to place obstacles to represent windows which may leave a hole in a wall at the height you specify. You can do this at any time before or after this command. You can change the container they are added to and the height (full height or half height) using the *Container Number* and *Height* parameters.

AI Complete Obstacles must be called some time after this command, after all static obstacles have been added, to complete the setup and produce waypoint data for the obstacles.

Syntax
AI Add Obstacle From Level *Object Number, Plane Height#, Min Length#, Min Height#*
AI Add Obstacle From Level *Object Number, Container Number, Plane Height#, Min Length#, Min Height#*
AI Add Obstacle From Level *Object Number, Container Number, Height, Plane Height#, Min Length#, Min Height#*
AI Add Obstacle From Level *Object Number, Container Number, Height, Plane Height#, Min Length#, Min Height#, Output To File*

Parameters
Obstacle Number, The id of the object you want to create obstacles from.
Container Number, (optional) The id of the container you want to add the obstacles to.
Height, (optional) The height you want the obstacles to be, full-height(1) or half-height(0).
Plane Height#, The height in world co-ordinates that you want obstacles from.
Min Length#, The minimum intersection length that can create an obstacle.
Min Height#, The minimum height of a polygon for it to contribute to obstacles.
Output To File, (optional) Set to 1 to output the code for creating the obstacles, 0 to add straight to the AI system. Default is 0.

Return
n/a

AI COMPLETE OBSTACLES

This command must be called after you have added all your static objects to the scene. It creates the collision and waypoint data for all obstacles at once to make sure expensive operations are performed only once, as opposed to every time you add an obstacle. If you make a change by adding or removing a static obstacle after this command you must call it again to see the effects in the scene. This does not apply to View Blocking obstacles which are not involved in movement data and can be added/removed in real-time with no special commands.

You can specify a particular container to update obstacle data for, you only need to update obstacles in the container that you added or removed them, other containers are not affected. The default is -1 which updates all containers.

You only need to do this once and since it is an expensive command should not be done in your main loop.

Syntax
AI Complete Obstacles
AI Complete Obstacles *Container Number*

Parameters
Container Number, (optional) The id of the container you want to update the obstacles data for.

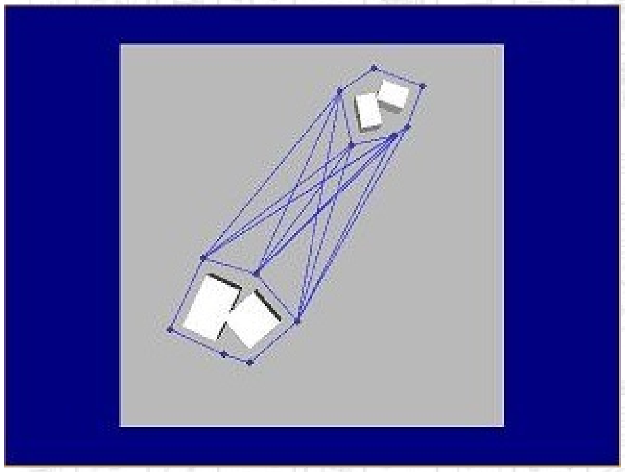
Return
n/a

WAYPOINTS

Waypoints define points which entities can use to avoid obstacles in the AI system. By default they are positioned at corners and linked together to build a network that can move an entity to any valid position within the system.

Waypoints are created automatically when adding obstacles but you can choose to create your own waypoints by adding them manually and updating their connections using *AI Update Waypoint Visibility*. Adding an obstacle to a container after changing the waypoints will reset the waypoint network back to its default construction.

This shows an example of the waypoints (dots) and their connections (lines) which can be used to make up an entity path from one position to another.



AI ADD WAYPOINT

This command adds a waypoint to a container at the specified position. The waypoint will not be used until it is linked into the waypoint network using *AI Update Waypoint Visibility* on the container, this allows you to add many waypoints before performing the expensive update operation. Waypoints are linked by their visibility to other waypoints, so if the waypoint is inside an obstacle or completely surrounded by them it will not be used in path finding.

No obstacle need to exist to add waypoints, waypoints can be added to any position in the container.

Waypoints are added to the beginning of the waypoint list.

Syntax
AI Add Waypoint *Container Number, Position X#, Position Z#*

Parameters
Container Number, The id of the container you want to add the waypoint to, must exist.
Position X#, The X component of the waypoint position, in world co-ordinates.
Position Z#, The Z component of the waypoint position, in world co-ordinates.

Return
n/a

AI REMOVE WAYPOINT

This command removes the specified waypoint from the container. The index should be in the range 1 up to and including the value returned by *AI Count Waypoints.* Waypoints are added to the beginning of the list so the most recently added waypoint will be in position 1. The index also relates to the order of waypoints when a memblock is created from them with the first waypoint being index 1.

Syntax
AI Remove Waypoint *Container Number, Waypoint Number*

Parameters
Container Number, The id of the container you want to remove the waypoint from.
Waypoint Number, The id of the waypoint in the container that you want to remove.

Return
n/a

AI COUNT WAYPOINTS

This command will return the number of waypoints in the specified container.

Syntax
return integer = AI Count Waypoints (*Container Number*)

Parameters
Container Number, The id of the container from which you want to retrieve the number of waypoints.

Return
The number of waypoints in the specified container.

AI CLEAR WAYPOINTS

This command clears all waypoints from the container effectively removing all path finding data for it. Entities will be restricted to straight line paths in containers with no waypoint data, and any obstacles in the way will result in no path.

Syntax
AI Clear Waypoints *Container Number*

Parameters
Container Number, The id of the container you want to clear of waypoints.

Return
n/a

AI MAKE MEMBLOCK FROM WAYPOINTS

This command will create a memblock containing the waypoints from the specified container. If the memblock already exists it will first be deleted.

The structure of the memblock is as follows:
Number of waypoints as DWORD (4 bytes)
Number of edges as DWORD (4 bytes)

Then for each waypoint:
X# as FLOAT (4 bytes)
Z# as FLOAT (4 bytes)
Cost# as FLOAT (4 bytes)

Then for each edge:
First Waypoint Index as DWORD (4 bytes)
Second Waypoint Index as DWORD (4 bytes)
Edge Cost# as FLOAT (4 bytes)

The cost for a waypoint defines the cost of moving through that waypoint when using it as a path, entities will attempt to avoid using higher cost waypoints if they increase the path cost too much and an alternative route is available. The default cost for all waypoints is 0, and must be a positive value.

Edges reference waypoints from the waypoint list above it, with index 0 being the first waypoint in the list. The edge is valid from the first waypoint to the second, but not the other way around. Therefore, if you want to define that the edge between two waypoints can be used in both directions you must define two edges, one going from the first waypoint to the second waypoint and the second edge going from the second waypoint to the first waypoint. The edge cost must always be at least the straight line distance between the two waypoints for the path finding to work correctly, but can be any number above this. Entities will avoid edges with a large cost where possible preferring the least cost path.

Syntax

AI Make Memblock From Waypoints *Memblock Number, Container Number*

Parameters

Memblock Number, The id of the memblock you want to use to store the waypoint data.
Container Number, The id of the container you want to use to fill the memblock.

Return

n/a

AI MAKE WAYPOINTS FROM MEMBLOCK

This command replaces the container's existing waypoints with those from the memblock. You can choose to specify no edges and instead call the *AI Update Waypoint Visibility* command afterwards to update the links between the waypoints for use in path finding based on line of sight.

The structure of the memblock should be as follows:
Number of waypoints as DWORD (4 bytes)
Number of edges as DWORD (4 bytes)

Then for each waypoint:
X# as FLOAT (4 bytes)
Z# as FLOAT (4 bytes)
Cost# as FLOAT (4 bytes)

Then for each edge:
First Waypoint Index as DWORD (4 bytes)
Second Waypoint Index as DWORD (4 bytes)
Edge Cost# as FLOAT (4 bytes)

The cost for a waypoint defines the cost of moving through that waypoint when using it as a path, entities will attempt to avoid using higher cost waypoints if they increase the path cost too much and an alternative route is available. The default cost for all waypoints is 0, and must be a positive value.

Edges reference waypoints from the waypoint list above it, with index 0 being the first waypoint in the list. The edge is valid from the first waypoint to the second, but not the other way around. Therefore, if you want to define that the edge between two waypoints can be used in both directions you must define two edges, one going from the first waypoint to the second waypoint and the second edge going from the second waypoint to the first waypoint. The edge cost must always be at least the straight line distance between the two waypoints for the path finding to work correctly, but can be any number above this. Entities will avoid edges with a large cost where possible preferring the least cost path.

Syntax
AI Make Waypoints From Memblock *Container Number, Memblock Number*

Parameters
Container Number, The id of the container you want to fill with waypoints, must exist.
Memblock Number, The id of the memblock you want to use to use as source data.

Return
n/a

AI UPDATE WAYPOINT VISIBILITY

This command updates the links between waypoints that define valid routes between unobstructed waypoints. This is automatically done when adding obstacles but should be done manually when adding waypoints manually or using memblocks to replace waypoints. You can optionally specify a maximum distance that will restrict the distance at which waypoints can be linked, this can be useful if you want to create lots of closely spaced waypoints which can create many unnecessary edges in large open spaces.

Syntax
AI Update Waypoint Visibility *Container Number*
AI Update Waypoint Visibility *Container Number, Range Limit#*

Parameters
Container Number, The id of the container you want to remove the waypoint from.
Range Limit#, (optional) The maximum distance between two visible waypoints that will form an edge.

Return
n/a

PATHS

A path holds a list of points which can be used to describe a route in the order the points are defined. The main purpose for this is to assign patrol routes to entities, but they can also be used as a simple structure for holding a list of unrelated points.

You can create create a path between two points that will take into account any obstacles in the way, allowing you direct access to the path finding system. Such paths will follow waypoints and valid edges between them to build up a complete path. If the start or end points are inside obstacles attempts will be made to find the closest point outside the obstacle (this may fail where multiple overlapping obstacles create a difficult solution).

AI MAKE PATH

This command will create an empty path with the specified id to which you can then add points to. A path contains a list of points which among other things can be passed to an entity to form a patrol route.

Syntax
AI Make Path *Path Number*

Parameters
Path Number, the id you want to use to represent this path.

Return
n/a

AI DELETE PATH

This command deletes the path corresponding to the specified id. Any entity that was using this path as a patrol route will stop patrolling.

Syntax
AI Delete Path *Path Number*

Parameters
Path Number, The id of the path you want to delete.

Return
n/a

AI PATH ADD POINT

This command adds the specified point to the end of the path. Points need not be related in any way, a path simply holds a list of points that can be used later for various purposes. When adding points for a patrol route the points should be added in the order you want the entity to visit them.

Syntax
AI Path Add Point *Path Number, X#, Y#*

Parameters
Path Number, The id of the path you want to add the point to.
X#, The X component of the point position in world co-ordinates.
Y#, The Y component of the point position in world co-ordinates.

Return
n/a

AI PATH COUNT POINTS

This command returns the number of points in the path.

Syntax
return integer = AI Path Count Points (*Path Number*)

Parameters
Path Number, The id of the path from which you want to return the number of points.

Return
The number of points contained in the path.

AI PATH GET X

This command returns the X component of the specified point in the path. The points are stored in the order they were added and begin at index 1 up to and including the value returned by *AI Path Count Points.*

Syntax
return float = AI Path Get X (*Path Number, Point Index*)

Parameters
Path Number, The id of the path from which you want to return the X position.
Point Index, The id of the point (starting at 1) of which you want to return the X position.

Return
The X position of the path point.

AI PATH GET Z

This command returns the Z component of the specified point in the path. The points are stored in the order they were added and begin at index 1 up to and including the value returned by *AI Path Count Points.*

Syntax
return float = AI Path Get Z (*Path Number, Point Index*)

Parameters
Path Number, The id of the path from which you want to return the Z position.
Point Index, The id of the point (starting at 1) of which you want to return the Z position.

Return
The Z position of the path point.

AI MAKE PATH BETWEEN POINTS

This command will create a path between the two points specified and will take into account all obstacles in its way. This gives you direct access to the path finding system of a particular container (default is 0). If the begin and end points lie within obstacles an attempt will be made to find the closest point that is not within an obstacle. If this fails, or there is no clear path between the points then the path will be created with no points.

The *Max Start Cost* parameter sets the maximum distance the path can move from the start point without moving through a waypoint. Normally if the start and end points are not obstructed a path is created directly between them, if the max start cost is less than the distance between the start and end points the path must connect to a waypoint (with the max start cost range) in order to reach the destination. The end point must also be within this range of a waypoint for the path to connect to it. If either the start or end point are more than the max start cost from all waypoints then no path is created, even if the points are unobstructed. Set this to -1 to set no limit on the start and end distance (default).

If successful the begin and end points are included as part of the path.

Syntax
AI Make Path Between Points *Path Number, X1#, Y1#, X2#, Y2#*
AI Make Path Between Points *Path Number, Container Number, X1#, Y1#, X2#, Y2#*
AI Make Path Between Points *Path Number, Container Number, X1#, Y1#, X2#, Y2#, Max Start Cost#*

Parameters
Path Number, The id of the path you want to use to hold the returned data, should not exist.
Container Number, (optional) The id of container you want to create the path through, default is 0.
X1#, The X component of the start position.
Y*1#,* The Y component of the start position.
X2#, The X component of the end position.

Y2#, The Y component of the end position.
Max Start Cost#, The maximum distance between the start/end points and their connecting waypoints. (default is -1)

Return
n/a

AI MAKE PATH FROM CLOSEST WAYPOINTS

This command creates a path that contains a list of points that represent the visible waypoints from the specified point. As such this does not represent a real path but is instead used to store a list of related points. The waypoints represent the places an entity could move to from its the specified position which are not obstructed by obstacles. They can also denote good places to run for cover since a waypoint represents the corner of an obstacle.

Syntax
AI Make Path From Closest Waypoints *Path Number, Container Number, X#, Y#*

Parameters
Path Number, The id of the path you want to use to hold the returned data.
Container Number, The id of container you want to create the path from.
X#, The X component of the position.
Y#, The Y component of the position.

Return
n/a

AI MAKE MEMBLOCK FROM PATH

This command will create a memblock from the specified path, if the memblock already exists it will first be deleted. You can then delete the path, modify the memblock, and use *AI Make Path From Memblock* to modify the path.

The structure of the memblock is as follows:
Number of points as DWORD (4 bytes)

Then for each point:
X# as FLOAT (4 bytes)
Z# as FLOAT (4 bytes)

Syntax
AI Make Memblock From Path *Memblock Number, Path Number*

Parameters
Memblock Number, The id of the memblock you want to hold the data.
Path Number, The id of the path you want to use to fill the memblock.

Return
n/a

AI MAKE PATH FROM MEMBLOCK

This command creates a path from the specified memblock. You can use this to create a complete path from scratch or to copy a path from one id to another along with *AI Make Memblock From Path.*

The structure of the memblock should be as follows:
Number of points as DWORD (4 bytes)

Then for each point:
X# as FLOAT (4 bytes)
Z# as FLOAT (4 bytes)

Syntax
AI Make Path From memblock *Path Number, Memblock Number*

Parameters
Path Number, The id of the path you want to create from the memblock, should not exist.
Memblock *Number,* The id of memblock you want to use as source data.

Return
n/a

ZONES

Zones define areas of visibility that can be assigned to one or more entities. When a valid target enters a zone all entities assigned to the zone are notified of the event and can act as if the target entered their line of sight. Entities will make their way to the event in order to check it out, and whilst the target remains in the zone their position will be known to assigned entities. Entities rate the threat of zone targets as lower than normal targets so an entity will prefer to seek out targets it recently lost before moving to zone targets.

Zones can be assigned to friendly or enemy entities or both at the same time, and can replace their field of view. If an entity's view range is set to 0 they will only be able to see targets that enter any assigned zones, and as such any target that leaves the zone will be lost by such an entity. Alternatively entities with a valid view range can be set to follow targets out of the zone for as long as they can see them using an aggressive stance.

Zones can be restricted to a specific container so that only entities from that container, which enter the zone, will trigger an event. Zones can be assigned to entities from any container regardless of which container the zone is in, this allows an unusual case in which, for example, entities in container 1 can be assigned a zone from container 0 and be able to see entities from container 0 that enter the zone.

This could be used to simulate a balcony scenario where the balcony is container 1 and the ground floor is container 0, the zone can be created to cover the area of the ground floor visible from the balcony and assigned to all entities on the balcony. By creating a half height obstacle around the edge of the balcony entities will be prevented from leaving the balcony whilst still being able to see over it. Entities that enter the ground floor zone will then be spotted by the entities on the balcony and shoot at them. There are some restrictions to this method, any entity on the ground floor that ducks will be hidden by the half height balcony obstacle and not be shot at, obstacles on the ground floor container will not obstruct the line of sight of entities on the balcony zones provide a region of complete visibility (you could use multiple overlapping zones to restrict visibility only to areas you want), entities on the balcony but not near the edge will still act as if they can see directly to the ground floor, any balcony entity that is assigned the zone will be notified of entities entering the zone no matter where they are (you may want to assign entities to the zone only when they are on the balcony). The ground floor entities will not be able to see the balcony entities, but you could apply the same method by creating a zone on the balcony in container 1 and assigning it to all the entities on the ground in container 0 to achieve the same effect.

AI ADD ZONE

This command creates a square zone defined by the two opposing corners *MinX, MinZ* and *MaxX, MaxZ,* and acts as an area of visibility that can be assigned to entities using *AI Entity Assign Zone*.

The container number parameter allows you to specify a container to limit the zone to, with which only entities from that container will trigger a response from anyone assigned to the zone. This can be useful for preventing entities from overlapping containers from triggering each others zones. The default is -1 for all containers. This can be used to achieve a special effect that entities assigned a zone from another container can see entities from that container. See the Zones description for details and an example usage

The *Zone Number* must be unique across all containers, no two zones with the same id can exist at the same time.

Syntax
AI Add Zone *Zone Number, MinX#, MinZ#, MaxX#, MaxZ#*
AI Add Zone *Zone Number, MinX#, MinZ#, MaxX#, MaxZ#, Container Number*

Parameters
Zone Number, the id you want to use to represent this zone.
MinX#, The minimum X bound of the zone.
MinZ#, The minimum Z bound of the zone.
MaxX#, The maximum X bound of the zone.
MaxZ#, The maximum Z bound of the zone.
Container Number, (optional) the id of a container to restrict the zone to.

Return
n/a

AI REMOVE ZONE

This command removes the zone from the AI system and from any entities that were assigned to it.

Syntax
AI Remove Zone *Zone Number*

Parameters
Zone Number, the id of the zone you want to remove.

Return
n/a

AI ZONE EXIST

This command checks if the specified zone already exists within the system.

Syntax
return integer = AI Zone Exist (*Zone Number*)

Parameters
Zone Number, The id of the zone you want to check exists.

Return
1 if the zone exists, 0 if not.

SOUNDS

Sounds act as markers to entities that can hear them and create points of interest that entities can choose to investigate. Entities respond to sounds in automatic mode depending on their stance, which can be set to make them run away from or move towards sounds. In manual mode an entity will still detect sounds but it is up to you to move them in response.

Sounds are not created automatically by the AI system so all sounds should be added using the *AI Create Sound* command. Sounds exist for a short period of time, usually about a second, and are then remove automatically. During this time any entity that moves within range of the sound can hear it and respond to it.

Sounds can be given a priority level that will make some sounds more important than others, such as gun shots compared with foot steps. Given a choice between two sounds being heard at the same time an entity will investigate the high priority sound. An entity can also have a choice between following up on a lost target, by going to its last position, or investigating a sound it has just heard. In which case if the sound type is less than 10 the entity will go to the target, otherwise it will investigate the sound. If an entity can see a target it will always chose to attack the target over any sound heard.

AI CREATE SOUND

This command creates a sound at the specified position that can be detected by nearby entities. Sounds are not created automatically so they must all be created using this command. You can choose when and where sounds, such as gun shots, are created and do not have to represent any real event. When in automatic mode, entities will respond to sounds depending on their stance, more aggressive stances will move to the point of the sound to look for targets.

The type of sound determines how important the entity will rate a particular sound when heard, for example footsteps, gunshots or explosions. Higher type values are regarded as more important and will be investigated before lower values.

The radius is how large the sound is, if the sound radius plus the entity hearing range is greater than the distance between the two, the sound can be heard by the entity.

The container parameter allows you to limit the sound to a particular container, so that only entities from that container will hear the sound. The default is -1 for all containers.

Syntax
AI Create Sound *Position X#, Position Z#, Type, Radius#*
AI Create Sound *Position X#, Position Z#, Type, Radius#, Container Number*

Parameters
Position X#, The X position where you want to sound to be.
Position Z#, The Z position where you want to sound to be.
Type, The type of sound as a positive integer.
Radius#, The radius of the sound.
Container Number, (optional) The id of a container to limit the sound to.

Return
n/a

DEBUG COMMANDS

The debug commands allow you to view the internal data of the AI system in the form of Dark Basic 3D objects. Free objects for all debugging information are found by starting at object ID 65535 and decreasing until a free object is found. This object is then occupied until the debug data is hidden again using the appropriate command, at which point the object is deleted.

Some frequently changing debug objects, such as avoidance angles, can cause a performance hit whilst displaying due to calculations required to create the debug object displayed. Waypoints, waypoint edges, and obstacle bounds are only calculated once in the 'show' command, and remain static until hidden. If any changes are made to waypoints or obstacles after displaying the debug object it needs to be hidden and re-shown for the changes to display.

In addition to displaying data directly in the Dark Basic world, the console commands can be used to display detailed information about the parameters of an entity, such as destinations, targets and general parameters, which can sometimes help explain why an entity is behaving a certain way. As such it's probably most useful when using the manual entity commands to create custom behaviours.

Finally there are error control commands that allow you to suppress or display error messages in response to invalid parameters. The former is useful when there are parameters that are not known before hand, such as user defined or script defined values, that you do not want ending the program. Displaying error messages is the recommended mode in most cases.

AI DEBUG SHOW WAYPOINTS

Turns on the display of waypoints for the specified container. Waypoints mark points which entities can use to get around obstacles. These are shown as blue points along the edges of obstacles.

Syntax
AI Debug Show Waypoints *Container Number, Position Y#*

Parameters
Container Number, The id number of the container for which you want to display waypoints.
Position Y#, The Y height you want the debug objects to appear in world co-ordinates.

Return
n/a

AI DEBUG HIDE WAYPOINTS

Turns off the display of waypoints in the specified container.

Syntax
AI Debug Hide Waypoints *Container Number*

Parameters
Container Number, The id of the container for which you want to hide waypoint debug data.

Return
n/a

AI DEBUG SHOW WAYPOINT EDGES

Turns on the display of waypoint visibility between the waypoints of the specified container. If two waypoints are connected by an edge then they are not obstructed by any obstacles and entities can use it as a section of path during path finding. Edges are shown as blue lines joining waypoints.

Syntax
AI Debug Show Waypoint Edges *Container Number, Position Y#*

Parameters
Container Number, The id number of the container for which you want to show waypoint visibility.
Position Y#, The Y height that you want the debug objects to appear in world co-ordinates.

Return
n/a

AI DEBUG HIDE WAYPOINT EDGES

Turns off the display of waypoint visibility for waypoints in the specified container.

Syntax
AI Debug Hide Waypoint Edges *Container Number*

Parameters
Container Number, The id of the container for which you want to hide waypoint visibility.

Return
n/a

AI DEBUG SHOW OBSTACLE BOUNDS

Turns on the display of all obstacle boundaries in the specified container. These represent the edges of all obstacle in the specified container. Half height obstacle bounds appear in a darker green colour than full height obstacles. View blocking obstacles appear in cyan. The bounds objects will flash white about once a second to help visualise them against surrounding objects.

You need to specify the Y height of the debug objects since the obstacles themselves contain no Y axis data.

Syntax
AI Debug Show Obstacle Bounds *Container Number, Position Y#*

Parameters
Container Number, The id number of the container you wish to hide the obstacle bounds for.
Position Y#, The Y position at which you want the debug objects to appear in world co-ordinates.

Return
n/a

AI DEBUG HIDE OBSTACLE BOUNDS

Turns off the display of all obstacle boundaries in the specified container.

Syntax
AI Debug Hide Obstacle Bounds *Container Number*

Parameters
Container Number, The id number of the container you wish to hide the obstacle bounds for.

Return
n/a

AI Debug Show Paths

Turns on the display of all entity paths in all containers. These represent each entity's final destination and the path they've calculated to get there. Paths are shown as red points connected by red lines, starting at the entity's current position and ending at the entity's final destination.

Syntax
AI Debug Show Paths *Position Y#*

Parameters
Position Y#, The Y position you would like the debug objects to appear in world co-ordinates.

Return
n/a

AI Debug Hide Paths

Turns off the display of all entity paths in all containers.

Syntax
AI Debug Hide Paths

Parameters
n/a

Return
n/a

AI DEBUG SHOW VIEW ARCS

Turns on the display of entity view and hearing angles at the specified height. View angles are shown as transparent red circular arcs and represent the range and angle that an entity can see (if not obstructed by obstacles). Hearing ranges are shown as a yellow ring and any sounds within this ring can be heard by the entity.

Syntax
AI Debug Show View Arcs *Position Y#*

Parameters
Position Y#, The Y height at which you want the debug objects to appear, in world co-ordinates.

Return
n/a

AI DEBUG HIDE VIEW ARCS

Turns off the display of entity view and hearing ranges for all entities in all containers.

Syntax
AI Debug Hide View Arcs

Parameters
n/a

Return
n/a

AI DEBUG SHOW SOUNDS

Turns on the display of all sounds, as they occur, as yellow points at the specified height. Sounds are displayed for as long as they can be heard by surrounding entities.

Syntax
AI Debug Show Sounds *Position Y#*

Parameters
Position Y#, The Y Position you want sounds to appear at in world co-ordinates.

Return
n/a

AI DEBUG HIDE SOUNDS

Turns off the display of sounds.

Syntax
AI Debug Hide Sounds

Parameters
n/a

Return
n/a

AI DEBUG SHOW AVOIDANCE ANGLES

Turns on the display of entity avoidance angles in all containers. These represent areas where an entity has detected another entity and will not move in blocked directions. Avoidance angles are shown in green and contain the area where the entity will not move.

Syntax

AI Debug Show View Arcs *Position Y#*

Parameters

Position Y#, the height you want the debug objects to appear in world co-ordinates.

Return

n/a

AI Debug Hide Avoidance Angles

Turns off the display of all entity avoidance angles in all containers.

Syntax
AI Debug Hide Avoidance Angles

Parameters
n/a

Return
n/a

AI SET CONSOLE OUTPUT ON

This command opens a console window that displays detailed information about the specified entity such as destination, target and general parameter information. If the console window is closed manually an error will occur, us AI Set Console Output Off to safely remove the window.

Syntax
AI Set Console Output On *Entity Number*

Parameters
Entity Number, The id of the entity for which you want to display detailed information.

Return
n/a

AI SET CONSOLE OUTPUT OFF

This removes any console window currently displaying detailed entity information.

Syntax
AI Set Console Output Off

Parameters
n/a

Return
n/a

AI SHOW ERRORS

This command returns error displaying to the normal mode where invalid parameters result in an error message.

Syntax
AI Show Errors

Parameters
n/a

Return
n/a

AI HIDE ERRORS

This command suppresses all error messages that may occur from using invalid parameters in AI commands. Instead any failed commands will return control to the Dark Basic program and make no change to the AI system. When using this command you should check actions such as adding an entity succeed by using the *AI Entity Exist* command and handle any errors yourself.

Syntax
AI Hide Errors

Parameters
n/a

Return
n/a

PLAYER

The player is added to the AI system only to provide entities with the current position of the player. This can be done directly through the use of a DarkBasic object linked to the player or through the use of *AI Set Player Position* to do it manually. The AI system does not move or change the player in any way nor does it calculate paths or restrict the player from moving through obstacles. It is only used to provide the system with information about where the player is.

You can only add one player to the AI system and it is automatically placed on the friendly team.

AI ADD PLAYER

This command adds a player to the AI system, and optionally an object to use as positional information. The AI system does not move or change the object in any way, it only uses it to read the new player position directly from the object to simplify the process. If you do not want this to occur you can use 0 for both *Object Number* and *Use Object* and use the *AI Set Player Position* command to notify the AI system of the player's positions.

Only one player can be added to the system.

Syntax
AI Add Player *Object Number*
AI Add Player *Object Number, Use Object*

Parameters
Object Number, The id of the object that represents the player (ignored if not using the object)
Use Object, (optional) 1 to link the player to the specified object, 0 to not use the object. Default is 1

Return
n/a

AI KILL PLAYER

This command removes the player from the AI system so that entities can no longer target it. Any object that was linked to the player remains and you should remove this yourself.

Syntax
AI Kill Player

Parameters
n/a

Return
n/a

AI GET PLAYER X

This command returns the X component of the player position that the AI system is using as the current player position. If the player is linked with an object then this returns the position of the object. If it is not linked then this returns the internal position that is currently being used.

Syntax
return float = AI Get Player X ( )

Parameters
n/a

Return
The X component of the player's current position.

AI GET PLAYER Z

This command returns the Z component of the player position that the AI system is using as the current player position. If the player is linked with an object then this returns the position of the object. If it is not linked then this returns the internal position that is currently being used.

Syntax
return float = AI Get Player Z ( )

Parameters
n/a

Return
The Z component of the player's current position.

AI SET PLAYER POSITION

This command sets the player position when no object is linked to the player. If an object is linked it will also position the object, but in this case using either this command or positioning the DarkBasic object manually is an acceptable method to position the player. This position is used by entities to determine if they can see and attack the player. It is also used by allies when they are set to follow the player.

Syntax
AI Set Player Position *X#, Z#*

Parameters
X#, The X component of the new position.
Z#, The Z component of the new position.

Return
n/a

AI SET PLAYER ANGLE Y

This command sets the internal Y angle for the player, if the player is linked to an object the object's Y angle is also set.

Syntax
AI Set Player Angle Y *Angle#*

Parameters
Angle#, The angle you want to set the player to.

Return
n/a

AI SET PLAYER DUCKING

This command sets the ducking state of the player. You should use this to notify the AI system if the player is currently ducking so that entities will not be able to see the player from behind half height objects.

Syntax
AI Set Player Ducking *Mode*

Parameters
Mode, 1 to set the player as ducking, 0 to set the player as standing.

Return
n/a

AI SET PLAYER CONTAINER

This command tells the AI system which container the player is currently in. This cannot be determined automatically because containers may be separated by height which is not recorded in the system. The only purpose for setting the player container is to avoid entities from other containers from being able to see the player, only entities in the same container will be able to see the player.

Syntax
AI Set Player Container *Container Number*

Parameters
Container Number, The id of the container you want the player to be in.

Return
n/a

AI GET PLAYER IN ZONE

This command will check if the player is in within the bounds of the specified zone.

Syntax
return integer = AI Get Player In Zone (*Zone Number*)

Parameters
Zone Number, The id of the zone you want to check the player is in.

Return
1 if the player is within the bounds of the zone, 0 if not.

ENTITIES

Entities represent the computer controlled players within the AI system and they will move and react according to their surroundings and the values of their parameters. These parameters can be set using commands beginning *AI Set Entity* ___ and retrieved using *AI Get Entity* ___ and usually have no immediate impact on the entity until an event occurs that they need to react to, such as discovering an enemy. The commands beginning *AI Entity* ___ are more like commands to the entity to tell it to do something and usually have an immediate effect on the entity's movement, depending on their current mode.

Entities have two main modes of operation, *automatic mode* and *manual mode.* The default is automatic mode where an entity will react according to its stance, which can be one of *Run Away, Stationary, Cautious*, or *Aggressive* (see *AI Set Entity Stance*). In manual mode you are free to instruct the entity to behave and move where you want by using the range of manual mode commands, which provide much of the functionality that automatic mode uses to move entities around.

In both modes entities automatically avoid obstacles and other entities in their current container when moving about, without a need for a separate collision system. However, dynamic obstacles such as crates should not be added to the AI system, instead use an external collision/physics system to detect collisions with dynamic obstacles and use *AI Set Entity Collide.*

Entities are grouped into three teams, the player's team, the enemy team, and a neutral team. The player's team help the player by attacking enemies, enemies attack the player and its allies. Neither will attack neutral entities, and neutral entities will not attack anyone.

AI ADD ENEMY

This command adds an entity to the enemy team (against the player) using the id that you specify. This should be a non-zero positive integer, and not already be assigned to an entity.

You can call this command with one, two or three parameters:

The second parameter allows you to link the entity with the Dark Basic object of the same id, which will allow the AI system to move this object for you in response to any entity movement. The AI system will move the object in the X and Z directions and will not change the objects Y position. If you choose to disabled it you can retrieve the AI entity position with *AI Get Entity X()* and *AI Get Entity Z()* and use it to position your own representation of the entity.

The third parameter allows you to choose which container to assign the entity to, the entity will be bound by the obstacles within the specified container and will not automatically move to another container. You can change an entity's container later by using *AI Set Entity Container* command. If you are not using containers you can ignore this value.

Syntax
AI Add Enemy *Entity Number*
AI Add Enemy *Entity Number, Use Object*
AI Add Enemy *Entity Number, Use Object, Container Number*

Parameters
Entity Number, The id you want to use to represent this entity, when set to use a Dark Basic object this must be the same as the object id.
Use Object, (optional) Set this to 1 to link the entity directly to the Dark Basic object with the same id as the entity. Use 0 if you want to create an entity that has no links to any objects. The default value is 1.
Container Number, (optional) The id of the container you want to add this entity to, the default value is 0.

Return
n/a

AI ADD FRIENDLY

This command adds an entity to the friendly team (allied with the player) using the id that you specify. This should be a non-zero positive integer, and not already be assigned to an entity.

You can call this command with one, two or three parameters:

The second parameter allows you to link the entity with the Dark Basic object of the same id, which will allow the AI system to move this object for you in response to any entity movement. The AI system will move the object in the X and Z directions and will not change the objects Y position. If you choose to disabled it you can retrieve the AI entity position with *AI Get Entity X()* and *AI Get Entity Z()* and use it to position your own representation of the entity.

The third parameter allows you to choose which container to assign the entity to, the entity will be bound by the obstacles within the specified container and will not automatically move to another container. You can change an entity's container later by using *AI Set Entity Container* command. If you are not using containers you can ignore this value.

Syntax
AI Add Friendly *Entity Number*
AI Add Friendly *Entity Number, Use Object*
AI Add Friendly *Entity Number, Use Object, Container Number*

Parameters
Entity Number, The id you want to use to represent this entity, when set to use a Dark Basic object this must be the same as the object id.
Use Object, (optional) Set this to 1 to link the entity directly to the Dark Basic object with the same id as the entity. Use 0 if you want to create an entity that has no links to any objects. The default value is 1.
Container Number, (optional) The id of the container you want to add this entity to, the default value is 0.

Return

n/a

AI ADD NEUTRAL

This command adds an entity to the neutral team (will not fight) using the id that you specify. This should be a non-zero positive integer, and not already be assigned to an entity.

You can call this command with one, two or three parameters:

The second parameter allows you to link the entity with the Dark Basic object of the same id, which will allow the AI system to move this object for you in response to any entity movement. The AI system will move the object in the X and Z directions and will not change the objects Y position. If you choose to disabled it you can retrieve the AI entity position with *AI Get Entity X()* and *AI Get Entity Z()* and use it to position your own representation of the entity.

The third parameter allows you to choose which container to assign the entity to, the entity will be bound by the obstacles within the specified container and will not automatically move to another container. You can change an entity's container later by using *AI Set Entity Container* command. If you are not using containers you can ignore this value.

Syntax
AI Add Neutral *Entity Number*
AI Add Neutral *Entity Number, Use Object*
AI Add Neutral *Entity Number, Use Object, Container Number*

Parameters
Entity Number, The id you want to use to represent this entity, when set to use a Dark Basic object this must be the same as the object id.
Use Object, (optional) Set this to 1 to link the entity directly to the Dark Basic object with the same id as the entity. Use 0 if you want to create an entity that has no links to any objects. The default value is 1.
Container Number, (optional) The id of the container you want to add this entity to, the default value is 0.

Return

n/a

AI KILL ENTITY

This command removes the specified entity from the AI system, any DarkBasic object assigned to it remains and should be deleted manually. Other entities will remove this entity from their target list.

Syntax
AI Kill Entity *Entity Number*

Parameters
Entity Number, The id of the entity you want to remove from the AI system.

Return
n/a

AI ENTITY EXIST

This command checks if the specified entity has been added to the AI system. Entities are not removed from the system automatically once they have been added, to remove an entity use the *AI Kill entity* command.

Syntax
return integer = AI Entity Exist (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check exists.

Return
1 if entity exists, 0 if not.

AI ENTITY RESET

This command resets the state of an entity back to 'Idle' when in automatic mode. However the entity surroundings can change the state if the entity is allowed to attack or hear.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Reset *Entity Number*

Parameters
Entity Number, The id of the entity you want to reset.

Return
n/a

AI ENTITY FOLLOW PLAYER

This command tells the entity to follow and defend the player, as such this command is only for friendly entities that are allied with the player. If no player has been added or the entity is not allied with the player no change is made to the entity.

This command is similar to *AI Entity Defend Area* except that the defend position is constantly set to the player's current position. When the distance between the entity and the player exceeds the specified distance the entity will choose a point close to the player and move to it.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Follow Player *Entity Number, Follow Distance#*

Parameters
Entity Number, The id of the entity you want to follow the player.
Follow Distance#, The maximum distance you want the entity to be from the player.

Return
n/a

AI ENTITY SEPARATE

This command is used to stop an entity following and defending the player, it is the opposite to *AI Entity Follow Player*. This command removes the entity from a defending state and returns it to behaving in accordance with it's stance (see *AI Set Entity Stance*)

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Separate *Entity Number*

Parameters
Entity Number, The id of the entity you want to stop following the player.

Return
n/a

AI ENTITY DEFEND AREA

This command sets the entity into a defensive mode that will attempt to keep them within the specified area whilst still attacking any targets that come within range. The entity still allowed to perform normal evasive moves such as strafing but if it begins to leave the specified area it will automatically move itself back within the area.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Defend Area *Entity Number, Position X#, Position Z#, Radius#*

Parameters
Entity Number, The id of the entity you want to defend the area.
Position X#, The X component in world co-ordinates of the centre of the area to defend.
Position Z#, The Z component in world co-ordinates of the centre of the area to defend.
Radius#, The radius of the area you want the entity to defend, from the centre.

Return
n/a

AI ENTITY DEFEND POINT

Similar to *AI Entity Defend Area* this command sets the entity into a defensive mode but will keep the entity fixed at the specified point, where the only allowed response to attack is to duck/stand and return fire. This can be useful for making sure particular entities remain at key defence points and not wonder off following targets. This command resets the entity's idle position to the defend point.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Defend Point *Entity Number, Position X#, Position Z#*

Parameters
Entity Number, The id of the entity you want to defend the point.
Position X#, The X component in world co-ordinates of the point to defend.
Position Z#, The Z component in world co-ordinates of the point to defend.

Return
n/a

AI ENTITY SEARCH AREA

This command will change the entity state to search the area. This is automatically called for aggressive entities when they lose sight of all their targets, but entities in other stances will not follow or search for their targets. Use this command when you want to force an entity to search.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Search Area *Entity Number*

Parameters
Entity Number, The id of the entity you want to start searching.

Return
n/a

AI ENTITY GO TO POSITION

This command sets the entity's destination to the specified position, the entity will immediately calculate a path to the new destination and begin moving towards it. If the destination lies inside an obstacle an attempt will be made to choose the closest point which is not inside an obstacle. If this fails, or the path to the destination is blocked completely by obstacles the entity will not move.

This is an manual mode command (see *AI Set Entity Control)*

This command will work in automatic mode by changing the entity state, but surrounding conditions (such as a visible target) can distract the entity from your specified destination.

Syntax
AI Entity Go To Position *Entity Number, Position X#, Position Z#*

Parameters
Entity Number, The id of the entity you want to move
Position X#*,* The X component of the position in world co-ordinates that you want the entity to move to.
Position Z#*,* The Z component of the position in world co-ordinates that you want the entity to move to.

Return
n/a

AI ENTITY STOP

This command resets the entity destination to it's current location so it will stop moving. This does not effect turning.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Stop *Entity Number*

Parameters
Entity Number, The id of the entity you want to stop.

Return
n/a

AI ENTITY LOOK AT POSITION

This command turns the entity to look in the direction of the specified point. If the entity is moving and the view of the specified point is obscured by obstacles the entity will look in the direction of its current movement, until the point becomes visible. If the entity is stationary it will look at the point whether it is visible or not.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Look At Position *Entity Number, Position X#, Position Z#*

Parameters
Entity Number, The id of the entity you want to hold position.
Position X#, The X component of the position you want the entity to look at.
Position Z#, The Z component of the position you want the entity to look at.

Return
n/a

AI ENTITY LOOK AROUND

This command will make the entity choose a random direction to turn, between the two angles you specify. Angles are relative to the entity's current angle so an angle of 180 will turn the entity 180 degrees from it's current direction. The entity will choose randomly whether to turn left or right in this direction.

This command can be useful for making the entity check its current location for targets that maybe behind it or out of it's current view arc. If the entity is already turning this command has no effect. Use *AI Entity Look At Position* to specify an exact point to look at.

Angles should be in the range 0-180, if *min* and *max* angles are the same the entity will turn exactly that amount either left or right.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Look Around *Entity Number, Min Angle#, Max Angle#*

Parameters
Entity Number, The id of the entity you want to hold position.
Min Angle#, The minimum angle the entity will turn.
Max *Angle#,* The maximum angle the entity will turn.

Return
n/a

AI ENTITY RANDOM MOVE

With this command the entity will choose a random direction from 0-360 degrees, and move a random distance between *Min Distance* and *Max Distance* in that direction. If the direction is blocked by an obstacle the entity will move as far as the obstacle and stop.

This command can be useful in simulating a roaming entity by calling this command every so often to randomly move it about.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Random Move *Entity Number, Min Distance#, Max Distance#*

Parameters
Entity Number, The id of the entity you want to move.
Min Distance#, The smallest distance you want the entity to move.
Max Distance#, The largest distance you want the entity to move.

Return
n/a

AI ENTITY MOVE CLOSE

This command moves the entity close to a specified point. The entity picks a random point that is at most *Max Distance* from the position and moves to it. If many obstacles surround the specified point the chosen point maybe on the opposite side of an obstacle, the entity will attempt to choose a point that is not inside an obstacle. Where it fails to do this (due to many overlapping obstacles) the entity will not move.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Move Close *Entity Number, Position X#, Position Z#, Max Distance#*

Parameters
Entity Number, The id of the entity you want to move.
Position X#, The X component of the position you want the entity to move towards.
Position Z#, The Z component of the position you want the entity to move towards.
Max Distance#, The maximum distance the entity will be from the specified position after moving.

Return
n/a

AI ENTITY MOVE TO COVER

This command will tell the entity to pick a destination from which it can no longer see the specified position. It checks nearby waypoints to see which would provide cover from the position and moves to the closest, preferring points behind it. If no nearby waypoints can be found, or none provide cover, the entity moves directly away from the position. If the entity is already covered from the specified position and cannot see it, it does not move.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Move To Cover *Entity Number, Position X#, Position Z#*

Parameters
Entity Number, The id of the entity you want to move.
Position X#, The X component of the position you want the entity to hide from.
Position Z#, The Z component of the position you want the entity to hide from.

Return
n/a

AI ENTITY MOVE TO IDLE POSITION

This command sets the entity destination to its stored idle position. This can be set using *AI Set Entity Idle Position* and defaults to the position the entity was when it was added to the AI system.

This command replicates the usual behaviour of an entity in the idle state in automatic mode. In manual mode this feature can be used to remember a position and move to it using this command.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Move To Idle Position *Entity Number*

Parameters
Entity Number, The id of the entity you want to move.

Return
n/a

AI ENTITY MOVE TO CLOSEST SOUND

This command will move the entity towards the closest detected sound, if the entity has not heard a sound it will not move. You can detect if the entity has heard a sound using the *AI Get Entity Heard Sound command.*

The entity will automatically calculate any necessary paths to its destination.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Move To Closest Sound *Entity Number*

Parameters
Entity Number, The id of the entity you want to move.

Return
n/a

AI ENTITY MOVE AWAY FROM SOUND

This command will move the entity away from the closest sound. The entity will pick a point in the opposite direction from the sound and move towards it. You can check if the entity has heard anything using *AI Get Entity Heard Sound* command. If the entity has not heard a sound this command has no effect.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Move Away From Sound *Entity Number*

Parameters
Entity Number, The id of the entity you want to move.

Return
n/a

AI ENTITY ADD TARGET

This command will add an entity or player (*Target ID*) to another entity's target list (*Entity Number*) which will then be regarded as a valid target and fired upon as if the target was an enemy of the entity. The target will remain in the target list whilst the entity can see the target and for a short time after. Once the target has been lost for a certain amount of time it will be removed from the target list and will not be re-added unless it is re-spotted and is naturally a valid target or you add it again.

For example, normally enemy entities will only attack friendly entities or the player, but using this command you can add enemy entities to each others target lists and they will attack each other so long as they can see each other. To create a free for all style scenario you can create a group of entities, all on the same team, and use the *AI Get Entity Can See* command to check if each entity can see any other, and if so to add it to the target list using this command.

If the target already exists in the target list it will not be added again, so this command can safely be used many times on the same target. If you want to remove a target from the list before it loses sight use the *AI Entity Remove Target* command. The target does not have to exist, if it does not no change is made.

Entities are allowed any number of targets in their target list and it will be sorted automatically to select what the entity thinks is the biggest threat to it at the time, this is mostly based on distance.

Syntax
AI Entity Add Target *Entity Number, Target ID*

Parameters
Entity Number, The id of the entity you want to add the target to
Target ID, The id of the target object you want to add to the entity's target list.

Return
n/a

AI ENTITY REMOVE TARGET

This command will remove the specified target from the entity's target list, whether it was added automatically or by using the *AI Entity Add Target* command. If the target was added automatically and the *AI Set Entity Can Select Targets* parameter is still set to 1 then the target will be re-added as soon as the entity sees it.

If the entity does not have the specified target currently in its target list then no change is made to the entity.

Syntax
AI Entity Remove Target *Entity Number, Target ID*

Parameters
Entity Number, The id of the entity you want to remove the target from.
Target ID, The id of the target object you want to remove from the entity's target list.

Return
n/a

AI ENTITY STRAFE TARGET

This command will set the entity destination to a point that will move it sideways relative to the entity's current target. If the entity has no targets the entity will not move. Strafing is good to avoid getting hit as it moves the entity across the view of their target forcing them to continually update their point of fire. Call this command every so often to set a strafe movement, calling it continuously will reset the movement point too often and result in jumpy movement.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Strafe Target *Entity Number*

Parameters
Entity Number, The id of the entity you want to strafe.

Return
n/a

AI ENTITY HOLD POSITION

This command will stop the entity, move them back to their idle position and prevent them from setting their destination. It sets the entity stance to 2, which corresponds to a stationary behaviour. The entity will still turn and react to visible targets but will not move in response.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Hold Position *Entity Number*

Parameters
Entity Number, The id of the entity you want to hold position.

Return
n/a

AI ENTITY DUCK

This command tells the AI system that the entity should act as if it was ducking. Ducking means the entity cannot see, or be seen, over half height obstacles (see *Obstacles*) and moves at a slower speed. Any 3D object linked to the entity will not be changed in any way since there are many different ways of displaying a ducking stance, so it is left to the programmer to implement in their world.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Duck *Entity Number*

Parameters
Entity Number, The id of the entity you want to set as ducking

Return
n/a

AI ENTITY STAND

This command sets the entity to behave as if it was standing, this is the opposite of the *AI Entity Duck* command, the default mode is standing. Standing entities can see and shoot over half height obstacles, but cannot see entities that are ducking behind half height obstacles. In automatic mode, entities duck and stand depending on their stance and surroundings.

This is a manual mode command (see *AI Set Entity Control)*

Syntax
AI Entity Stand *Entity Number*

Parameters
Entity Number, The id of the entity you want to stand.

Return
n/a

AI ENTITY ASSIGN ZONE

Use this command to assign an existing zone to an entity that will provide an area of complete visibility for the entity. Any opposing entity or player that enters the specified zone will be added to the entity's target list and acted upon. Multiple zones can be assigned to entities at the same time.

Zones can replace or add to an entity's view range, so that if the entity's view range is set to 0 it will still be able to see enemies that enter any assigned zones. If the entity's target leaves such a zone the entity will act as if it lost sight of its target.

Syntax
AI Entity Assign Zone *Entity Number, Zone Number*

Parameters
Entity Number, The id of the entity you want to assign the zone to.
Zone Number, The id of the zone you want to assign to the entity.

Return
n/a

AI ENTITY REMOVE ZONE

This command removes an entity from a previously assigned zone so that the entity will no longer be notified of any target that enters the zone.

Syntax
AI Entity Remove Zone *Entity Number, Zone Number*

Parameters
Entity Number, The id of the entity you want to remove from the zone.
Zone Number, The id of the zone you want to remove the entity from.

Return
n/a

AI ENTITY ASSIGN PATROL PATH

This command assigns an existing path to the specified entity for use as a set of patrol points. If the specified path contains one or more points then the entity's idle state is replaced with a patrol state, such that whenever the entity would be idle, it now patrols along the points in the patrol path. Using this command with a path with no points will return the entity's idle state.

When the entity reaches the end of the path it returns to the first point and cycles around again. Any changes made to the path at the specified number instantly effect the patrol path of any entities using it as a patrol path.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Entity Assign Patrol Path *Entity Number, Path Number*

Parameters
Entity Number, The id of the entity you want to assign the path to.
Path Number, The id of the path you want to assign to the entity.

Return
n/a

AI GET ENTITY X

This command gets the current position of the entity. If the entity is linked to an object then it returns the current position of the object. If it is not linked to an object this returns the internal position of the entity and you should use this to position your representation of the entity.

Syntax
return float = AI Get Entity X (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check.

Return
The X component of the entity's position.

AI GET ENTITY Z

This command gets the current position of the entity. If the entity is linked to an object then it returns the current position of the object. If it is not linked to an object this returns the internal position of the entity and you should use this to position your representation of the entity.

Syntax
return float = AI Get Entity Z (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check.

Return
The Z component of the entity's position.

AI GET ENTITY MOVE X

This command returns the distance the entity moved in the X direction during the last call to *AI Update*.

Syntax
return float = AI Get Entity Move X (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check.

Return
The X component of the entity's move direction.

AI GET ENTITY MOVE Z

This command returns the distance the entity moved in the Z direction during the last call to *AI Update*.

Syntax
return float = AI Get Entity Move Z (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check.

Return
The Z component of the entity's move direction.

AI GET ENTITY DESTINATION X

This command returns the entity's final destination that it is currently heading towards. This does not include any paths the entity is taking to avoid obstacles which involve an intermediate destination to the path points.

Syntax
return float = AI Get Entity Destination X (*Entity Number*)

Parameters
Entity Number, The id of the entity for which you want to get the destination.

Return
The X component of the entity's destination.

AI GET ENTITY DESTINATION Z

This command returns the entity's final destination that it is currently heading towards. This does not include any paths the entity is taking to avoid obstacles which involve an intermediate destination to the path points.

Syntax
return float = AI Get Entity Destination Z (*Entity Number*)

Parameters
Entity Number, The id of the entity for which you want to get the destination.

Return
The Z component of the entity's destination.

AI GET ENTITY TARGET X

This command gets the position of the target at the top of the entity's target list. The top target is one that is the closest and visible. If no targets are visible the top target is the one that was most recently visible.

Syntax
return float = AI Get Entity Target X (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check.

Return
The X component of the entity's target's position.

AI GET ENTITY TARGET Z

This command gets the position of the target at the top of the entity's target list. The top target is one that is the closest and visible. If no targets are visible the top target is the one that was most recently visible.

Syntax
return float = AI Get Entity Target Z (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to check.

Return
The Z component of the entity's target's position.

AI GET ENTITY ANGLE Y

This command returns the current Y angle the entity is facing. If the entity is linked to an object the Y angle of the object is returned. If the entity is not linked to an object then the internal Y angle that the entity is facing is returned and you should use this value to rotate your own representation to the correct angle.

Syntax
return float = AI Get Entity Angle (*Entity Number*)

Parameters
Entity Number, The id of the entity for which you want to get the Y angle.

Return
The Y angle of the entity in degrees.

AI GET ENTITY STATE$

This command gets a string description of the current state which is controlling the entity. In automatic mode the entity is controlled by a finite state machine which contains a set of states each with a different behaviour. An entity switches states depending on its surroundings for example the 'attack' state is used when the entity spots a target. This command can be useful to see why an entity is doing what it is doing.

The states that can be returned are:
Manual – The entity is under manual control and will not do anything itself.
Idle – The entity has nothing to do and will return to its idle position.
Patrol – The entity will patrol its patrol path.
Go To Destination – The entity will move to its destination.
Investigate – The entity has detected something (e.g. a sound) and will investigate it.
Attack – The entity has spotted a target and will attempt to attack it.
Run And Attack – The entity has spotted a target and will move towards it.
Strafe And Attack – The entity has spotted a target and is moving sideways.
Defend – The entity is will return to its defend area.
Search Area – The entity will check a possible path a target could have taken.
Wait In Cover – The entity will duck behind available cover and wait to pop up and shoot.

Syntax
return string = AI Get Entity State$ (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
A string representing a description of the state.

AI GET ENTITY ACTION$

This command gets a string description of the entity's current action, such as moving or attacking. This will tell you in which direction the entity is moving and if it is currently in an attack state. If an entity is attacking it may be shooting, if it is not attacking it will not be shooting. This can be used to set the entity's animation to an appropriate walking direction, or you can use *AI Get Entity Move X( )* and *AI Get Entity Move Z( )* to calculate this yourself.

The possible returned values are:
Stopped
Moving Forwards
Moving Backwards
Strafing Left
Strafing Right
Stopped And Attacking
Moving Forwards And Attacking
Moving Backwards And Attacking
Strafing Left And Attacking
Strafing Right And Attacking

Syntax
return string = AI Get Entity Action$ (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
A string representing a description of the current action.

AI GET ENTITY CAN FIRE

This command will return true if the entity has a target, can see it, and the target is within the fire angle set with *AI Set Entity Fire Arc*. Entities will not fire automatically as there are many different ways to represent an attack so you will need to create your own bullets and move them.

Syntax
return integer = AI Get Entity Can Fire (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to know can fire.

Return
1 if the entity can fire, 0 if not.

AI GET ENTITY CAN SEE

This command will return if the entity can see the specified point. If the entity is ducking, or *Height* is 0 then half height objects will register as blocking the entity's view. This command will return which view angle the point is in, if it can be seen, in the order specified below. If the point is beyond the entity's view range (see *AI Set Entity View Range*) then 0 will be returned.

Syntax
return integer = AI Get Entity Can See (*Entity Number, X#, Z#, Height*)

Parameters
Entity Number, The id of the entity you want to know can see.
X#, The X position of the point to see.
Z#, The Z position of the point to see.
Height, 1 if the point to see is at eye level, above half height objects, 0 if low.

Return
2 if the the point is within the inner view arc, 1 if the point is within the outer view arc, 0 if the point cannot be seen.

AI GET ENTITY COUNT TARGETS

This command returns the number of targets the entity has in its target list. Targets remain in the entity's list for a short time after it loses sight of it which allows the entity to remember where previous targets were. As such not all targets in the list may currently be visible but gives a rough idea of how outnumbered an entity may be.

This command works in both manual and automatic mode, in manual mode an entity just doesn't act against the targets.

Syntax
return integer = AI Get Entity Count Targets (*Entity Number*)

Parameters
Entity Number, The id of the entity to get the number of targets.

Return
The number of targets the entity has.

AI GET ENTITY TARGET ID

This command will get the id of the target from the specified index in the entity's target list. The target list contains all opponents that have been spotted by this entity including those that have recently disappeared. The list is sorted in order of distance for visible targets, with non-visible targets stored after all visible targets sorted by the time they were last seen. After a few seconds of not being seen a target will be removed from the target list.

If *Target Index* is not in the range 1 up to and including the value returned by *AI Get Entity Count Targets* then this command returns 0.

Syntax
return integer = AI Get Entity Target ID (*Entity Number, Target Index*)

Parameters
Entity Number, The id of the entity for which you want to retrieve the container.
Target Number, The index of the target from the entity target list you want the id of. Starts at 1.

Return
The id of the target entity or player, 0 if not found.

AI GET ENTITY TEAM

This command will return an integer representing the team the entity is assigned to.

Syntax
return integer = AI Get Entity Team (*Entity Number*)

Parameters
Entity Number, The id of the entity for which you want to retrieve the container.

Return
0 for the neutral team, 1 for the friendly team or 2 for the enemy team.

AI GET ENTITY IS MOVING

This command will return if the entity is currently moving towards its destination. If the entity has no destination this will return 0. If the entity has a destination but cannot reach it, or is stuck, this will return 0.

Syntax
return integer = AI Get Entity Is Moving (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
1 if the entity is moving, 0 if not.

AI GET ENTITY IS TURNING

This command will return if the entity is currently turning to face a specified point. If no look at point has been given (see *AI Set Entity No Look At Point*) this will return true whilst the entity is moving. This can be used with *AI Entity Look Around* to detect when an entity has reached its intended angle.

Syntax
return integer = AI Get Entity Is Turning (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
1 if the entity is turning, 0 if not.

AI GET ENTITY IS DUCKING

This command returns if the entity is currently ducking or standing. In automatic mode an entity may occasionally duck according to its state and stance. You should use this command to detect when an entity is ducking and represent that in the object, as there are many ways to represent ducking the AI system will not do it automatically.

Syntax
return integer = AI Get Entity Is Ducking (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
1 if the entity is ducking, 0 if not.

AI GET ENTITY AVOIDING

This command returns if the entity is involved in an avoiding movement around some unexpected object from the *AI Set Entity Collide* command. Once this command returns 0 then the entity will assume it has avoided the object and attempt to continue on its normal path. This command will not notify when an entity avoids an expected object such as an obstacle or other entity.

Syntax
return integer = AI Get Entity Avoiding (*Entity Number*)

Parameters
Entity Number, The id of the entity you want to know is avoiding.

Return
1 if the entity is avoiding something, 0 if not.

AI GET ENTITY HEARD SOUND

This command returns if the entity has heard a sound during the last call of the *AI Update* command.

Syntax
return integer = AI Get Entity Heard Sound (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
1 if the entity has heard a sound, 0 if not.

AI GET ENTITY STANCE

This command will get the current stance of the entity. The stance can be defined using *AI Set Entity Stance*, and is not changing by the AI system. See *AI Set Entity Stance* for the different behaviours that the stances correspond to.

In manual mode this value has no effect and can be used to store an integer value for any purpose.

Syntax
return integer = AI Get Entity Stance (*Entity Number*)

Parameters
Entity Number, The id of the entity to check.

Return
The entity stance as a positive integer.

AI GET ENTITY CONTAINER

This command will get the id of the container that the entity is currently assigned to, note that this is not the same as which container the entity may currently be standing in as the AI system cannot tell where one container ends and another begins. Use *AI Set Entity Container* to tell the AI system when an entity has changed containers. By default all entities are assign to container 0. If an entity has no container it cannot be updated.

Syntax
return integer = AI Get Entity Container (*Entity Number*)

Parameters
Entity Number, The id of the entity for which you want to retrieve the container.

Return
The id of the container the entity is assigned to. -1 if it has no container.

AI GET ENTITY IN ZONE

This command will check if the entity is in within the bounds of the specified zone. The entity does not need to be assigned to the zone, any entity can be checked against any zone.

Syntax
return integer = AI Get Entity In Zone (*Entity Number, Zone Number*)

Parameters
Entity Number, The id of the entity you want to check is within the zone.
Zone Number, The id of the zone you want to check the entity is in.

Return
1 if the entity is within the bounds of the zone, 0 if not.

AI SET ENTITY ACTIVE

This command sets whether the entity is updated when the *AI Update* command is called. When the entity is deactivated the AI system will not move the entity, in neither manual nor automatic mode, but you can still move the object in DarkBasic and the AI system will use the new position when re-activated. The default mode is 1.

Syntax
AI Set Entity Active *Entity Number, Mode*

Parameters
Entity Number, The id of the entity you want to set the active mode.
Mode, 0 to deactivate the entity, 1 for normal behaviour.

Return
n/a

AI SET ENTITY AVOID DISTANCE

This command sets the avoidance distance an entity will use when attacking another entity. If the target becomes closer than this distance, by either one of them moving, this entity will move away from the target to a point at least this distance from the target in a strafing backwards action (whether or not Can Strafe is set). This effects both Cautious and Aggressive stances.

This distance should be less than the entity's attack distance (see *AI Set Entity Attack Distance)*, avoidance distance overrides attack distance.

A value of 0 will cause the entity to never perform avoiding actions with its target but will still be bound by any entity-entity avoidance resulting from two entities getting closer than twice the radius apart. (see *AI Set Avoid Mode*).

Syntax
AI Set Entity Avoid Distance *Entity Number, Distance#*

Parameters
Entity Number, The id of the entity to set the distance.
Distance#, The distance for the entity to use.

Return
n/a

AI SET ENTITY ATTACK DISTANCE

This command sets the distance an entity will attack from when in an aggressive stance(stance 1). If the entity spots a target that is further than this distance it will move towards it until either this distance is reached or it becomes too close and moves to avoid the target (see *AI Set Entity Avoid Distance*).

This distance should normally be greater than twice the set radius otherwise the entity will attempt to continually move on top of its target, and will be prevented from doing so if entity-entity avoidance is on. (see *AI Set Avoid Mode*)

This distance should normally be greater than the entity's avoid distance (see *AI Set Entity Avoid Distance*) otherwise the entity will move towards the target, go beyond its avoid distance triggering it move further away, then move back towards the target possibly in rapid succession.

Syntax
AI Set Entity Attack Distance *Entity Number, Distance#*

Parameters
Entity Number, The id of the entity to set the distance.
Distance#, The distance for the entity to use.

Return
n/a

AI SET ENTITY CONTROL

Use this command to set an entity into automatic or manual control. The default is automatic mode.

In automatic mode an entity will move and respond to its surroundings, such as spotting an enemy and moving towards it, and you can use commands denoted as *automatic mode* commands to control entity behaviour. You can use *manual mode* commands, such as *AI Entity Look At Position*, but their effects can quickly be overridden by the entity so are not guaranteed to work.

In manual mode an entity will make no movements or actions by itself and you must use the commands denoted as *manual mode* commands to move the entity around. *Automatic mode* commands, such as *AI Entity Defend Area*, have no effect when an entity is in manual mode, except to modify the entity parameters that may take effect when the entity is next put into automatic mode.

Syntax
AI Set Entity Control *Entity Number, Mode*

Parameters
Entity Number, The id of the entity to set the control mode.
Mode, Use 1 to set automatic control, 0 to set manual control.

Return
n/a

AI SET ENTITY STANCE

This command sets the entity stance mode which controls behaviour during automatic mode control (see *AI Set Entity Control*). During manual control this has no effect except to store the value.

There are 4 main behaviours which have been implemented in automatic mode that you can use to define how you want entities to behave. These can be described as *Run Away, Stationary, Cautious* and *Aggressive*.

- *Run Away* is the default state for neutral entities and results in an entity moving away from any sound it hears or damage it receives from a known direction. Entities are still able to fire in this mode (except neutral entities) but are likely to quickly move out of sight and lose their target. (mode=3)

- *Stationary* limits the entity to their idle position but they are still allowed to return fire and duck. The entity will not investigate sounds nor move when attacked. (mode=2)

- *Cautious* allows the entity to move about freely whilst attacking a target but will not attempt to follow or search for a target that goes out of sight. Also it does not readily approach a target preferring to shoot from a distance. (mode=0)

- *Aggressive* is similar to cautious except that the entity will move towards targets and follow them when they go out of sight. It will also briefly search for targets that remain out of sight for extended periods before returning to its idle position. (mode=1)

You can then use the *AI Set Entity Can* ____ commands to refine the behaviour to further restrict or allow certain actions that the entity can perform.

Syntax
AI Set Entity Stance *Entity Number, Mode*

Parameters
Entity Number, The id of the entity to set the stance.
Mode, The new stance for the entity as a positive integer.

Return
n/a

AI SET ENTITY AGGRESSIVE

This command sets the entity stance to 1, which corresponds to an aggressive behaviour in automatic mode. See *AI Set Entity Stance* for a full list of behaviours.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Set Entity Aggressive *Entity Number*

Parameters
Entity Number, The id of the entity you want to set as aggressive.

Return
n/a

AI SET ENTITY SPEED

This command sets the entity movement speed in units per second. When calling *AI Update* it will use timer based movement to keep entities moving their defined speed regardless of the current frames per second. The default speed is 5 DarkBasic units per second. When the entity is ducking this speed is halved.

Syntax
AI Set Entity Idle Position *Entity Number, Speed#*

Parameters
Entity Number, The id of the entity to set the speed.
Speed#, The new speed in units per second.

Return
n/a

AI SET ENTITY TURN SPEED

This command sets the speed at which the entity can rotate in degrees per second. Calling *AI Update* will use timer based movement to keep the entity to its defined rotate speed regardless of the current frames per second. The default value is 240 degrees per second.

Syntax
AI Set Entity Turn Speed *Entity Number, Speed#*

Parameters
Entity Number, The id of the entity to set the turn speed.
Speed#, The new speed in degrees per second.

Return
n/a

AI SET ENTITY PATROL TIME

This command sets the average time an entity will wait at each patrol point along its patrol path. The actual time is worked out by taking this value and adding a random value between -0.5 and 0.5. So if you want to ensure the entity never waits at its patrol points you need to use a time value less than -0.5

This value is in seconds.

Syntax
AI Set Entity Patrol Time *Entity Number, Time#*

Parameters
Entity Number, The id of the entity to set the turn speed.
Time#, The time you want the entity to wait in seconds.

Return
n/a

AI SET ENTITY HEARING RANGE

This command sets the range at which an entity can hear sounds added to the AI system. Sounds can also have a radius of their own, if this radius plus the entity's hearing range is greater than the distance between the two then the sound is heard. If you want to stop the entity hearing sounds altogether use the *AI Set Entity Can Hear* command.

Syntax
AI Set Entity Hearing Range *Entity Number, Radius#*

Parameters
Entity Number, The id of the entity to set the hearing range.
Radius#, The radius within which the entity can detect sounds.

Return
n/a

AI SET ENTITY HEARING THRESHOLD

This command sets the lowest priority sound the entity can hear. If the sound type is less then this value the entity will not hear it and will not react to it. If you want to stop the entity hearing sounds altogether use the *AI Set Entity Can Hear* command.

Syntax
AI Set Entity Hearing Treshold *Entity Number, Threshold*

Parameters
Entity Number, The id of the entity to set the hearing range.
Threshold, The lowest type of sound to hear. Default is 0.

Return
n/a

AI SET ENTITY VIEW RANGE

This command sets the distance the entity can see from its current location. Anything within this distance and within its view angle can be seen by the entity. The default value is 60 DarkBasic units.

This command should be used with *AI Set Entity View Arcs* to completely define the area the entity can see.

Syntax
AI Set Entity View Range *Entity Number, Distance#*

Parameters
Entity Number, The id of the entity to set the view range.
Distance#, The maximum distance the entity can see.

Return
n/a

AI SET ENTITY VIEW ARC

This command will set the angles at which an entity can see its surroundings. The angles should be specified in degrees in the range 1-360 with 360 meaning the entity can see all around it, and 1 meaning the entity can only see the exact direction it is facing. The default value is 360 degrees.

The inner and outer angles allow you to divide up the view into two sections. Although only the outer angle matters during automatic mode, the *AI Get Entity Can See* command will return a different value depending on what angle contains the view point.

This command should be used with *AI Set Entity View Range* to completely define the area an entity can see.

Syntax
AI Set Entity View Arc *Entity Number, Inner Angle#, Outer Angle#*

Parameters
Entity Number, The id of the entity to set the view angles.
Inner Angle#, The inner angle that the entity can see, in degrees.
Outer Angle#, The outer angle that the entity can see, in degrees.

Return
n/a

AI SET ENTITY FIRE ARC

This command sets the angle at which the entity is able to fire. This angle is used in the *AI Get Entity Can Fire* command to calculate not only if an entity can see a target but also if the target is within this angle.

The angle should be specified in the range 1-360 with 360 meaning the entity can shoot all around itself and 1 meaning it can only shoot the direction it is facing.

Syntax
AI Set Entity Fire Arc *Entity Number, Angle#*

Parameters
Entity Number, The id of the entity to set the fire arc.
Angle#, The angle that the entity will be able to fire.

Return
n/a

AI SET ENTITY DEFEND DISTANCE

This command changes the distance that an entity will use as the radius when using *AI Entity Defend Area* or *AI Entity Follow Player*. The radius is used to keep the entity within the containing area.

This does not effect the *AI Entity Defend Point* action. If the entity is not involved in a defending action this command has no effect.

Syntax
AI Set Entity Defend Distance *Entity Number, Distance#*

Parameters
Entity Number, The id of the entity to set the distance.
Distance#, The distance for the entity to use.

Return
n/a

AI SET ENTITY DEFENDING

This command can be used to toggle the defend mode of the entity. When using the commands *AI Entity Defend Area* and *AI Entity Follow Player* the entity will remain in a defending state until the defend mode is set back to 0. This command will also remove the restriction on the entity from *AI Entity Defend Point* command and it be allowed to move about again.

This command can also be used to turn defending on, in which case the entity will act as if *AI Entity Defend Area* had been called with no change to any previously set defend point and radius.

Syntax
AI Set Entity Defending *Entity Number, Mode*

Parameters
Entity Number, The id of the entity to set the defending state.
Mode, 1 to set the entity as defending, 0 to stop defending.

Return
n/a

AI SET ENTITY IDLE POSITION

This command sets the position an entity will return to when in automatic mode, in the 'Idle' state, and is not allowed to roam (see *AI Set Entity Can Roam*). The default position is the position the entity was in when it was added to the AI system and can be changed at any time including whilst the entity is in the Idle state.

In manual mode this command has no effect but to store the position for use with *AI Entity Move To Idle Position*.

Syntax
AI Set Entity Idle Position *Entity Number, X#, Z#*

Parameters
Entity Number, The id of the entity to set the idle position.
X#, The X component of the new position.
Z#, The Z component of the new position.

Return
n/a

AI SET ENTITY POSITION

This command sets the current entity position both within the AI system and to any linked object. When linked to an object this is the preferred method of resetting the position of an entity as opposed to positioning the object in DarkBasic. When not linked to an object this command sets the position within the AI system and you should position any representation yourself.

Syntax
AI Set Entity Position *Entity Number, X#, Z#*

Parameters
Entity Number, The id of the entity to set the position.
X#, The X component of the new position.
Z#, The Z component of the new position.

Return
n/a

AI SET ENTITY ANGLE Y

This command sets the internal Y angle for this entity, if the entity is linked to an object the object's Y angle is also set.

Syntax
AI Set Entity Angle Y *Entity Number, Angle#*

Parameters
Entity Number, The id of the entity for which you want to set the Y angle.
Angle#, The angle you want to set the entity to.

Return
The Y angle of the entity in degrees.

AI SET ENTITY CAN DUCK

This command can be used to prevent an entity from ducking whilst in automatic mode, if you do not want to handle that behaviour. The default mode is to allow ducking and an entity will occasionally duck depending on its stance and surroundings. Ducking allows an entity to hide behind half height obstacles which normally do not obstruct sight.

Syntax
AI Set Entity Can Duck *Entity Number, Mode*

Parameters
Entity Number, The id of the entity for which you want to set the duck mode.
Mode, Use 1 to allow the entity to duck, 0 to prevent ducking.

Return
n/a

AI SET ENTITY CAN ATTACK

This command sets if the entity is allowed to attack other entities whilst in automatic mode, the default mode is to allow attacks. When disabled the entity will not enter any 'Attack' state but will still respond to other events such as sounds. The entity will still keep a target list but won't move to attack the targets.

Syntax
AI Set Entity Can Attack *Entity Number, Mode*

Parameters
Entity Number, The id of the entity you want to set the attack mode.
Mode, Use 1 to allow the entity to attack, 0 to prevent attacking.

Return
n/a

AI SET ENTITY CAN STRAFE

This command sets whether an entity is allowed to strafe a target in either automatic mode or manual mode. The default mode is to allow strafing. If enabled in automatic mode, an entity will attempt to strafe every so often whilst attacking to avoid taking damage from its targets.

Syntax
AI Set Entity Can Strafe *Entity Number, Mode*

Parameters
Entity Number, The id of the entity for which you want to set the strafe mode.
Mode, Use 1 to allow the entity to strafe, 0 to prevent strafing.

Return
n/a

AI SET ENTITY CAN SEARCH

This command sets whether an entity is allowed to search for a target when in aggressive mode and has moved to the target's last known position. By default this is set to 1, which allows the entity to briefly search likely points around the target's last known position. When set to 0 the entity will still move to the target's last known position but if the target still cannot be seen it will return to the idle state.

Syntax
AI Set Entity Can Search *Entity Number, Mode*

Parameters
Entity Number, The id of the entity for which you want to set the strafe mode.
Mode, Use 1 to allow the entity to search, 0 to prevent searching.

Return
n/a

AI SET ENTITY CAN HEAR

This command can be used to stop an entity responding to sounds in automatic mode. The default mode is to allow an entity to respond by investigating sounds within its range of hearing.

Syntax
AI Set Entity Can Hear *Entity Number, Mode*

Parameters
Entity Number, The id of the entity for which you want to set the hearing mode.
Mode, Use 1 to allow the entity to hear, 0 to prevent hearing.

Return
n/a

AI SET ENTITY CAN ROAM

This command can be used to make the entity move around whist in the 'Idle' state in automatic mode. The default mode is for the entity to remain at the idle position and not roam about. When allowed to roam an entity will randomly pick a nearby point every so often and move to it.

Syntax
AI Set Entity Can Roam *Entity Number, Mode*

Parameters
Entity Number, The id of the entity for which you want to set the roaming mode.
Mode, Use 1 to allow the entity to roam, 0 to prevent roaming.

Return
n/a

AI ENTITY CAN SELECT TARGETS

This command will set whether an entity can select targets based on its team and the target of the possible target. If the entities are on opposing teams (friendly/enemy) then they will automatically add each other as targets when they see each other. This is the default behaviour.

If the mode is set to 0 then the entity will not be allowed to select any targets based on teams and will only receive targets from the *AI Entity Add Target* command.

Syntax
AI Set Entity Can Select Targets *Entity Number, mode*

Parameters
Entity Number, The id of the entity for which you want to change the target selection mode.
Mode, 1 to allow the entity to select targets (default), 0 to prevent selection of targets.

Return
n/a

AI SET ENTITY NO LOOK AT POINT

This command removes any set point the entity has been given to focus on. If a point is set the entity will look in its direction so long as it has a clear line of sight to the point. If it does not have line of sight, or no point is set, it will look in the direction it is currently travelling.

This is a manual mode command (see *AI Set Entity Control*).

Syntax
AI Set Entity No Look At Point *Entity Number*

Parameters
Entity Number, The id of the entity to remove its look at point.

Return
n/a

AI SET ENTITY COLLIDE

Use this command to notify the entity when it has collided with an object that is not in the AI system, for example from an external collision or physics system. The entity will move sideways for a brief period to try and avoid this object, during which time *AI Get Entity Avoiding* will return 1. Since the entity has no knowledge of the object's size it may not succeed in going around it in one move. You should detect when the entity has decided to stop avoiding and repeat the *AI Set Entity Collide* command if necessary.

This does not apply to objects set as obstacles or entities in the AI system, the entity will avoid these automatically.

Syntax
AI Set Entity Collide *Entity Number, Collide Object*

Parameters
Entity Number, The id of the entity that has hit something.
Collide Object, The id of the object the entity has hit, 0 for none.

Return
n/a

AI SET ENTITY HIT

Use this command to notify an entity when it gets struck by a bullet or other noteworthy damage from a particular direction. In automatic mode the entity will respond by looking for the source of the damage and acting against it if found. The direction should be specified as the direction the the projectile was travelling when it hit, i.e. pointing towards the entity. In the case that a direction is not available use a value of 0 for both *X* and *Z* and the entity will look around itself.

Syntax
AI Set Entity Hit *Entity Number, X#, Z#*

Parameters
Entity Number, The id of the entity that is hit.
X#, The X direction that the projectile was travelling.
Z#, The Z direction that the projectile was travelling.

Return
n/a

AI SET ENTITY RUN AWAY WHEN HIT

This command sets the entity stance to 3, which corresponds to a run away behaviour. The entity will also move away from any sounds it detects within range. See *AI Set Entity Stance* for a full list of behaviours.

This is an automatic mode command (see *AI Set Entity Control)*

Syntax
AI Set Entity Run Away When Hit *Entity Number*

Parameters
Entity Number, The id of the entity to set the stance.

Return
n/a

AI SET ENTITY CONTAINER

This command sets the container the entity is assigned to. An entity will only avoid obstacles and only see entities that are in its current container. By default all entities and obstacles are added to container 0 creating a single complete world, but it does not detect differences in height. Use containers to separate floors so that entities only avoid obstacles that are on their floor. If you want to move entities between floors you will need to detect when an entity moves into a new container and use this command to notify the AI system.

Syntax
AI Set Entity Container *Entity Number, Container Number*

Parameters
Entity Number, The id of the entity to set the container.
Container Number, The id of the container you want to assign the entity to.

Return
n/a

TEAMS

There are three teams within the AI system, the friendly team, the enemy team and the neutral team. The friendly team assists the player by attacking the enemy team and can be set to follow the player around the map. The enemy team attacks the player and its allies and can be set to defend key points or allowed to roam about the map. The neutral team represents civilians that do not engage in any combat and by default run away from sounds and received damage. The run away behaviour can be changed but they will not keep a target list and therefore not attack anything. The friendly and enemy teams will not attack neutral entities.

AI TEAM FOLLOW PLAYER

This command sets all entities in the friendly team to follow the player to within the specified distance. This can be useful for making the players allies defend the player, any enemies encountered will most likely be distracted by the group of targets and only a few may engage the player. Using a negative value for the maximum distance preserves the entities' current defend distances, otherwise all entities will use the new value.

Alternatively use the *AI Entity Follow Player* command to make individual entities follow the player. You can also use the *AI Set Entity Defend Distance* to change the maximum distance on a per entity basis.

Syntax
AI Team Follow Player *Maximum Distance#*

Parameters
Maximum Distance#, The maximum distance the entities will allow between the player and themselves before moving closer.

Return
n/a

AI TEAM SEPARATE

This command sets all entities in the friendly team to stop following the player and return to normal behaviour. You can Also use *AI Entity Separate* to stop individual entities from following the player.

Syntax
AI Team Separate

Parameters
n/a

Return
n/a

CONTAINERS

Containers divide the system into separate areas which do not interact with each other. By default a single container (0) is created to represent the entire system, but you may create new containers to add to it.

Containers can be used to represent different floors each with their own set of obstacles and entities, such that an entity can occupy the same X and Z positions as an entity from another container but they will not interfere with each other. Entities can be moved between containers manually by detecting when an entity has moved into another container and setting it to the new container.

Path finding also occurs only within the container it is started from, allowing it to detect impossible paths sooner if containers are kept small, and also preventing an entity from leaving its container automatically.

AI CONTAINER EXIST

Checks if the specified container has been added to the AI system.

Syntax
return integer = AI Container Exist (*Container Number)*

Parameters
Container Number, the id number of the container you want to check exists.

Return
1 if container does exist, 0 if not.

AI REMOVE CONTAINER

Deletes the container from the AI system, any obstacles that belong to the container are deleted with it, any entities that are assigned to the container must be assigned a new container to remain active.

Syntax
AI Remove Container *Container Number*

Parameters
Container Number, the id of the container you want to remove

Return
n/a

AI SET CONTAINER ACTIVE

This allows you to set the active mode of a container and the entities it contains. A deactivated container will not be updated during the AI Update command and its entities will not move or react, even to manual commands. An active container behaves as normal.

Syntax
AI Add Container *Container Number, Active Mode*

Parameters
Container Number, the id number of the container you want to change
Active Mode, 0 to deactivate the container, 1 to active the container.

Return
n/a

AI RESET

This command will remove all data from the AI system and return it to its starting state. All obstacles, entities, etc. are removed from the system and must be re-added after this command.

Syntax
AI Reset

Parameters
n/a

Return
n/a

AI START

This command will create the AI system and must be called before any other AI command. It sets up function pointers to allow the AI system to interact with DarkBasic.

Syntax
AI Start

Parameters
n/a

Return
n/a

AI SET RADIUS

This command sets the global radius that will be used by all entities. This controls how much space is allowed between the edge of obstacles and the waypoints that define movement around them. Since entities use the waypoints to navigate around their world they will attempt to maintain at least *radius#* distance from all obstacles.

This command rebuilds the waypoint network for all containers which takes time and should not be used in your main loop.

Syntax
AI Set Radius *Radius#*

Parameters
Radius#, The global radius you want to use for all entities in the AI system.

Return
n/a

AI SET AVOID MODE

This command sets the avoidance mode for entities when two entities become closer than twice their radius apart. There are 4 modes to choose from,

No Avoidance (mode 3) entities will not avoid each other and are free to move through each other,

Delayed Update (mode 2) entities will avoid each other but only check for nearby entities only every so often, this can help increase the performance, compared with mode 1, when many entities are present but may result in some entities getting closer then normal or sometimes moving through each other.

Real-Time (mode 1) entities will avoid each other and check for other nearby entities every update, constantly changing positions can result in an entity jumping about as its movement direction changes every update. This was the only mode in the original release version and is default.

Smoothed (mode 0) entities will avoid each other as in mode 1 but their movement directions will be smoothed over 20 updates where rapid changes in movement is not desired. However, this may cause delays in movement when an entity has to make a rapid change of direction in normal circumstances.

This value can be changed whilst the system is in motion.

Syntax
AI Set Avoid Mode *Mode*

Parameters
Mode, The avoidance mode to use (0-3).

Return
n/a

AI UPDATE

This command will update and move all entities within the AI system. You should call this command once per frame (in the same manner as *sync*). The internal timer will make sure that objects move in accordance with their set speed in units per second, you can override this by using the optional *Time Step* parameter to advance the internal state by *Time Step* seconds.

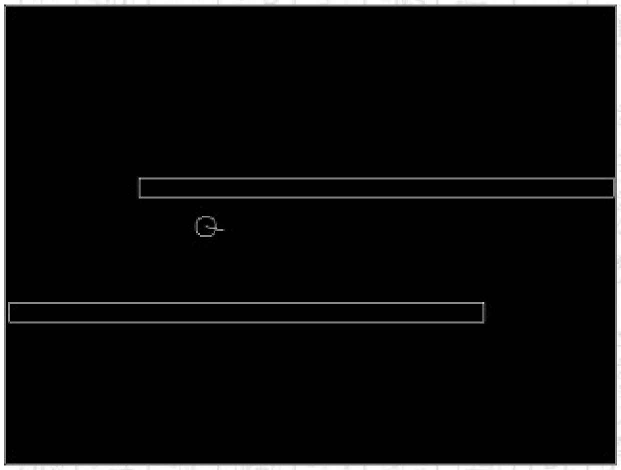
The recommended usage is to call *AI Update* with no parameter.

Syntax
AI Update
AI Update *Time Step#*

Parameters
Time Step#, (optional) This amount of time you want to move the scene on in seconds.

Return
n/a

2D DEMO



This demo shows the use of the AI system in a non-3D environment and details the methods to move entities around when the AI system cannot move your entities automatically.

This section will detail the AI commands used to construct this demo in the context they are used. For general use of each command you can look them up in the command list section of the help file.

AI Start
AI Set Radius 10.0

The AI Start command creates the AI system and is always called first, the Set Radius command sets the radius for all entities and is usually the second command called. The radius should be set in the main setup code since it can become an expensive operation when many obstacles have already been added.

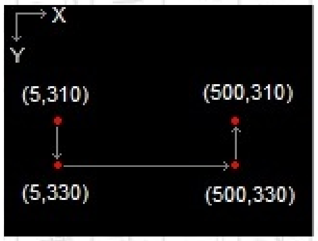
AI Add Enemy 1,0
AI Set Entity Speed 1,70
AI Set Entity Position 1,entityX, entityY
AI Set Entity Idle Position 1,entityX, entityY

This section creates our single enemy, in this case the entity ID can be any value since we are not linking it with an object and we use this value to refer to the entity from now on. The speed, in this case, is set in pixels per second since that is what we will be using as our units and the AI system will make sure that any movement happens at a speed that is consistent with 70 pixels per second no matter how often AI Update is called. We then set our entity to a position on the screen using a predefined X and Y position, we must also set the position we want the entity to return to when it becomes idle (i.e. Bored) since our entity defaults to automatic mode.

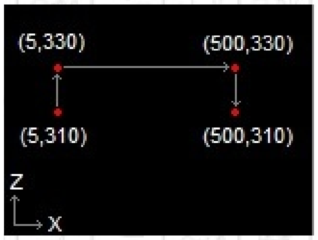
```
AI Start New Obstacle
AI Add Obstacle Vertex 5,310
AI Add Obstacle Vertex 5,330
AI Add Obstacle Vertex 500,330
AI Add Obstacle Vertex 500,310
AI End New Obstacle 0,1

AI Start New Obstacle
AI Add Obstacle Vertex 140,180
AI Add Obstacle Vertex 140,200
AI Add Obstacle Vertex 635,200
AI Add Obstacle Vertex 635,180
AI End New Obstacle 0,1

rem Boundary
AI Start New Obstacle
AI Add Obstacle Vertex 0,0
AI Add Obstacle Vertex 640,0
AI Add Obstacle Vertex 640,480
AI Add Obstacle Vertex 0,480
AI End New Obstacle 0,1
```

This section (in the MakeLevel function) creates all our obstacles, since we cannot use AI Add Static Obstacle when not using DarkBasic objects. The first two create the two long walls in the middle of the demo, in a clockwise direction for obstacles, and the third creates a boundary which surrounds the visible area, in an anti-clockwise direction, to prevent the entity leaving the screen. Remember the AI system is still using an X, Z system so the co-ordinates should be specified in clockwise as if looking down on to the X, Z axes even though we are using a 2D co-ordinate system.

For example, the first obstacles uses the corners (5,310) (5,330) (500,330) (500,310) in that order. If we were to draw that on the screen taking the co-ordinates as X,Y values (where 0,0 is in the top left corner) we would be drawing in an anti-clockwise direction:



But from the AI's point of view it is being defined in X,Z co-ordinates (where 0,0 is in the centre of the screen) in which case we are drawing in a clockwise direction:



It is the AI's point of view which is important, and which ultimately defines whether the obstacle is a boundary or localized obstacle so always order your points as if you are drawing in the X,Z system no matter how it may appear in your co-ordinate system.

The third obstacle in the list is the boundary which prevents the entity leaving the screen, so is specified in an anti-clockwise direction (in X,Z co-ordinates) effectively making everything 'behind' it one big obstacle.

At this point the AI setup is complete and we enter our main loop where we call AI Update to start updating our system. These commands are included in our main loop.

entityX = int(AI Get Entity X(1))

```
entityY = int( AI Get Entity Z(1) )
angle# = AI Get Entity Angle Y(1)

if mouseclick()=1 then AI Entity Go To Position 1,mousex(),mousey()

AI Update
```

The first three commands get the updated position and angle of the entity so that it can be drawn to the screen, the entity position is updated after every call to AI Update and since we are not using DarkBasic objects we need to position the entity ourselves. The next command sets a new destination for the entity when the mouse is clicked, the entity will then begin moving towards the destination, wait a while, then move back to its idle position.

DIRECT INTEGRATION DEMO

This demo shows a very simple AI program to show how DarkAI can take direct control of DarkBasic objects, and therefore serves as a good starting place to learn the main structure of an AI controlled program.

```
sync on : sync rate 0
AI Start
```

The AI Start command creates the AI system and must always be called before other AI commands.

```
make object cube 1,10
AI Add Enemy 1
AI Entity Go To Position 1,0,30
```

This creates our single entity as a DarkBasic cube and adds it to the AI system as an enemy. In this case it does not matter which team we add it to since there will be no other entities added. We then give the entity something to do by telling it to move 30 units forward, away from the camera.

```
do

AI Update
sync

loop
```

This is the main loop and simply updates the AI system and draws everything to the screen. The AI Update command moves our entity to his destination, then it will wait a while and move back to its idle position (the position it was at when added to the system). The cube is automatically move for us because by default an entity is linked to the object with the same number, in this case the cube, and will move it in the X and Z directions as well as rotate on the Y axis. No other change will be made to the object.

PATH FINDING DEMO

This demo shows a single entity working its way through a cluster of obstacles to a user defined point by calculating the shortest available path around the obstacles to its destination. The demo incorporates obstacles created with both automatic calculation, using an existing object to create a convex shape, and manual creation, to create the boundary around the outside. The smaller obstacles are randomly positioned each the demo is run to create a different scenario every time.

This document will detail the AI commands used to create and control the demo in the context they are used. For general usage of the commands you can find them in the command list section of the help file.
The demo begins with:

AI Start

AI Set Radius 2.5

This creates the AI system and sets the radius for all entities to 2.5 and all obstacles added will use this new value. It is advised that the Set Radius command be called early in the setup since using it once several obstacles have been added means the system has to spend time re-adjusting those obstacles.

make object box 111,80,10,2

```
position object 111,-10,5,-10

make object box 112,80,10,2
position object 112,10,5,10

rem add them to the AI system
AI Add Static Obstacle 111
AI Add Static Obstacle 112
```

We then create the two long horizontal walls you see in the screenshot, position them and add them to the AI system. You must position your obstacles before you add them since the Add Static command will use the object's current position for its internal use, and this cannot be changed once added. Using default values these obstacles are created as full height obstacles and added to container 0 (container 0 is created by default when the AI system is started).

```
for i=113 to 120

make object box i,rnd(10)+5,5,rnd(10)+5
position object i,rnd(100)-50,2.5,rnd(100)-50
yrotate object i,rnd(360)

AI Add Static Obstacle i

next i
```

This section creates the smaller randomly placed obstacles which are also added to the AI system. The obstacles also have a Y rotation which will be taken into account, again this must be done before calling Add Static and only the Y angle will be read.

```
AI Start New Obstacle 10
AI Add Obstacle Vertex -50,-50
AI Add Obstacle Vertex 50,-50
AI Add Obstacle Vertex 50,50
AI Add Obstacle Vertex -50,50
AI End New Obstacle 0,1
```

Next the boundary is created that prevents the entity moving beyond the edges of the floor. It is specified in an anti-clockwise direction (assuming X,Z co-ordinates) to enclose the area and make everything 'behind' it as one big obstacle that the entity will avoid. In this case we have given the obstacle an id of 10, which would only be used to remove it later and does not link it to anything with that number. More than one obstacle can have the same id in which case they would both be removed together. It has no other effect on an obstacle. We then finish the new obstacle and add it to container 0, as a full height object.

```
AI Complete Obstacles
```

We call this command to tell the AI system we have finished adding our static obstacles so that it can process them and create waypoint and visibility data all at once instead of doing it every time we add an obstacle. This can be a slow process when lots of obstacles are involved but it only needs to be performed once.

```
make object cone 2,5
xrotate object 2,90
fix object pivot 2
position object 2,0,2.5,40

AI Add Enemy 2
AI Set Entity Speed 2,10.0
```

This adds our single entity to the system, making sure to use the same value for the entity as the object since the default values link the entity to the object. We also set the entity speed to 10 units per second which, with the level being 100 units across, means it would take 10 seconds for the entity to get from one side to the other (100/10 = 10 seconds).

That completes the setup and we now enter the main loop where we use the mouse position to place the mouse marker object (object 1) on our floor. We then find:

```
if mouseclick()=1 then AI Entity Go To Position 2,object position x(1),object position z(1)
```

Which sets the entity destination to the mouse marker's position whenever the mouse is clicked. The destination will automatically be moved outside any obstacles where possible, you can see this by watching the path's final point which represents the destination. There may be a situation when the entity and its destination are completely blocked by obstacles in which case the entity will not attempt to move towards it. There are also 4 debug controls defined like this:

```
if keystate(2)=1 and ptimer<timer()

ptimer = timer()+300
pMode = 1-pMode

if ( pMode=0 ) then AI Debug Hide Paths else AI Debug Show Paths 2.5

endif
```

This one registers when the number 1 key is pressed and toggles the display of the current entity path. The timer variable prevents the display toggling too often and the mode variable switches between showing and hiding the path. This is the same format for the other 3 debug controls.

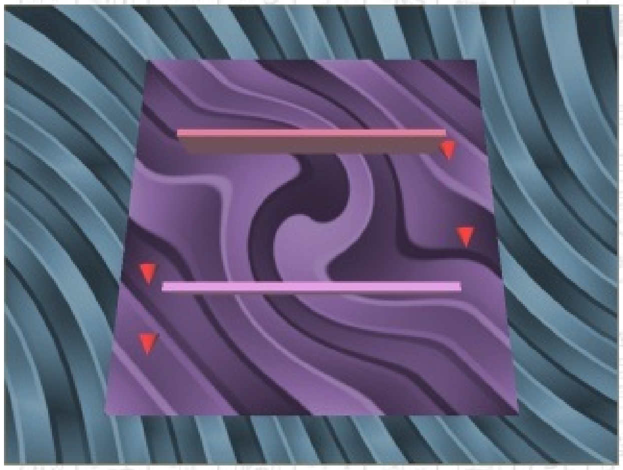
```
print "Entity State: ";AI Get Entity State$(2)
```

The Get Entity State command is used during automatic mode to get the current state that is controlling the entity. In automatic mode an entity switches between states based on what it thinks it should be doing given it current surroundings and recent events. A state can be as simple as keeping an entity en route to a destination such as in the 'Go To Destination' state, or a sequence of actions to perform an action as in the 'Search Area' state.

```
AI Update
sync
```

And finally we update the system and draw it.

PATROLLING DEMO



This demo shows the use of paths to create a patrol route for one or more entities, in this case two paths are created and two entities are assigned to each path. The two paths are created with the same points but in opposite directions, making entities patrol both clockwise and anti-clockwise around the level. This also demonstrates entities avoiding each other when they meet and how they cope with overcrowded destinations.

This document will describe the main AI commands used to create and control this demo in the context they are used. This demo starts very similar to the Path Finding demo so obstacle creation will not be covered here. It is advised you familiarise yourself with the path finding demo first to see how obstacles are setup.

```
AI Make Path 1
AI Path Add Point 1,40,40
AI Path Add Point 1,40,-40
AI Path Add Point 1,-40,-40
AI Path Add Point 1,-40,40

AI Make Path 2
AI Path Add Point 2,-40,40
AI Path Add Point 2,-40,-40
AI Path Add Point 2,40,-40
AI Path Add Point 2,40,40
```

After the obstacle setup we come to creating the two paths we want to use. The first is the clockwise path (path 1), starting in the top right hand corner, and the second (path 2) is the anti-clockwise path starting in the top left hand corner. We will refer to these later when we assign them to the entities.

```
for i = 2 to 5

make object cone i,5
xrotate object i,90
fix object pivot i
position object i,i*5 – 30,2.5,40

AI Add Enemy i,1
AI Set Entity Speed i,10.0

next i

AI Entity Assign Patrol Path 2,2
AI Entity Assign Patrol Path 3,2
AI Entity Assign Patrol Path 4,1
AI Entity Assign Patrol Path 5,1
```

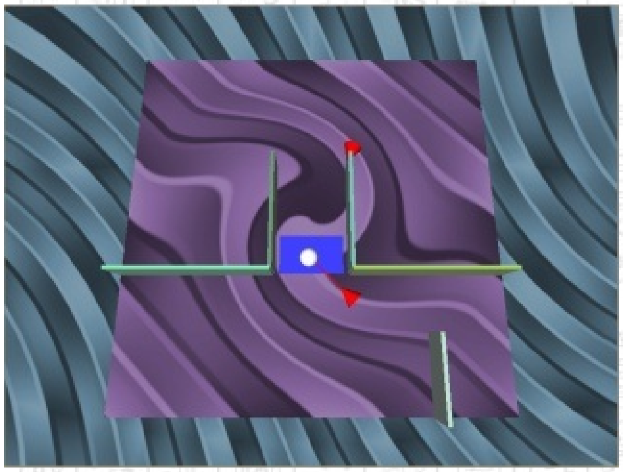
This creates 4 entities (id's 2-5 inclusive) and sets their speed. The second parameter when adding the entities is to set link the entity to its object, this would be set by default anyway but shows the usage of the second parameter. We then get to assign the entities their patrol paths, two to the clockwise path, two to the anti-clockwise path and immediately changes the behaviour of the entity to patrol whenever it would normally go into an idle state. This allows it to perform all the normal actions of an entity such as attacking and moving but be able to return to the patrol when it has nothing else to do. Any changes to the path, for example adding extra points, will take an immediate effect to any entity using it as a path. To remove the patrol state from an entity and return its idle state assign the entity a path with no points, or delete the path that is assigned to the entity.

```
AI Update
```

```
sync
```

We then call these two commands in the main loop and the entity movement is automatically handled for us. Entities will start at the beginning of their patrol paths and patrol forever, starting at the beginning again when they reach the end.

ZONES DEMO



This demo shows the zone feature and how it can be assigned to entities to trigger a response, and also has a user controlled object that acts as a player in the AI system. It creates two entities and positions them out of sight of the player, and a zone (blue) which is assigned to both entities. When the player moves over the zone the entities will be notified and move to attack the player, even though they have not seen the player themselves. If the player leaves the zone the entities will still check out the area the player entered but unless they can see the player they will return to their idle positions.

This demo also shows how you can create a basic visual attack, like a laser, by drawing a line between the entity and its target. This demo will not cover the setup of obstacles, you should review the path finding demo for details on obstacle setup. The only point to note on obstacles with this demo is the use of convex (in this case square) obstacles to create more complex walls, as long as the walls are no more than than the entity radius apart they will overlap and create a solid wall. This document will cover the AI commands used to create and control the demo in the context they are used.

dim shootTimer(3) as float

This line creates an array that we will use to store a timer for each entity to prevent entities firing too quickly. The AI system will return the 'ready to fire' signal continuously whilst an entity can see a valid target so we must handle delayed firing in DarkBasic to prevent continuous fire. Or we could not if continuous fire is what we wanted, but for this demo we will handle delayed firing.

```
for i = 2 to 3
make object i,1,0
position object i,(i-2)*40 – 20,2.5,15
color object i,rgb(255,0,0)

make object i+1000,2,1
set object light i+1000,0
hide object i+1000

AI Add Enemy i
AI Set Entity Speed i,10.0
AI Set Entity View Arc i,90,170
AI Set Entity View Range i,50
AI Set Entity Can Strafe i,0
next i
```

This loop creates two entities along with two attack objects (i+1000) that we will use later to represent attacks, and sets some entity parameters for the AI system. The Set View Arc command defines the angle at which the entities can see with in an inner (90) and outer (170) angle, in this case the inner angle does not matter since automatic mode does not make use of it, it would only be used by using the Can See command. The outer angle of 170 defines just short of a forward half circle which allows the entity to effectively spot targets out of the corner of its eye but not behind it. An angle of 90 defines a forward cone which means a target has to be mostly in front of the entity to be spotted. An angle of 360, which is the default, means the entity can see all around it. The Set View Range defines the distance the entity can see in any direction that is within its outer view angle, once both the angle and distance restrictions are checked an entity can determine if is can see a target. The Can Strafe command is used for finer control over the behaviour of the entity, in this case the entity is prevented from moving sideways relative to its target, which can normally be used to avoid fire. Since the player cannot shoot in this demo there is no reason to strafe.

```
AI Add Zone 1,-10,-15,10,-5
make object box 11,17,0.1,10
position object 11,0,0.05,-10
color object 11,rgb(0,0,255)

AI Entity Assign Zone 2,1
AI Entity Assign Zone 3,1
```

This creates the zone and makes the blue object to show it on screen, which is not required for the zone to function it is just used to display the extent of the zone. The zone is specified using the minimum and maximum corners relative to the X,Z co-ordinate system, (-10,-15) is the bottom left corner of the zone and (10,-5) is the top right corner of the zone. The first two values must always be lower than the last two so (10,-15) (-10,-5) is incorrect. Zones can only be rectangular, and cannot be rotated. We then assign our two entities to this zone so that they will be notified when the player enters it.

```
make object sphere 1,5
position object 1,20,2.5,-40
AI Add Player 1
```

Next we create a sphere to represent the player and add it to the AI system. This is all we need to do for the player since the AI system will now read the player object's position automatically every frame and update its internal values since the default values link the player to the object.

After some debugging controls and display output we come to calling AI Update and displaying any attacks that are currently occurring.

```
for i = 2 to 3
if AI Entity Exist(i)=1
if AI Get Entity Can Fire(i) and shootTimer(i)<=0
tx# = AI Get Entity Target X(i)
tz# = AI Get Entity Target Z(i)
x# = object position x(i)
z# = object position z(i)
```

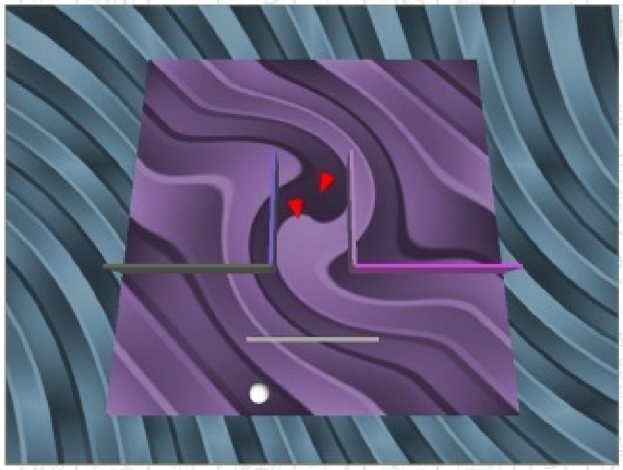
```
dx# = ( x# + tx# ) / 2.0
dz# = ( z# + tz# ) / 2.0
dist# = sqrt ( (tx#-x#)*(tx#-x#) + (tz#-z#)*(tz#-z#) )
ang# = acos ( (tz#-z#) / dist# )
if ( (tx#-x#) < 0 ) then ang# = 360 – ang#
position object i+1000,dx#,2.5,dz#
yrotate object i+1000,ang#+0.1
scale object i+1000,100,100,100*dist#
show object i+1000

shootTimer(i)=100
endif
endif

if shootTimer(i)>0 then shootTimer(i) = shootTimer(i)-(speed#*3)
next i
```

The loop cycles through both entities, first checking that they both exist within the AI system and then checking if they can see a valid target by calling Can Fire. If this returns 1 we check the entity timer to make sure they have not already fired recently, if so we let them wait a bit longer, if not then we display a new attack and reset the timer. This is what separates the continuous attack from a delayed attack.

To position the attack object for this entity we take the co-ordinates of the target the entity is firing at and the current position of the entity and average them to place the attack object between the two (stored in dx# and dz#). Next we calculate the distance between the entity and its target to be able to scale the attack object the correct length to reach between the two. Finally we calculate the Y angle between the two to rotate the attack object correctly and apply all the values to the object (i+1000). Then the entity timer is reset to 100 to prevent the entity firing again until it has reached zero.

The entity timer is decreased every frame by a value that is proportional to the frame rate, stored in speed#, to make sure the entities fire at roughly the same rate no matter the frame rate.

SOUNDS DEMO



This demo shows the use of sounds to attract the attention of nearby entities. It allows a player object to make a sound at it current location causing entities that hear it to move to investigate, resulting in an attack if the player is found at the location of the sound. Entities that attack also cause a sound in this demo which attracts further attention from entities which hear it. A conveniently placed wall allows the player to make a sound above it and run and hide from view that, as long as the player remains hidden, will cause the entities to investigate, find nothing, and return to their idle positions.

This demo is very similar to the Zone demo with a simple visual attack and obstacle layout. The obstacle setup will not be covered in this demo, you should familiarise yourself with the Path Finding demo for details on obstacle setup. The attack method will be covered in this demo.

dim shootTimer(3) as float

This line creates an array that we will use to store a timer for each entity to prevent entities firing too quickly. The AI system will return the 'ready to fire' signal continuously whilst an entity can see a valid target so we must handle delayed firing in DarkBasic to prevent continuous fire. Or we could not if continuous fire is what we wanted, but for this demo we will handle delayed firing.

```
for i = 2 to 3

    make object i,1,0
    position object i,(i-2)*40 – 20,2.5,15
    color object i,rgb(255,0,0)

    make object i+1000,2,1
    set object light i+1000,0
    hide object i+1000

    AI Add Enemy i
    AI Set Entity Speed i,10.0
    AI Set Entity View Arc i,90,170
    AI Set Entity View Range i,80
    AI Set Entity Hearing Range i,80
    AI Set Entity Can Strafe i,0

next i
```

This loop creates two entities along with two attack objects (i+1000) that we will use later to represent attacks, and sets some entity parameters for the AI system. The Set View Arc command defines the angle at which the entities can see with in an inner (90) and outer (170) angle, in this case the inner angle does not matter since automatic mode does not make use of it, it would only be used by using the Can See command. The outer angle of 170 defines just short of a forward half circle which allows the entity to effectively spot targets out of the corner of its eye but not behind it. An angle of 90 defines a forward cone which means a target has to be mostly in front of the entity to be spotted. An angle of 360, which is the default, means the entity can see all around it. The Set View Range defines the distance the entity can see in any direction that is within its outer view angle, once both the angle and distance restrictions are checked an entity can determine if is can see a target. The Hearing Range defines the radius within which the entity can hear sounds from its current location, this also applies to sounds behind walls which have no effect in blocking the detection of sounds. The Can Strafe command is used for finer control over the behaviour of the entity, in this case the entity is prevented from moving sideways relative to its target, which can normally be used to avoid fire. Since the player cannot shoot in this demo there is no reason to strafe.

```
make object sphere 1,5
position object 1,20,2.5,-40
AI Add Player 1
```

Next we create a sphere to represent the player and add it to the AI system. This is all we need to do for the player since the AI system will now read the player object's position automatically every frame and update its internal values since the default values link the player to the object.

In the main loop are the controls for moving the player object, which is automatically transferred to the AI system in when AI Update is called, along with this line:

```
if spacekey()=1 then AI Create Sound object position x(1),object position z(1),0,1
```

Which creates a sound at the player's current position. The sound is created as type 0, which is the lowest valid type and signals a low priority sound, e.g. foot steps. The entity will prefer to investigate higher priority sounds if more than one type can be heard at the same time, but since this is a quiet environment with no other sounds present the entity will investigate this low value sound. The radius is set to 1 which does not significantly increase the range of the sound, this radius is added to the entity's hearing range and if the two combined are greater than the distance between the two then it is heard. The radius provides a way of allowing larger sounds to be heard by entities that would normally be too far away to hear it.

```
for i = 2 to 3

    if AI Entity Exist(i)=1

        if AI Get Entity Can Fire(i) and shootTimer(i)<=0

            tx# = AI Get Entity Target X(i)
            tz# = AI Get Entity Target Z(i)
```

```
                x# = object position x(i)
                z# = object position z(i)
                dx# = ( x# + tx# ) / 2.0
                dz# = ( z# + tz# ) / 2.0
                dist# = sqrt ( (tx#-x#)*(tx#-x#) + (tz#-z#)*(tz#-z#) )
                ang# = acos ( (tz#-z#) / dist# )
                if ( (tx#-x#) < 0 ) then ang# = 360 – ang#
                position object i+1000,dx#,2.5,dz#
                yrotate object i+1000,ang#+0.1
                scale object i+1000,100,100,100*dist#
                show object i+1000

                shootTimer(i)=100

                AI Create Sound x#,z#,1,1

          endif

     endif

     if shootTimer(i)>0 then shootTimer(i) = shootTimer(i)-(speed#*3)

next i
```

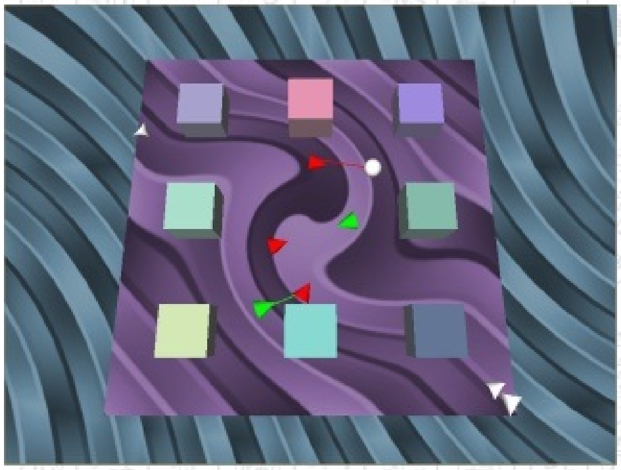
This loop cycles through both entities, first checking that they both exist within the AI system and then checking if they can see a valid target by calling Can Fire. If this returns 1 we check the entity timer to make sure they have not already fired recently, if so we let them wait a bit longer, if not then we display a new attack and reset the timer. This is what separates the continuous attack from a delayed attack.

To position the attack object for this entity we take the co-ordinates of the target the entity is firing at and the current position of the entity and average them to place the attack object between the two (stored in dx# and dz#). Next we calculate the distance between the entity and its target to be able to scale the attack object the correct length to reach between the two. Finally we calculate the Y angle between the two to rotate the attack object correctly and apply all the values to the object (i+1000). Then the entity timer is reset to 100 to prevent the entity firing again until it has reached zero.

The difference here between this and the zone demo is we also create a new sound, of type 1, whenever an entity attacks drawing more attention to the scene of the combat.

The entity timer is decreased every frame by a value that is proportional to the frame rate, stored in speed#, to make sure the entities fire at roughly the same rate no matter the frame rate.

TEAMS DEMO



This demo combines several features with the built in teams to create a simple fight scene. It includes obstacle avoidance from the Path Finding demo, patrol paths from the Patrolling demo, the visual attack and sounds from the Sound demo and introduces friendly and neutral entities. Friendly entities support the player by attacking nearby enemies and following the player when ordered to. Neutral entities do not engage in combat and by default run away from sounds, their exact behaviour can be set use the Set Stance command or controlled manually as with other entities, but they will never pick targets and always return Can Fire as 0. Neutral entities will also not be shot at by other entities.

This document will not cover obstacle setup or patrolling since these are covered in detail in the Path Finding and Patrolling demos respectively. The visual attack is also the same as that found in the Zone or Sound demos the only difference being this demo deals with more entities. It does not matter which team the entity is on or who they are firing at since this method of visualising the attack is handled the same for all entities and targets.

First we will cover the creation and addition of the enemy entities.

```
for i = 2 to 4

    make object i,1,0
    position object i,i*5 – 30,2.5,40
    color object i,rgb(255,0,0)

    make object i+1000,2,1
    set object light i+1000,0
    hide object i+1000

    AI Add Enemy i
    AI Set Entity Speed i,10.0
    AI Set Entity Aggressive i
    AI Set Entity View Arc i,90,180
    AI Set Entity View Range i,60
    AI Set Entity Hearing Range i,100

next i

AI Entity Assign Patrol Path 2,1
AI Entity Assign Patrol Path 3,1
AI Entity Assign Patrol Path 4,1
```

The entity objects are created from a mesh and positioned where we want them to start along the top of the level. We create the attack object, one per entity, and then set the initial entity parameters. We use the Set Aggressive command to make the enemies approach and follow their targets so that they will continue to attack until either, one of them dies (not featured in this demo), a closer target is spotted, or it loses sight of all targets and cannot find them. We give the enemies a generous hearing range so they can hear and respond to any entity attacking on the level. Entities are also given a view angle of 180 so they cannot see behind themselves, and a view range of 60 so they cannot immediately spot the player and its allies on the other side of the level. The entities are then all assigned the same patrol route, to move back and forth across the top of the level.

Next we create and add the neutral entities.

```
for i = 11 to 13

    make object i,1,0
    position object i,rnd(50)-25,2.5,rnd(50)-25
    AI Add Neutral i
    AI Set Entity Speed i,10.0
    AI Set Entity View Arc i,90,180
    AI Set Entity Hearing Range i,100

next i
```

Since neutral entities cannot attack we do not give them an attack object, and we position them randomly around the level. Neutral entities will run away by default, and since we don't want to change this behaviour we will just set the range at which the entity will be able to hear and see. You could also use the Can Roam command here if you wanted neutral players to randomly move about instead of remaining fixed until an event occurs.

Finally we create and add the friendly entities.

```
for i = 21 to 22

    make object i,1,0
    position object i,i*5 – 120,2.5,-40
    color object i,rgb(0,255,0)

    make object i+1000,2,2
    set object light i+1000,0
    hide object i+1000

    AI Add Friendly i
    AI Set Entity Speed i,10.0
    AI Set Entity View Arc i,90,180
    AI Set Entity View Range i,60
    AI Set Entity Hearing Range i,80

next i
```

There is little difference here to the enemy setup, except instead of calling Add Enemy we use Add Friendly to specify the team. It does not matter in which order the different teams are created, or which id's they are given as all entities are processed the same way. Friendly entities default to a cautious stance which means they will not approach and follow targets, if you wish this can be changed by setting the entity stance.

```
make object sphere 1,5
position object 1,20,2.5,-40
AI Add Player 1
```

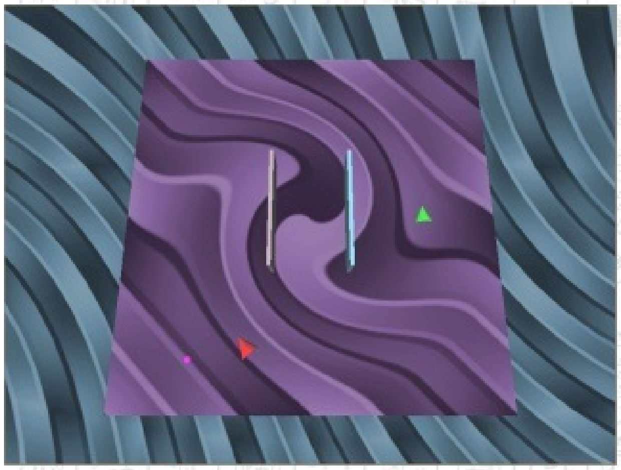
This creates the player object and adds it to the AI system. This is all we need to do for the player since the AI system will now read the player object's position automatically every frame and update its internal values since the default values link the player to the object.

Once the setup is complete the AI system now has enough information to move and control entities based on their team and current surroundings to produce a believable responses completely automatically by calling AI Update. The main loop provides a 'follow player' command for the friendly team to provide some control over the movement of allies.

```
if keystate(33)=1 and ftimer<timer()

    ftimer = timer()+300
    AI Team Follow Player 20

endif

if keystate(31)=1 and stimer<timer()

    stimer = timer()+300
    AI Team Separate

endif
```

The Follow Player command takes a distance as its parameter, specifying the maximum distance friendly entities should remain from the player. When the distance between the player and the entity exceeds this value the entity will move towards the player, otherwise it will be content with its current location. The Separate command tells all friendly entities to stop following the player and return to worrying about themselves.

COMMANDS DEMO



This demo makes use of the entity's manual mode to demonstrate some of the commands used by the automatic mode to control entities, allowing you to create custom behaviours that are not covered by the automatic mode's stance mode and command set. This involves setting the entity destination and look at point depending on the information an entity has about its surroundings, such as recently seen targets or heard sounds.

The demo itself allows you to position a marker object (magenta) and apply various commands to an enemy entity that use this marker for different purposes, like looking or moving towards it. A stationary friendly entity is provided to serve as a target when the enemy gets within sight, including a couple of obstacles to show path finding still working in manual mode and blocking line of sight. This demo does not include any form of attack as it focuses on the manual mode commands.

The first loop sets up the two entities.

```
for i = 2 to 3

    make object i,1,0
    position object i,(i*2 - 5) * 30,2.5,0

    if ( i=2 )

        AI Add Enemy i
        color object i,rgb(255,0,0)

    else

        AI Add Friendly i
        color object i,rgb(0,255,0)

    endif

    AI Set Entity Speed i,10.0
    AI Set Entity View Arc i,90,170
    AI Set Entity View Range i,50
    AI Set Entity Control i,0

next i
```

Entity 2 becomes the enemy and 3 becomes the friendly, both have the same view angle and range set and both use the Set Control command to set them to manual mode. The friendly entity is set to manual mode to prevent it moving on its own which helps to keep the scene a little cleaner.

The main loop sets the marker object under the mouse when clicked and provides 10 commands that can be given by the user.

AI Entity Go To Position 2, x#, z#

This command tells the entity to move to the marker's position avoiding any obstacles along the way. If the marker is inside an obstacle the entity will move to the closest point outside the obstacle. If the destination is completely blocked by obstacles (the destination is valid but unreachable) the entity will not move. This is the standard method of moving entities around.

AI Entity Stop 2

This command stops the entity moving by setting its destination to its current location no matter what it is currently doing.

AI Entity Look At Position 2, x#, z#

This command tells the entity to look at the marker's position and sets the entity as having a look at point. An entity with a look at point will always look at it whilst it can see it and is not blocked by obstacles (not including half height obstacles), even when moving away from it. If the entity cannot see its look at point it looks in its direction of travel until the point becomes visible again; being visible does not require the point to be in the entity's view angle just that the entity has a clear line of sight to the point.

AI Entity Look Around 2, 90, 180

This command picks a random angle between 90 and 180 and turns the entity either left or right (random choice) by that angle by setting the look at point. The entity will continue to look at this point when moving.

AI Entity Random Move 2, 10, 20

This command picks a random direction from all possible directions, and a random distance between 10 and 20 for the entity to move. The entity's destination is then set to the point at the end of this direction and distance.

AI Entity Move Close 2, x#, z#, 10

This command moves the entity to within 10 units of the marker's position. If the entity is already within 10 units of the marker it picks a new point within 10 units of the marker to move to.

tx# = AI Get Entity Target X(2)
tz# = AI Get Entity Target Z(2)
AI Entity Look At Position 2, tx#, tz#

This set of commands retrieves the position of the entity's current target and sets the entity to look at it. This point will need to be updated each frame if you want the entity to continue looking at its target. The entity may have more than one target in its list but the first target, and the one for which data is returned, is always the closest visible target in its list.

AI Set Entity Position 2, x#, z#
AI Entity Stop 2

This sets the entity's position directly creating a teleport effect where the entity will jump from its current position to the new position. Since this does not also set the entity's destination Stop should be called to prevent the entity moving back to its original location. An entity's destination is always in effect and an entity doesn't move only because its destination is under its feet. Therefore, if the entity's position is changed its destination should be set also for the entity to remain stationary.

AI Set Entity No Look At Point 2

This removes any look at point the entity currently has and returns it to always looking in its direction of travel. When an entity stops moving it will continue looking in the last direction it was moving.

AI Entity Strafe Target 2

This command requires the entity to have a target in its list, if so it picks a sideways direction relative to the direction of the target and moves to a destination randomly placed at the end of this direction. The direction is chosen internally to be either left or right.

Finally the demo displays some information about some entity parameters that can help in deciding what commands to give the entity.

print "Enemy State: ";AI Get Entity State$(2)
print "Num Targets: ";AI Get Entity Count Targets(2)
print "Moving: ";AI Get Entity Is Moving(2)
print "Turning: ";AI Get Entity IS Turning(2)
print "Can Fire: ";AI Get Entity Can Fire(2)

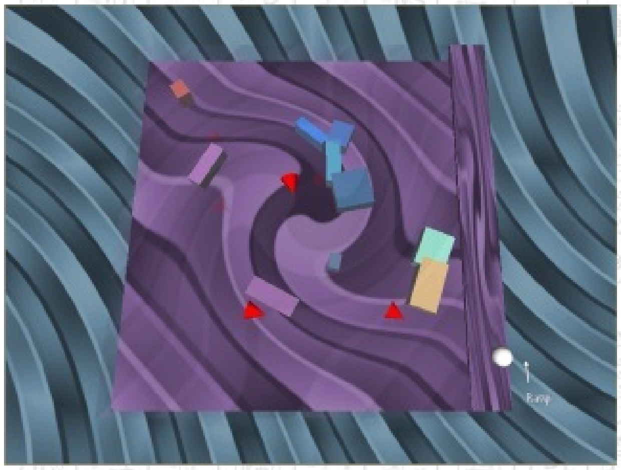
-Get State returns a description of the current entity state and is most useful in viewing automatic behaviour, its function here is to display the state as being manual and not under automatic control.
-Count Targets returns the number of targets currently in the entity's target list, but visible and recently seen. Targets will slowly be removed

from the list when they are not visible for extended periods of time.
-Get Is Moving returns if the entity is currently attempting to move to a destination, whether it is succeeding or not. An unreachable destination will produce a moving result of true even though the entity may not actually be moving.
-Get Is Turning depends if the entity has a look at point, if so it will calculate if it is looking at the point and return false if it is already within 0.1 degrees, if it does not have a look at point it returns true when moving and false when not moving.
-Get Can Fire returns true when the entity has at least one target, at least one target is within its view angle and range and the target is within its fire arc angle.

AI Update

This moves the entity to its destination and turns it to its look at point in response to any commands that have change these values, and updates the entity's target list by adding any new and removing any old targets.

CONTAINER DEMO



This demo shows the use of containers to represent two floors, each occupying the same space but completely separated from each other inside the AI system. A player controlled object is created which is free to move between the containers by means of a ramp and can only be seen by the occupants of the container it is currently in. Obstacles are also created for a ground floor, which prevent the movement of ground floor entities but do not hinder the upper floor entities. Entities will not display any attacks in this demo but will move towards targets as if they were going to attack. This is to keep the focus on containers and using them to separate entities and floors.

The demo begins with the following setup code.

```
AI Start
AI Set Radius 2.5
AI Add Container 1
```

This starts the AI system and sets the radius as with most demos, and also creates a new container with id 1. The ground floor is represented using the default container 0, which is created automatically in AI Start, and the new container 1 is added to represent the upper floor. The container can be given any positive id, but it must not already exist. Obstacles are added as in the Path Finding demo except for the one boundary on the upper floor.

```
AI End New Obstacle 1,1
```

Which specifies a container number of 1 for the upper floor. By default obstacles are added to container 0 so we only need to change the container number parameter when we want to assign things to other containers.

```
for i=2 to 4

      make object i,1,0
      position object i,rnd(80)-40,2.5,rnd(80)-40
      color object i,rgb(255,0,0)
      AI Add Enemy i,1,0

next i

for i=10 to 12

      make object i,1,0
      position object i,rnd(80)-40,9,rnd(80)-40
      color object i,rgb(255,0,0)
      AI Add Enemy i,1,1

next i
```

This creates two sets of entities, the first for the ground floor and container 0, the second for the upper floor and container 1. The entity parameters are left as default since this demo focuses on containers and the separation of the floors. The default values will allow the entities to see all around themselves at a reasonable distance, in this case about half the level.

In the main loop we need to detect when the user moves the player over the ramp in order to control both its height and which container it is currently in.

```
if object position x(1)>40
```

```
    position object 1,object position x(1),(object position z(1)+50)*
    (6.5/100.0)+2.5,object position z(1)
    if object position y(1)>5.75 then playerLevel = 1 else playerLevel = 0

else

    position object 1,object position x(1),playerLevel*6.5 + 2.5,object
    position z(1)

endif
```

The ramp is on the right side of the screen starting at X = 40 and ending at X = 50, and runs in the Z direction from -50 to 50. So if the player has an X co-ordinate greater than 40 it is on the ramp and we need to adjust its height relative to how far along the ramp in the Z direction it is. This is done by (*PosZ+50)*(6.5/100.0)+2.5*, which takes the Z position between -50 to 50, converts it into a value between 0 and 100, then divides by 100 to get it into the range 0.0 to 1.0. This represents a value of 0.0 when the entity is at the bottom of the ramp, and 1.0 when the entity is at the top of the ramp. Since the height difference between the floors is 6.5 we multiply this 0.0 to 1.0 value by 6.5 so that when at the top of the ramp the entity has moved up a whole floor height. Finally 2.5, the radius of the entity, is added to bring the entity up out of the floor to sit on top of it.

Alternatively this can be handled by a separate collision system that keeps the player and/or entities on the floor whilst they move about in their X and Z directions. The AI system does not mind objects being moved about in DarkBasic in the Y direction since from a top down view it makes no difference. You can also move the entity's object in the X and Z directions and the AI system will register the move but you may hinder entities from moving where they are trying to get to.

To check which floor the player is on we look at its object's Y position, if it is greater than the half-way point between the floors we set it as on floor 1, otherwise we set it as being on floor 0, we'll use this value next to set the player container. If the player is not on the ramp then its Y position is set using the playerLevel variable and multiplying it by the height difference between the floors.

AI Set Player Container playerLevel

Here we tell the AI system which container the player is in, 1 for the top floor, 0 for the ground floor. This cannot be calculated automatically since no height information is stored about the containers so the AI system would not be able to tell the difference between container 1 and container 0. You can continuously set the player container to the same value every frame as it simply sets a value in the AI system and does not do any calculation.

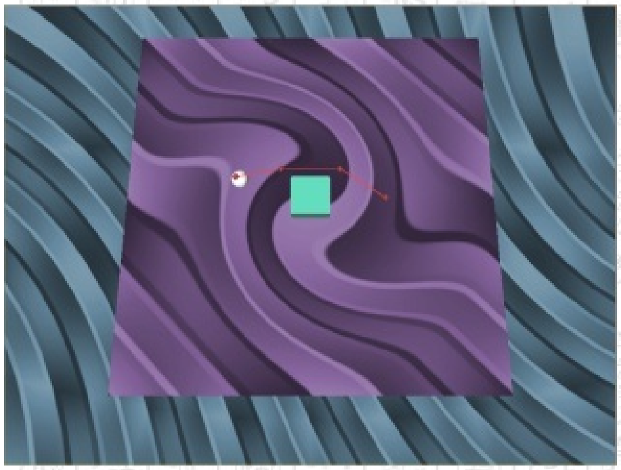
```
if showBounds=1

    if currentShowBounds<>playerLevel

        AI Debug Hide Obstacle Bounds currentShowBounds
        AI Debug Show Obstacle Bounds playerLevel,playerLevel*6.5 +
        2.5
        currentShowBounds = playerLevel

    endif

else

    AI Debug Hide Obstacle Bounds 0
    AI Debug Hide Obstacle Bounds 1

endif
```

This is an addition to the debug control that displays obstacle edges to detect when the player changes floors whilst the debug object is visible. If this happens it switches between showing the obstacles bounds for floor 0 or floor 1 and at the height of the floor they represent. This is done for clarity to only show the bounds for one floor at one time although you could show both at the same if this was desired.

AI Update

Finally the Update command handles all the details of moving and controlling entities in all containers. Entities will only register that they can see the player when the player is assigned to the same container as them. When the player leaves the container the entity will behave as if it lost sight of the player, and in some cases search for it. Although the entity cannot see it again until it re-enters the entity's container.

DEBUG DEMO



This demo shows the complete range of debugging commands available for viewing what is going on inside the AI system. These function by creating new DarkBasic objects and manipulating them within the AI system to display the desired information, therefore these commands work best when you are using a 3D world to represent the world. The information available for display includes waypoints, waypoint edges, obstacle bounds, sounds, entity paths, entity view and hearing ranges, entity-entity avoidance data, and detailed entity parameter information.

It demonstrates each command in turn by showing a simple example of the command's display then removing it for the next. As such this demo does not follow the usual pattern of setup -> main loop -> end, but instead has several mini setups with a pause to let the user move to the next.

AI Start
AI Set Radius 2.5

As usual, we start by creating the AI system and setting the radius we want to use. The Start command creates container 0 where we will be adding our objects and displaying them. We then move to the first debug command, showing waypoints.

AI Add Static Obstacle 101

```
backdrop off
cls

AI Debug Show Waypoints 0,2.5
print "Debug Commands Demo, Press Any Key To Continue"
print
print "1 - Displaying Waypoint Data"
wait key
```

This adds a single obstacle in the centre of the screen, adds it to the AI system which automatically creates four waypoints for it, then displays the waypoints for container 0 at a height of 2.5. The height is necessary because there is no Y data stored for waypoints or obstacles so you can choose a Y position for them to appear. Waypoints are shown as blue dots that represent the corners of obstacles so entities have a point of reference about how to move around an obstacle. The debug command chooses a DarkBasic object id by starting at 65535 and decreasing until a free object is found or it reaches 0, in which case the command fails and has no effect. The Show Waypoint command only calculates its object once, and so does not automatically update its display if you add/remove waypoints whilst it is displaying. You would have to call Show Waypoints again to re-calculate it. The program then waits for the user to press a key before continuing.

```
AI Debug Hide Waypoints 0
cls
AI Debug Show Waypoint Edges 0,2.5
```

Next we hide the waypoints and display the waypoint edges. These represent the visibility of waypoints from each other, an edge between two waypoints means an entity can use it as a valid section of path when moving around obstacles. Edges are shown as blue lines and again you choose the Y position you want the debug object to appear and the object number is chosen by starting at 65535. Edges are not re-calculated automatically so you need to call Show again to display changes.

```
AI Debug Hide Waypoint Edges 0
cls
```

```
AI Debug Show Obstacle Bounds 0,2.5
```

This hides the edges and displays the obstacle bounds which define the extent of all obstacles with the radius (set above) added. The difference between waypoint edges and obstacle bounds is that obstacle bounds are drawn surrounding each and every obstacle even when they overlap, whereas waypoints take the obstacles as a whole and draw around overlapping obstacles as if they were one. Waypoints edges can also link obstacles together. The debug object is chosen by starting at 65535 and is not re-calculated automatically. Call Show again to update any changes to the debug display.

```
AI Debug Hide Obstacle Bounds 0
cls
AI Debug Show Sounds 2.5
```

We now move on to displaying sounds which can attract nearby entities, they are shown as yellow dots and they are updated in real time. The DarkBasic objects to display them are chosen by starting at object id 65535 and decreasing. This requires some sounds exist in order to view the debug output.

```
repeat

    if scancode()<>0 then hold=1
    set cursor 0,0
    print "Debug Commands Demo, Press Any Key To Continue"
    print
    print "4 - Displaying Random Sounds"
    AI Update
    if rnd(screen fps()) = 0 then AI Create Sound
    rnd(80)-40,rnd(80)-40,0,0

until scancode()=0 and hold=1
```

This loop creates random sounds around the level, which will be displayed by debug objects, until the user presses a key. The yellow dot will remain for as long as the sound can be heard, about 1 second, and then be removed from system. The Update command is needed to remove and display sounds as it is a real time debugging command. The next debug command involves entity paths.

AI Add Enemy 1
AI Make Path 1
AI Path Add Point 1,20,0
AI Path Add Point 1,-20,0
AI Entity Assign Patrol Path 1,1

AI Debug Show Paths 2.5

This creates an entity and a patrol path to use in the next debug command. The patrol path deliberately crosses the obstacle from the previous setup to create a more interesting path, which is display in red. The final point in the path represents the entity's final destination, which it has generated the path to, and the intermediate points are shown as red dots which represent the waypoints the entity is using to get to its destination, if any. The points are connected by red lines which represent the order the points will be visited in. Path displays are updated in real time and you can specify a Y position for the debug object. Paths will be displayed for all entities. Another loop updates the AI system to move the entity and update the path.

AI Debug Hide Paths
AI Set Entity View Arc 1,90,170
AI Debug Show View Arcs 2.5

A view angle is assigned to the entity from the previous setup for the next debug command, which displays view angles, view range, and hearing range. The view is shown as a transparent red circle that extends to the distance of the view range and angle within which points can potentially be seen. The hearing range is shown as a yellow circle within which sounds can be heard. The calculation for the display object is only done once since view angles and ranges rarely change it is faster to keep the object from frame to frame. Any changes to the view range, angle or hearing range need a call to Show View Arcs to update. The position and rotation of the display object will be updated in real time when Update is called as this can be done quickly.

AI Debug Hide View Arcs
make object sphere 2,5
position object 2,9.5,2.5,4
AI Add Enemy 2
AI Debug Show Avoidance Angles 2.5

The next command requires another entity to display entity to entity avoidance data. This is where an entity detects nearby entities and marks their area as blocked, restricting movement in that direction. Blocked directions are shown as a small green arc extending to the full width of the marked area and update in real time. This is a slow command as it must re-calculate the avoidance object every time it changes which is every frame when it is close to one or more entities.

AI Debug Hide Avoidance Angles
AI Kill Entity 2
delete object 2
AI Set Console Output On 1

The second entity is no longer needed so is removed from the AI system and deleted, always remove an entity before deleting its object if the object is linked to it. The Set Console Output command is different in that it opens a new window to display a detailed list of the main entity parameters that is updated in real time. Closing this new window manually will result in the program crashing, you should use Set Console Output Off to close it.

AI Set Console Output Off
AI Hide Errors

Hiding errors is used to prevent the AI system exiting the program and displaying error messages when an invalid action or command is performed, such as setting a parameter for an entity that does not exist. It is useful if you want to handle errors yourself, for example adding an entity and then checking if the entity exists will let you know if the addition succeeded or failed.

AI Set Entity Speed 3,10

In this case the loop contains an invalid command since entity 3 has not been added, with errors off the command will have no effect.

Compatibility With A Physics System

The main issue with combining DarkAI with a physics system, such as DarkPhysics, is that physics will also want to control the DarkBasic objects representing your entities to stop them moving through each other. If you link the objects with the AI system with the physics system trying to move them conflicts may arise between which position the object should be moved to. The solution to this is to give control of the object over to physics as normal and to not link the object to the AI system, instead using the Set Position and Get X/Z commands to change and retrieve the entities' positions.

If the physics system can detect when an object is re-positioned and update its internal values then the two systems should work together as long as AI Update is called before updating physics. This avoids the possibility that after physics has prevented collisions the AI moves the entity into a new collision. If this is not the case then you can remove the AI system from controlling the entity and use the physics system to position the entity where the AI wants it to go.

When not linked to an object DarkAI will use internal values that represent where it thinks the entity should be when *AI Update* is called, use *AI Get Entity X* and *AI Get Entity Z* commands to get these values. You should then use methods provided by the physics system to move the DarkBasic object to this position, for example using forces, velocities or positions whichever the physics system prefers, and let the physics system move the DarkBasic object. Then update the physics system so that it positions the DarkBasic object for you and adjusts for collisions, etc, automatically. Then get the new position of the object using the normal DarkBasic commands *Object Position X* and *Object Position Z* and use the *AI Set Entity position* command to set the new internal position for DarkAI to work with. This new value will then be used by the AI system in the next *AI Update* to work out where it wants the entity to move to, it stores these new values internally and the cycle repeats. It is recommended you always update the AI before you update the physics in your loop.

You can use *AI Set Entity Collide* to tell an entity when it is blocked by something it doesn't know exists (a physics object that is not an obstacle in the AI system) which the AI would constantly try to move through and physics would keep stopping it. This tells the entity to move sideways for a bit.