Unangband Monster AI

Andrew Doull

http://roguelikedeveloper.blogspot.com/2007/10/unangband-monster-ai-part-one-history.html

Contents

Chapter 1

History and Context

Unangband monster artificial intelligence routines have evolved from the Sangband and Oangband's so-called 4GAI which is short-hand for Leon Marrick and Bahman Rabii's "4th generation Artificial Intelligence". It's a set of common functions and algorithms used across a number of Angband variants: in addition to Sangband and Oangband (and FAAngband), its been adopted separately by NPPAngband and Unangband, both versions of which have evolved in different directions.

Its called 4th generation, because it's the fourth major evolution of the monster AI routines in Angband's history. Leon Marrick has written extensively about it in the comprehensive design manual[1].

I'm going to quote the development history liberally from this manual as the original is a text document which likely won't format well in your browser. The history is interesting reading and written with Leon Marrick's usual mix of enthusiasm and iconoclasm:

> History: Standing on the shoulders of giants
>
> The history of coding monster intelligence ("monster AI") in Moria and Angband is a long one. You will not be reading the true history here; the following is one man's (i.e. LM's) opinion and viewpoint on previous AIs. If you don't disagree with me at least once in the next six paragraphs, you need to learn more about this subject!
>
> *The First Generation:*
>
> The first generation AI was mostly the work of Robert Koeneke. Each turn, a monster would attempt to cast a spell, if it had any. If it passed the spell frequency test, it would pick a spell at random. Although a primitive method, choosing spells at random had the great advantage of keeping the monster true to character. Not only was every spell guaranteed to be used, but monster designers could tweak the frequency of, say, harassment spells against attack spells by changing the number of each.
> Monsters that didn't cast spells entered the movement code. This involved choosing a direction that best approached the character, plus the two directions that flanked it on either side, for a total of five possible directions. The monster would then try these five choices in order, testing any doors or glyphs, moving as soon as it found a grid it could enter or succeeded in clearing. Again, to say that this was crude is less important than to note that, in the conditions of Moria, it worked acceptably.
> The young Angband added many spells and new features, but made few changes to this basic system. However, at or before version 2.6.2, intelligent monsters were made to cast better spells if desperate, and the first monster terror code appeared.
>
> *The Second Generation:*
>
> When Ben Harrison took over, things started to change. First came a massive code cleanup. David Reeve Sward contributed the basic monster learning system we still use today; by storing information about character resists and immunities, monsters could learn to probe for missing resists - and then exploit them cruelly! William Tanksley came up with a way for monsters to target the character, which allowed them to track their enemies down in a realistic, limited fashion. Ben Harrison introduced the famous monster flow, which stored the route distance from the player up to a range of 32 every time the player moved. Although

[1] http://www.oangband.com/dl_misc/4GAI.txt

extremely CPU-intensive, and criticized as giving monsters too much of an advantage over the player, the monster flow code made such a difference to monster intelligence as to become almost indispensable. It eventually replaced the monster tracking code. This, then, is the suite of features of the second generation of monster AIs, and the common possession of all modern Angbands.

The Third Generation:

When the Keldon Jones AI came out, it caused a sensation. Monsters stopped killing each other off with magic missiles, Zephyr Hounds starting luring the player into rooms, orcs started to surround the player, and monsters chose spells according to their tactical situation so cleverly that some of them became nearly unkillable. It took a while, but eventually almost every major variant and Standard Angband adopted this code with greater or lesser modifications. The best word on the Keldon Jones AI is that of Greg Wooledge, explaining why he included this code in his variant: "I simply couldn't stand watching the novice rangers act like Keystone Kops any longer.".

The Fourth Generation:

But the Keldon Jones AI was by no means perfect. For starters, it was extremely slow. In addition, this system made many monsters act out of character by discriminating against certain classes of spells or by making monsters suppress their most useful attack, and was difficult to fine-turn, either by the monster designer or the coder. Because of this, Standard Angband, Angband/64, Drangband, Oangband, and Zangband all made significant improvements at one point or another.

In the minds of Bahman Rabii and Leon Marrick, the work appearing in Oangband (remember, that's short for Opinion-Angband) 0.5.1 and 0.5.2 is the first of these efforts to go decisively beyond the level of the third generation, and merit the title:

"The Fourth Generation AI".

The manual is full of implementation details and notes for Angband variant writers, and is probably useful for most other roguelike developers as well.

NPPAngband has evolved the 4GAI in a number of interesting ways, with a primary focus on getting the correct route across multiple different types of terrain. Jeff Greene and Diego Gonzalez adopted parts of the Unangband terrain code, and decided to extend the existing flow code to use multiple per-terrain flows with varying movement costs based on the type of terrain and the 'native-ness' of the creature to the terrain it is planning a route through. More than enough improvements to the 4GAI were done to justify the moniker of "4.5GAI"" when looking at the NPPAngband source code[2].

Increasing the different types of terrain complicates a lot of the monster AI route finding, because it is no longer possible to guarantee that the route the player and the monster takes is the same. It also exponentially increases the number of routes to be computed based on the number of different types of terrain restrictions. For instance, the shortest route may require that a monster swim through lava, then fly over a chasm, then open a door. Up to eight flows would have to be computed in this instance, based on the combinations of lava-nativeness or lack of nativeness, and ability or lack of ability to fly and open doors. Efficiencies can of course be gained by noting which monsters on a level have these capabilities.

However, in the long run, I've decided the effort to handle these combinations is probably not worth it. Even with perfect path finding, the player may be able to freeze the water, preventing a swimming monster swimming through it. Or a door may be opened or bashed down by a monster capable of doing so, thus allowing other monsters through that may be not so capable. The dynamic flow should pick up and allow the monsters to notice the changes, but in the mean time, they may have moved down a less efficient route, taking them away from the player. And the most interesting monster is one that is walking towards the player, trying to attack, as opposed to one that is moving away.

So I've decided to stick with the existing 4GAI 'flow-by-scent' and 'flow-by-noise' approximations, even if I have yet to adopt the 4GAI noise-making routines. They are good enough approximations for the moment, and have a credible real-world explanation for how the monster finds the player. And I will do some route-finding improvements at some point. I've tentatively sketched out extending flow scent to have a separate 'scent through water' flow, that monsters may or may not be able to take advantage of. And I definitely want smarter fleeing routines, and improved

[2] http://members.cox.net/nppangband/download.htm

ability to escape hazardous terrain.

I'd also like the ability for monsters to try and 'sneak up' on the player without them being invisible - the current Unangband implementation just doesn't work very well on this account. I've developed some currently commented out code for the monster equivalent to the player running algorithm, to try to get monsters moving between rooms in an interesting fashion. But, as the denizens of rec.games.roguelike.development note: if the player can't see it, it isn't worth it.

Chapter 2

What is worth doing?

I want to expand on that idea of 'if the player can't see it, its not worth doing', which R. Dan Henry, current maintainer of Gumband[1] is a big exponent of. There's a great article on artificial intelligence in F.E.A.R. called 'Three States and a Plan: The A.I. of F.E.A.R.'[2] which anyone who is serious about artificial intelligence development for games should read. While the technical details are interesting, the key point worth making is that player's don't want realistic AI in games. They want AI that tells them what their opponents are thinking.

A player in a game will be frustrated if while they're hiding in a vent, an opponent rolls a grenade in and kills them without warning. That would be the realistic AI option, but that development effort ends up with the unintended result that the player just thinks the game is hard, not that the enemies are smart. What the player instead wants, is to hear the enemy say 'Gordan Freeman is hiding in the vents. I'm using a grenade.' That's not realistic at all. In a movie, you'd cut away to show the enemy soldiers signalling with their hands (and it'd still be exaggerated). But because the player can't see this, the AI actions have to be telegraphed to them. And audio dialog is used in a FPS engine to do this.

One of the most effective AI actions in F.E.A.R. is when the player has killed all but one of their enemies and they hear the radio message 'Man down, I need support. Calling for reinforcements.' broadcast by the last remaining survivor.

The equivalent AI action is: do nothing at all. No additional backup arrives. The designers didn't need to worry about developing this code. The player perception is that the backup is coming, so they have an incentive to hunt down the last survivor. And the designers know that there are more soldiers around the next corner, which the player will perceive as the reinforcements arriving a little to late. It may sound like a cheap trick, but it is a huge saving in development cycles and CPU time.

So what does Unangband do?

I try to find cheap answers, as opposed to smart answers. I'm not going to exhaustively catalog them here, more give you tasters of the various techniques I use. A lot of developers equate Artificial Intelligence in gaming with computer science AI, in particular, route finding or path finding. These are more the domain of operations research, which concerns itself with finding the best solution. AI should be more about finding a 'good-enough' solution: for local agents using limited information. The local agents, in the Unangband case, are monsters, and where possible, I strengthen the use of local information, rather than global solutions.

The Unangband monster structure contains the following 'local AI state' which is inherited from the 4GAI:

1. A target (x,y) which the monster is heading towards.
2. A minimum range that the monster wants to stay away from the target.
3. A best range, that the monster wants to ideally be at.
4. A set of flags which contain some AI state information: in particular, did the player attacked me the last turn.
5. A set of smart flags indicating what resistances the monster has learned about the player.

Other than adding a few flags in 4, I did not expand at all on the existing monster AI structures. In return, I've been able to add the following additional behaviours:

[1] http://roguebasin.roguelikedevelopment.org/index.php?title=Gumband
[2] http://web.media.mit.edu/ jorkin/gdc2006_orkin_jeff_fear.pdf

1. Monsters talk to each other, conveying information about the player position, resistances or lack of resistances.
2. Monsters wake up their fellows nearby when they are under attack.
3. Archers will share ammunition with each other, so that each monster in the group will usually have ammunition available.
4. Monsters will eat when they are hungry, to help them recover from injuries.
5. If a group of monsters is fighting the player and the player can be flanked, some will sneak off and take the back route to the player.
6. Added friendly monsters, which intelligently move around the player and get out of their way, and attack targets, all done without adding a single 'order a monster to do something' command.
7. When fighting another group of monsters, archers and spell casters will stay at the back while warriors will move up and form a front line.
8. Faster monsters like wolves will attack isolated enemies, and the edges of the front line, dashing in for a bite and then retreating back.

That I've been able to achieve all of that is firstly a testament to the 4GAI code. I was amazed how readily I could add friendly monsters, despite Leon's opinion to the contrary. The concept of minimum range and best range is brilliant, and most of the formation code relies on this idea. But its also shows that you don't have to write particularly intelligent or deep code if you rely on using local agents 'naturally' and stick to local information. In the rest of today's article, I'm going to discuss point 1, and in follow up articles how I achieved the rest.

There are AI routines which already exists in the Angband code that allow the monster to learn about a player resistance, and then base the spell to cast on the information available. For instance, a vampire with several different elemental bolt spells might try the strongest fire bolt first, and, learning that the player resists fire, switch to lightning, then cold, and so on as additional resistances are learned. 4GAI provides much of the improvements to the code to allow this natural progression to occur, including adding monsters that rely on this.

In Unangband, if a monster learns about a player resistance, it tells all the other monsters around it. The player sees a message similar to 'The goblin wizard casts a fire bolt. The goblin says "The player is resistant to fire"'. Other information gets conveyed the same way, such as if the player is doing cheap exploits like hack-and-back or pillar dancing[3], the detection routines which come from improvements in the NPPAngband code. Pillar dancing may be seen as cheating by the august fellows of Rogue Basin (And I think John Harris[4] would disagree), but in NPPAngband, monsters notice you doing it to them, and in Unangband, they tell everyone else around them as well, so the advantage of doing so is lessened.

The information communicated includes a state change on the listening monsters so that the talker knows not to provide redundant information if someone nearby has already mentioned it. This state change already exists in the monster structure, either being the player resistance flag, or other local information (such as am I awake?). And there's some nice gloss to requiring that the player speaks orc, to understand what those Uruk are saying, and so on.

The talkers will wake up monsters near them, which lessens the ability of super-stealthy characters to wake up and fight one monster in a room at a time, which always feels unrealistic to me in Angband. Of course, to counter this, if you manage to stun or cut a monster badly enough, it loses its ability to talk, so a backstab with a blade or blunt weapon becomes more worthwhile. And if you go around beating up the townsfolk, they run away, and tell everyone around them to run (or turn the tables on you if the listener happens to be a battle-scarred veteran), because they don't clear the particular 'I've been attacked' state flag, and pass it on.

There's nothing complicated about the code used to add monsters speaking. In fact it turns out that the problematic code that in the latest version causing monsters to randomly note you have incorrect resistances was me trying to be too clever, and simplifying the code was the correct fix. The incidental effects greatly improve the player perception of the AI. And, it makes a 'hidden' feature that has been in Angband for a long time: the fact that monsters learn about your resistances, a lot more obvious to the player.

[3] http://roguebasin.roguelikedevelopment.org/index.php?title=Cheating
[4] http://www.gamesetwatch.com/column_at_play/

Chapter 3

Emergence

I mentioned that monsters were woken up as a side-effect of the monster talking code. Consider, for a moment, an AI system that only relied on monsters talking to each other, and did not implement any path or route finding algorithms. You could do this as simply as having monsters saying one of two things:

1. The player is not here
2. The player is here

The behaviours you'd then implement in a system would be:
If the player is here, stay where we are. If the player is not here, move somewhere else. Move away from positions you can hear a monster saying that the player is not, and towards positions that you can hear a monster saying the player is.
The emergent properties of that system would be that monsters would quickly move towards the player location, without having to rely on an explicit route finding algorithm.

In Unangband, such a property emerged from another AI flag I had implemented, without me even intending to do so. The flag in question is the wonderfully named MFLAG_PUSH, which I added to fix a problem I introduced modifying the 4GAI.

In Angband, particularly powerful monsters can push past weaker ones, swapping positions. The 4GAI expands on the numbers of monsters that do this, and in Unangband, I basically allow every monster to push past another. However, this can result in a deadlock situation. Consider two monsters that are running down a corridor. If the one in front is moving before the one behind, no problems occur. However, if the one behind moves first, it'll swap positions with the first. Then the first one, which is now behind, will swap positions again. The two monsters will end up effectively running in place, swapping positions and never going anywhere.

To avoid this situation, I added the MFLAG_PUSH flag, to which was set on both monsters, when one pushes past the other. And when a monster has MFLAG_PUSH set, it becomes unpushable, until the start of its next move, at which point it is cleared. The flag is set on both monsters to prevent either getting 'free moves' by being pushed around by multiple monsters: originally I did this to prevent a monster that must swim being pushed beyond the edge of water (which requires two consecutive pushes), but it equally applied to stop a powerful monster getting multiple moves against the player.

And something interesting happened. Consider the interaction of a group of monsters moving at the approximately the same speed towards the player, one behind the other. However, the monsters are moving in a larger area, like a room. At some point, the front monster will end up with the MFLAG_PUSH flag, because he'll be slightly faster than the others and pushed his way to the front. However, because he's not twice as fast, so the monster behind will get a turn first before he can move again. The monster behind clears the push flag, and considers which squares he can move into. The immediately front one is blocked, by a monster with an MFLAG_PUSH set. However, the two immediately diagonally to either side are clear. Since all Angband variants don't increase the movement cost of diagonal moves, the monster will choose to move into one of the diagonals.

And the monsters behind him will take the same approach. What emerges, is that the group of monsters naturally spread out to a wide front taking up the available width of the open area, or their group size, whichever is smaller. And it so happens, that a wide front is the smartest group for most monsters to form.

Its important to note that this occurred accidentally, without any concept of formations or other AI tricks programmed in, and is a single bit-flag setting in the monster structure. The route finding required for this formation is

completely local: a simple 'can I move into the adjacent grid' test. And the emergent property is impressive, resulting in a complex looking front on the playing field.

If you look hard at the concept of using local information to guide the monster AI, suddenly a number of other useful strategies make themselves apparent. I've already discussed the monster talking above; but here's a few more:

1. Monsters should hang around dead bodies, but not too close.
2. Monsters should hang around injured fellows, but again, not too close.
3. Monsters should aggressively search, if they find a dead body (This is straight out of Metal Gear Solid: of course, remember to mark the body as having been found, otherwise the monster will keep going into high alert mode every time they trip over the same corpse).
4. Monsters should share resources they don't need with each other.

Point 4 is a subtle one. It's not monsters should ask for resources they need. That behaviour is too complicated, and requires all sorts of smart AI assessment. Consider the resource of ammunition. Unangband has added ammunition on top of the 4GAI, to balance monster archery. And monsters do quickly run out of ammunition, if they're given half a chance to shoot at the player. At this point, they could either do a complicated search algorithm to try to find out which of their compatriots has ammunition, or just advance on the player and try to attack in melee.

And advancing, as I've noted previously, is a lot more interesting for the player. However, the monsters directly behind the archer, if any are there, may possibly have the ammunition that the monster needs. And they're not doing anything, so they should be sharing ammunition with anything they push into. Consider it a game economy of ammunition sinks, and ammunition providers. You want the providers to share freely with the sinks. The sinks will naturally be greedy without even trying.

I've implemented (approximately) the following algorithm:

1. If the monster I walk into and I use the same ammunition, average the amount between us.
2. If I have ammunition that the monster needs, give him half my inventory, including the ammunition.
3. Otherwise, give him all of my inventory.

And the result, of a simple sharing routine, is that everyone in a group quickly ends up with approximately the same ammunition. As it gets used up, the individuals in the group ensure that overall, each has the same amount. Again, the rules are only local, not global.

I'm in the process of adding a second emergent property to the same MFLAG_PUSH flag. The MFLAG_PUSH is another way of saying 'don't walk through me'. And when you think about it, the time you don't want that the most, is when you are doing useful work. The most useful work a monster can do, is successfully attack the player. By adding an MFLAG_PUSH to the monster when they've successfully attacked the player, a number of other useful emergent properties occur.

Chapter 4

Formation

At this point, I recommend you download Unangband[1] and check out the monster AI behaviour. The easiest way to do this is as follows:

1. Start a new game. Create a character - pick a man of Bree for your race for the moment (Maia is fine if you've chosen that option). Class doesn't matter, but warriors have more hit points.
2. Go into debug mode. Command is Ctrl-a. Confirm you want to continue.
3. Jump to the Battle of Five Armies. This is Ctrl-a, j, dungeon is 26, level 0.
4. You should find yourself on a big open plan with a mix of allies around you.
5. Use either the numpad or roguelike keys or mouse to move around. Watch as your allies mow down the enemy, which will usually be orcs (o) or ogres (O).
6. Steer clear of orc priests - they're light blue o's and will target you directly with wound spells. Alternately, Ctrl-a, e, and give yourself 1000 experience or so.
7. If you have problems finding enemies, Ctrl-a, D, will detect every monster on the level. You can use the 'L'ocate or 'W'here command to scroll around the full level map.
8. Ctrl-a, j, dungeon 26, level 0 will generate up new troops for you when you run out (actually a whole new level).
9. ENTER or ? for help.

Now, depending on the mix of troops you start with you'll see one of two behaviours:

- Warriors will charge up and engage the enemy. They'll form a front when they engage, that will spread out width wise until pretty much all of them are fighting.
- Archers will charge up until they're approximately 6-8 grids away. They'll then start shooting. They'll also form a front, using the MFLAG_PUSH behaviour that I described in part three.

This is usually fun, and would be a great mini-game in its own right. Implementing allies was relatively straightforward under the 4GAI. I had already modified monster combat and spell casting so that monsters could target and damage each other – and all I added was a target selection routine for allied monsters (and monsters that had been attacked by your allies).

The target selection routine is broken down into two parts: ranged attacks, which just involves picking any target within the monster's line of sight and using a spell against it, and movement, which involves picking the closest enemy monster and moving to the grid it occupies. These are not especially complicated, although somewhat inefficient.

The differing behaviour between warriors and archers comes from the best_range computation. Archers tend to have a best_range of 6, warriors of 1. In the target selection routine for movement, if the closest target picked is closer than the archer's best range, the archer forgets about it.

This is an unusual decision, but involves a complication that I haven't elaborated on. Your allies use your current target, if they are unable to find a target of their own. So when an archer forgets about their current target, they instead by default moves towards your current position, or whichever position you have targetted. The in-game tips, which you probably skipped over, outline the options you have:

[1] http://unangband.blogspot.com/

```
=== Commanding your troops ===

Most honoured general,

I have hand-picked a select group of your finest warriors from
your homeland with which to fight alongside you in this, your
hour of need. If you find you are in a bad position or the
troops inadequately provided for when you first arrive at the
battle, you can depart, and I will endeavour to supply you with
better forces.

In general, the troops you command will follow you and fight
intelligently when you are nearby. Archers and spell casters
will stand off of targets and bombard them with spells and
arrows. However, archers and spell casters will take a guide
from your current position, and move towards that position if
they are too close to the target. This may mean they foolishly
follow you into battle. Target a grid to hold, as I outline
below, in order to avoid this.

Your troops will take also take lead from targets that you
specify, as follows:

If you specify a monster target, your troops will move towards
that monster if they are not engaged in direct combat. Archers
and spell-casters will keep a distance from the target.

If you specify a race of monsters, using the 's' key while
targetting, your troops will preferentially target that race
against all others, ignoring other targets and direct combat
if they can see one of the race you specify.

If you specify a grid target, your troops will move towards
that position and hold it if they are not engaged in direct
combat. Archers and spell-casters will also move up to the
position: you can use this to specify your archers and spell
casters stay back while you engage in melee with the enemy.

You can also specify a grid position with the 'a' key, whilst
targetting. This will cause your troops to assault the target
grid. Troops in an assault will disregard nearby combat in
favour of attacking monsters near the grid that you are
assaulting. Archers and spell casters will also seek out
targets near the grid that you are assaulting, but will
prefer to move back towards your position if they are too
close to the target in question.

Once you have specified a grid target, either to hold or
assault it, your troops will continue to prefer that
location. To cancel the orders, you must either specify
another target, or hit the Escape key whilst targetting. At
that point they will begin to follow you again.

As always,

E.
```

This enables me for the most part, to avoid having to add any specific commands for ordering a player's allies around. The assault and hold commands are there to allow the player to elabourate on what they mean by pointing at a grid (attack it vs. run for it).

Chapter 5

Complications

In part four, I discussed how the Angband targetting system is used by the allied monster AI to implicitly take commands from the player. To reiterate, when the monster AI is not able to find a nearby enemy monster, the ally will instead move towards the location that the player has targetted, with a number of variations. I deliberately built the allied AI system this way so that the player would not have to learn additional commands in order to order his troops around. I then hinted that there was one additional command that needed to be implemented. And if you've been reading comments elsewhere on the Unangband home page, you'll know what this is.

If you've played with the Unangband allied AI, you will have seen that the majority of your troops actions are dictated out of the more general monster AI routine. I adapted the min_range and best_range algorithms slightly, but was able to get approximately the same behaviour in terms of troop movements as the 4GAI already does for enemy monsters. And otherwise, the AI routines are the same. With a target found, allied archers will shoot and allied warriors will melee. Allied river orcs will hide in water, allied wolves will harass the enemy and so on. The flexible 4GAI monster system works perfectly in this regard, honouring all the monster flags that Angband implements. Hopefully, your minions will survive the first couple of fights, loot the bodies of the dead (or eat them, if they are so inclined), and with their spoils, follow you on to further battles.

And that's the problem. If the allied monster has the TAKE_ITEM flag, which means they pick up whatever items are in the grid that they move into. And they will. Archers will strip the ammunition off of the enemy and re-use it. The (very limited) monster use an object routine I've implemented means that food and corpses will be consumed by some types of your allies to recover wounds. And you'll have no way of getting anything that your allies pick up off of them, unless you kill them, or have them killed.

It's even worse for your summoned allies. Due to the summoning veil implementation I discussed elsewhere[1], minions you summon will leave your service and disappear after a timeout period. And they'll take everything they picked up with them. That's an easy fix - just have the monster drop everything when they leave. But the more general case will be problematic.

So I'm going to have to do a 'interact with allies' command. This will allow you to get equipment from them, give stuff to them, dismiss them from your service and perhaps a few other house-keeping chores (such as targetting them with spells you'd otherwise only cast on yourself, such as healing). In fact, it should be generic enough to make a general 'talk to monsters' command - so you can always try to buy your equipment back from the thieves who stole it as opposed to kill them. Or buy that rope from an orc standing on the other side of a chasm ('No, throw me the idol').

In the long run, I'll have to implement a more sophisticated orders system. If I ended up implementing battles using a turned based tactical system, you'll want to be able to select only part of your forces and give them an order, as opposed to all of them. Or have an orcish warg rider convey your orders to a unit elsewhere on the battlefield.

Speaking of warg riders, and wolves in general, I fudged things when I said that the flexible 4GAI monster system works perfectly in implementing the monster abilities. And it stood out when I created a werewolf player character and had him appear at the Battle of Five Armies with his wolf cousins. Because they got demolished, repeatedly.

Wolves are a dangerous opponent to the player character, because they move quickly, which means that they attack twice as often. But it also means that the player can get surrounded by them much more easily and attacked from all eight sides. If a player is half way across a room and a door bursts open and wargs attack, he cannot outrun them back to the safety of a single-width corridor. The much-maligned Kheldon Jones pack AI implements exactly this test:

[1] http://roguelikedeveloper.blogspot.com/2007/08/summoning-veils-and-blood-debts.html

it determines how many blank squares are around a player and only has pack monsters advance to attack if there are sufficient squares free to make the player 'vulnerable' (It's maligned because it breaks the 'monsters must advance to stay interesting' paradigm).

But in a large battle-situation, with tens of opponents on either side, the wolf speed advantage gets eroded to just being able to attack faster. And this is not enough to withstand similar level monsters, who will have more attributes scaled up to make them equivalently dangerous to the player. For instance, orcs will hit harder, and have more hit points, than wolves, as they get deeper in the dungeon (It's worth noting that Unangband uses a reasonably sophisticated monster power algorithm to determine relative monster strengths, as opposed to determining monster level 'by hand').

I initially thought 'Ah - hack and back'. This is the traditional tactic to use when you have a speed advantage - where you attack, then step away before the enemy can respond in kind. And the hack and back algorithm is a 'tweak' to the monster minimum and best range, which increases the best range slightly if the monster is next to its target, and the target will get to act before the monster's next turn. Because I neglected to remove the diagnostics for this from the wip7a release, you may have seen the occasional 'backing' message while you were playing to indicate this decision had been made.

But hack and back is not nearly enough. It's a brief advantage only, and the wolves continued to get cut down without mercy. They simply didn't have the space to use hack and back tactics and tended to back themselves into a group around the player and get killed.

The Kheldon Jones pack AI concept came to the rescue. I added a 'vulnerability' weighting to the target selection routine, used if the monster is faster than the target. If this is the case, and the target has more than 4 unoccupied grids surrounding it, the target will be treated as closer than targets without this vulnerability, for the purpose of which enemy the monster will target. Now, when you appear on the battlefield with wolf allies, the wolves will attack the two ends of the advancing enemy front, splitting into two groups and harassing isolated individuals, where they are able to use the same tactics which were so effective against the player.

This concept of reweighting the target selection routine for specific monster behaviour could also be used for traditional racial enemies, so that your allies attack more distant targets if they have a particular hatred of them. It probably should be used in the instance of monsters that are vulnerable or nearly invulnerable to certain types of attacks, so that they are encouraged to fight those enemies which can hurt them the most (Currently ghosts and skeletons, immune to blunt and edged weapons respectively, also 'under-perform'). It shouldn't be abused to much though, as 'closest enemy' is a useful concept to retain.

Chapter 6

Conclusions

Well, what conclusions can I draw from my experience so far with the Unangband monster AI? I'll stick with listing a set of principles that have proved to be a useful guiding framework for anyone else working with roguelike (and other game) AI.

Complex behaviour can emerge from simple rules. There's a great article on swarming behaviour in the New York times that demonstrates the real world rules that ants and other animals use which results in highly sophisticated emergent systems. By adding the MFLAG_PUSH, I was able to get monsters to move in group formations without requiring any global rules. In general, the programmer in you will want to make things complicated. Resist it.

The corollary to the above: Experiment with simple rules first and observe what behaviour emerges. Then see if you can modify the rules to get the desired results and accommodate additional actions. With a simple rule set, it is very easy to determine what underlying monster states result in a particular AI action. The hardest to debug problems I found were where a complex series of tests were used to determine which of a binary choice to select; the easiest where when simple tests were used to determine a range of possible actions.

Monsters are more interesting when they are advancing. In particular, anything that the monster does that threatens the player is perceived as intelligent, be it moving forward, casting spells or shooting arrows. Both the Angband and 4GAI rely primarily on a couple of random checks ('do I try to use a ranged attack?' and 'which ranged attack?') which results in a complex seeming set of behaviours across a range of monster types, by giving each monster a different set of possible choices to make. The underlying algorithms are not sophisticated.

The corrollary: Invest more time in developing AI for monsters when they flee. As soon as a monster stops advancing, you've got the challenge of keeping them interesting and these times will be the point at which the player notices poorly behaved AI routines. I was lucky in this respect that the 4GAI delivers a great platform here, and very little work has been required to improve on it (in fact, I just added bleeding, so that a player can track a wounded monster when it's fled out of sight).

Minimise the number of commands required to control your allies. This was something I decided early on, and it works well. Despite having large numbers of allied monsters, the fact that you don't have to explicitly tell them where to go or what to do means that they feel helpful as opposed to a hindrance. It's not just a matter of having them move out of the way if you walk into them. They should use that as an indication to press forward, or take cover elsewhere. Similarly, if you're shooting at something, maybe they should be shooting at it too.

The corrollary: Have your allies try to do 'the right thing' and do it well. Sure, you can minimise the number of ways of telling them what to do, as long as what they do, they do well. So I deliberately hacked the allied monster AI so that they always fire when they are at the right range (the 'do I use a ranged attack' is always yes), and for a long time, monster vs monster combat never had any misses. Now it's just accelerated so that monsters do double-damage to each other. Monsters should be able to cheat when they fight each other, because the player will get the impression that his allies are effective, while they shouldn't cheat against the player, for the same reason.

Have enemies tell the player what they are doing, ideally, before they do it. And then tell the player what they did. F.E.A.R. is still acclaimed as having some of the best AI of any first person shooter. I was able to take that and expose some pre-existing Angband AI which players may not be aware existed in the game, using the same principles, and added a lot of flavour to combat as a result.

The corollary: If the player doesn't see or hear about it, it didn't happen. Don't waste time on developing a fully featured AI system, where monsters intelligently wander around the dungeon of their own accord.

Unless you are developing a stealth-based roguelike, there is little to no chance that the player will see and understand what they are doing. I simultaneously developed a 'living' dungeon generator, where wolves could be found in kennels, warriors in throne rooms, and priests in temples, and this added much more flavour than I could have done by having out of sight monsters talking to each other, trading items or trying to kill each other. In most games, these events are heavily scripted and taken out of the AI realm. In a roguelike, you don't have the control of the game space enough (due to random dungeon generation) to deliver these experiences.

Finally, design monsters with a range of behaviours, and have each behaviour work well with that monster
It's not good having weak melee monsters advance towards the player, they should stay at range. Similarly, melee monsters should be able to advance - ensure that they can bash through or open doors and other obstacles. The variety of terrain in Unangband is passable by different monsters in a variety of ways. And most monsters should be given a ranged attack along side their melee attacks, even if it is not terribly effective.

The corollary: Ensure that the player has a means of countering each monster attack and defense, or that
Magic using monsters should have a limited mana, archers should run out of arrows or have them destroyed by player acid and fire, summoners should not be able to overwhelm the player in numbers. Each monster should have exploitable weaknesses to keep them interesting.]