

Blades of Avernum

Editor Cookbook

Version 0.3

Written by Erik Westra

PLEASE NOTE:

Blades of Avernum is a fantasy role-playing game created by Spiderweb Software. All copyrights for the BoA game and associated graphics are owned by Spiderweb Software. The use of the name Blades of Avernum in this cookbook does not imply that this cookbook is endorsed or supported by Spiderweb software in any way.

Introduction

The purpose of this cookbook is to provide “recipes” for performing various common tasks within a Blades of Avernum scenario. It is not intended to be a replacement to the Blades of Avernum manuals, but rather to collect together and explain the various elements needed to perform the most common tasks within a scenario. As always, if something is unclear or doesn’t quite cover what you want, please refer to the BoA manual for more information.

This cookbook was written as a way of understanding and learning the BoA editor. It is provided as-is, in the hope that others may also find it useful. If you have any suggestions for how to improve this cookbook, or would like to submit a new recipe or have a correction which you would like to see included in the cookbook, or even just a comment you would like to make, please feel free to contact the author at ewestra@wave.co.nz.

Enjoy!

Erik Westra
April 2004

Contributors

The following people have contributed to this cookbook.

Erik Westra ewestra@wave.co.nz

Table of Contents

| | |
|---|----|
| Introduction..... | 2 |
| Contributors | 3 |
| Table of Contents | 4 |
| The Recipes..... | 5 |
| Entering and Leaving the Scenario | 6 |
| Entering and Leaving Towns | 8 |
| Area Descriptions..... | 10 |
| Doors | 11 |
| Gates | 12 |
| Locked Chests, etc | 13 |
| Traps | 14 |
| Teleporters | 15 |
| Special Items | 16 |
| Ritual of Sanctification | 18 |
| Special Encounters..... | 20 |
| Non-Player Characters | 22 |
| Dialogue..... | 29 |
| Shops..... | 37 |
| Boats and Horses..... | 42 |
| Quests..... | 44 |
| Outdoor Encounters | 47 |
| Splitting and Reuniting the Party | 53 |
| Quickfire | 54 |
| Hills and Heights..... | 56 |
| Appendices..... | 67 |
| Editor Shortcuts | 68 |
| Default Items..... | 69 |
| Default Creatures | 76 |
| Default Terrain Graphics | 85 |
| Default Floor Graphics | 88 |

The Recipes

Entering and Leaving the Scenario

Allowing the party to enter and leave your scenario is quite important. When you create your scenario, you need to tell BoA where the players will start playing, and similarly you need to provide some way the players can leave your scenario, or else they will be stuck in it forever.

When your scenario starts, the party will always be in a town; you cannot start a scenario with your party outdoors.

Setting the starting point for your scenario is a two-step process. First, you need to tell the game which outdoor section, and whereabouts in the outdoor section, the party will be in when they leave the starting town. Usually, this will be adjacent to the starting town. To set the starting outdoor location, edit the outdoor section which contains the starting town, and select the **Set Starting Location** command from the **Outdoors** menu. You will be prompted to click on the point in the outdoor section where the party will appear when they first leave the starting town.

Once you have set the starting outdoor location, you need to set the starting town location. To do this, edit the starting town, and select the **Set Starting Location** command from the **Outdoors** menu. You will be prompted to click on the point in the town where the party will appear when they first start playing the game.

If you would like to have special actions take place when the scenario is first entered (for example, displaying a message to get the party started), you can do this by entering some commands into the START_SCEN_STATE node within your scenario script.

A scenario script is a text file stored in your scenario folder. The script must have the same name as your main scenario file, with the extension '.txt'. At a minimum, your scenario script must contain the following text:

```
beginscenarioscript;  
body;  
  
beginstate LOAD_SCEN_STATE;  
break;  
  
beginstate START_SCEN_STATE;  
break;
```

To have some special actions performed when the players first enter your scenario, place your commands between the “beginstate START_SCEN_STATE” and “break” lines. For example:

```
beginstate START_SCEN_STATE;
    message_dialog("Welcome to my scenario!", "");
break;
```

To allow the players to leave your scenario, you need to use the **end_scenario()** command. For example, you might create a special encounter so that when the players move to the end of a dock, the game asks if they want to leave:

```
beginstate 17;
    reset_dialog();
    if (get_flag(7, 2) == 0) {
        add_dialog_str(0, "You haven't finished the
scenario!", 0);
        add_dialog_str(1, "Give up?", 0);
        add_dialog_choice(0, "Yes");
        add_dialog_choice(1, "No");
        if (run_dialog(0) == 1) {
            end_scenario(0);
        } else {
            block_entry(1);
        }
    } else {
        add_dialog_str(0, "Congratulations on completing
the scenario.", 0);
        add_dialog_str(1, "Are you ready to leave?", 0);
        add_dialog_choice(0, "Yes");
        add_dialog_choice(1, "No");
        if (run_dialog(0) == 1) {
            end_scenario(1);
        } else {
            block_entry(1);
        }
    }
break;
```

Notice how you can personalise the message displayed depending on whether the party have completed the scenario. You could, of course, have lots of different outcomes depending on whether certain conditions were met, and personalise the message appropriately.

To end the scenario, you use the **end_scenario()** command. If the parameter is set to 1, then the party is considered to have completed the scenario, and the computer will add one to their tally of completed scenarios; use **end_scenario(0)** to indicate that the party are leaving without successfully completing the scenario.

Entering and Leaving Towns

There are two ways a party can enter or leave a town: the normal way, using town entrances to move between the town and the outdoor section that contains it, and by using a special encounter to move into or out of a town via a script.

For ordinary towns, the outdoor section will include a “town entrance”, which is a rectangular area within the outdoors. When the party moves into this area, they are transported to the appropriate entrance point for that town. For example, if the party enter the town from the north end, they will be placed at the town’s northern entry point. Simiilary, if they leave the town from the southern edge, they will be placed to the south of the town entrance in the outdoor section.

You place a town within an outdoor section by drawing an appropriate terrain to represent the town (for example, using the small buildings and town walls to draw a small picture of the town as seen from afar). You then use the “Create Town Entrance” tool...



...to specify the top-left and bottom-right corners of the town’s entrance rectangle. Finally, you type the number of the desired town into the “Entrance to what town?” dialog box.

Your next step is to place the entrance points within the town itself. Edit the town, and use the “Place North Entrance”, “Place West Entrance”, “Place South Entrance” and “Place East Entrance” tools...



...to select the point at which the party appears when they enter the town from the given direction. Make sure you leave enough room for the entire party; the game will not place party members on blocked squares, or squares containing terrain which can hurt the party such as lava or swamp.

If you want, you can change the point in the outdoor section where the party appears once they leave town. To do this, select the **Town Details** command from the **Town** menu, and enter the coordinates within the outdoor section into the “Exit Town Location” fields.

So much for straightforward towns. Some towns don’t directly connect with the outdoors; for example, a dungeon might have several levels, each of which are represented by a town, and only the top-most level actually connects with the outdoors. To achieve this, you first have the top-most level connecting to the outdoors in the usual way. Then, you need to use a Special Encounter to move the players down to the next level.

Let's look at this in more detail. Firstly, you want to choose the point within the town that will move the players to a new town. At this point, you will probably want to place some steps leading down, a ladder, or whatever. Once you've placed your terrain, create a special encounter which will be called when the party enter the given space. Then link the special encounter to a town node which includes a **move_to_new_town()** command. For example:

```
beginstate 12;  
    move_to_new_town(4, 12, 13);  
break;
```

The first parameter is the number of the town to move to, while the remaining two numbers are the coordinates you want the party to be placed at. Obviously, if you have some stairs going down in one town or dungeon level, you will want to allow the party to go back up the stairs again to where they were before. To do this, place the party beside some stairs going up in the town they are moved to; you'll also want to place a special encounter over those stairs moving the party back to the same place in the previous town, so that the player can move backwards and forwards as appropriate.

Of course, you don't have to use this just for moving up or down through levels of a dungeon; you can use the same effect to create larger towns, linking together several town sections into a single massive town. In this case, the special encounter at the north end of a town section might move the players to the south end of another town section.

Area Descriptions

To help players understand where they are, and to make gameplay more interesting, it is often helpful to describe the area being walked through. You can do this at two levels: firstly, by naming the entire town or outdoor section. You can then give specific areas within the town or outdoor section more detailed descriptions if you wish.

To name a town, use the **Town Details** command from the **Town** menu. To name an outdoor section, you enter the name using the **Outdoor Details** command in the **Outdoors** menu.

To create a more detailed description of just part of the town or outdoor section, select the “Create Area Description” tool within the BoA editor...



...and then click on the upper-left and lower-right corners of the rectangular area where you want the description to apply. You will then be asked to enter the area’s description, which can be up to 30 characters long.

Each town or outdoor section can have up to eight area descriptions. To edit or remove an existing area description, choose the **Edit Area Descriptions** command from either the **Town** or **Outdoors** menus, as appropriate. You will be able to change the description of the rectangles you have defined, as well as deleting an area description if you wish.

Doors

When you create a new door, the door is initially unlocked. To lock a door, change the door's memory cell 0 to the strength or tool use skill level needed to unlock the door:

- 0** Door is unlocked.
- 5-6** Door is easily unlocked.
- 7-8** Medium difficulty.
- 8+** High difficulty.
- 200** The door cannot be unlocked by any normal means.

To have a door locked until the party have obtained a special item, set the door's memory cell 0 to 200 (so the door is locked), and set memory cell 1 to the number of the special item required to unlock the door.

You can have a door set a stuff done flag to 1 when it is unlocked. To do this, place the coordinates of the SDF into memory cells 2 and 3.

Similarly, you can have a door stay locked until a SDF has a non-zero value; simply set memory cell 0 to 200 (so that the door cannot be unlocked normally), and place the coordinates of the SDF into memory cells 2 and 3. The door will remain locked until the given SDF has a non-zero value.

To create a magic door (a door which can only be opened using an "Unlock" spell), you need to change the terrain script used by the door from "door" to "magicdoor". You then set the door's memory cell 0 value to the level of the "Unlock" spell needed to remove the door's magical defences.

Note that magic doors also allow you to control them using special items and stuff done flags, using the same memory cells as described above for normal doors.

Gates

Gates are typically controlled by levers, though you can also write scripts which open or close a gate under program control.

The simplest way to control a gate is to use the "doorlever.txt" terrain script. To use this script, first place a lever at a suitable point in your town, then place the "doorlever" terrain script on top of, or close to, the lever. Set memory cells 0 and 1 of this script to the X and Y coordinates of the lever, and set memory cells 2 and 3 to the X and Y coordinates of the gate. Voila! When the player walks onto your terrain script, the computer will ask if the player wants to pull the lever. If the player pulls the lever, the gate will be opened and your lever will be toggled to show that it has been pulled.

You can also store the coordinates of a stuff done flag into memory cells 4 and 5. If you do this, the terrain script will remember whether the gate was last open or closed, and will restore the gate to that state when the user re-enters the town.

If you don't want to use levers, you can manually control the opening and closing of a gate by using the **flip_terrain()** command within one of your own scripts. This command takes two parameters, the X and Y coordinates of the gate, and toggles the gate open or closed. You can use this, for example, in a dialogue script so that the player must ask somebody to open the gate for them.

If you use the **flip_terrain()** command to manually open or close a gate within a script, you may also want to use the **play_sound()** command to make an appropriate sound as the gate is opened. For example, the doorlever.txt script includes the following commands:

```
play_sound(106);  
play_sound(99);
```

the first of these makes a 'click' sound, while the second makes a rattling chain sound.

Locked Chests, etc

It is possible to lock a chest, desk, or similar container by making use of the “lockbox.txt” script. To use this script, place a copy of the “lockbox.txt” script into your scenario folder, and place this script onto the chest, box, or whatever it is that you want to lock. Then set the script's memory cells as follows:

- Place the tool skill level needed to unlock the box into memory cell 0. If this is set to zero, the box will be unlocked. Use a value of 200 if you don't want the box to be unlocked using any normal means.
- If you want to have a special item which unlocks the box, store the number of the special item into memory cell 1.
- If you want the lock associated with a Stuff Done Flag, store the coordinates of the SDF into memory cells 2 and 3. In this case, the box will be unlocked if the SDF has a non-zero value, and if the player unlocks the box the flag will be set to 1.

Traps

To place a trap, you must have the "trap.txt" terrain script within your scenario folder. Then place the 'trap' terrain script onto the square where you want the trap to be activated, and set the script's memory cells as follows:

- Store the difficulty of the trap into memory cell 0. This is the tool use skill level needed to disarm the trap. Select a large number (eg, 200) if you want the trap never to be disarmed.
- If you want to allow a special item to disable the trap, enter the number of the special item into memory cell 1. If the party has this special item, the trap will be disarmed.
- You can associate the trap with a Stuff Done Flag by entering the coordinates of the SDF into memory cells 2 and 3. If you do this, the trap will be disarmed if the given SDF has a non-zero value. Similarly, if the player disarms the trap the given SDF will be set to 1, so that the trap will still be disarmed when the player re-enters the terrain script's location.
- Memory cell 4 identifies the type of damage the trap does:
 - 0** A fire explosion which causes (1-8 * severity) points of damage to the character who activated the trap.
 - 1** A fire explosion causing (1-8 * severity) points of damage to everyone within four spaces.
 - 2** An explosion causing (1-8 * severity) points of damage to the character who activated the trap. If the character's health goes below zero, the character is killed.
 - 3** An explosion causing (1-8 * severity) points of damage to everyone within four spaces. If any character's health goes below zero, that character is killed.
 - 4** Spawns a creature.
- The severity of the trap is stored in memory cell 5. If the trap type is 4 (spawn creature), then memory cell 5 is the number of the creature to spawn. Note that creature numbers are listed in the **C1**, **C2**, **C3** and **C4** menus in the BoA editor.

Teleporters

Teleporters are a common feature of BoA scenarios. To place a teleporter in a town, create an appropriate terrain graphic and place a special encounter on that space. Then, in the town node which responds to the special encounter, add a **teleport_party()** command, like this:

```
beginstate 12;  
    teleport_party(7, 12, 0);  
break;
```

The **teleport_party()** command takes three parameters: the first two are the coordinates within the town where the party will be teleported to, while the third parameter specifies a custom special effect used while teleporting; for most teleporters, you can simply set this to zero.

Special Items

There are two types of items in a scenario: ordinary items, and special items. Ordinary items appear in the character's inventory, while special items only appear in the "Special Items" portion of the character info screen. In this section, we will examine how to define and use special items in your scenario.

Each special item has a unique number, in the range 0 through 59. To define a special item requires you to create a scenario script which contains instructions defining the special item(s) used in the scenario. You then use script commands to give or take away special items at appropriate points in your scenario, as well as check to see if your party has a given special item. Finally, you can have doors or chests remain locked until the party has a particular special item, as described in the sections on doors and locked chests.

To start defining a special item, you must first create a scenario script. This is a text file stored in your scenario folder, which has the same name as your main scenario file, with the extension ".txt". At a minimum, this scenario script file should contain the following text:

```
beginscenarioscript;  
  
body;  
  
beginstate LOAD_SCEN_STATE;  
break;
```

You can, of course, place more into this file if it is appropriate for your scenario, but the above is the minimum you need to be able to define a special item. Then, for each of your special items you need to place an **init_special_item()** command into this file, between the "beginstate LOAD_SCEN_STATE;" line and the "break;" command. This command looks like this:

```
init_special_item(itemNum, name, description);
```

where 'itemNum' is the item number of the special item, in the range 0 through 59, 'name' is the name of the item (29 characters max), and 'description' is a brief description of the item (199 characters max). For example:

```
init_special_item(4, "Black Rod", "This rod of pure onyx is  
icy cold to the touch. Merely looking on it fills you with  
dread.");
```


Once you have defined your special item, your next task is to create an appropriate script which gives the party the special item in the appropriate situation. For example, you might want to give the party an item in response while talking with a non-player character (see "dialogue" below), or searching an object (see "special objects" below). No matter what type of script you use, however, you'll want to use the:

```
change_spec_item(itemNum, howMany);
```

command to give the party 'howMany' copies of special item number 'itemNum'. For example:

```
change_spec_item(4, 1);
```

will give the party special item number 4.

To remove a special item, you can either call **change_spec_item()** with a negative 'howMany' value, or else use the:

```
take_special_item(itemNum);
```

command, which takes one special item number 'itemNum' from the party.

Finally, you can check to see how many of a particular special item the party has by using the:

```
has_special_item(itemNum)
```

call, which returns the number of special item number 'itemNum' the party currently has. This means that you can do things like:

```
if (has_special_item(4) > 0) {  
    ...  
}
```

to perform an action only if the party has at least one copy of special item number 4.

Ritual of Sanctification

To allow your party to use the Ritual of Sanctification on something such as an evil altar, you need to first have some means of teaching the ritual to your party. The Ritual of Sanctification ability is lost when you leave a scenario, so you need to include some way of teaching this ritual to your party within your scenario.

To teach the Ritual of Sanctification to the party, you use the **set_char_trait()** command, like this:

```
set_char_trait(0, 22, 1);  
set_char_trait(1, 22, 1);  
set_char_trait(2, 22, 1);  
set_char_trait(3, 22, 1);
```

Don't worry about the specifics -- the first number is simply the number of the party member you are teaching it to, 22 is the magic number for the Ritual of Sanctification, and the last number means that you are giving this ability to the character rather than taking it away. Notice that we give this ability to all four party members, so that any one of them will be able to perform this ritual.

Now that you have given the party the ability to cast the Ritual of Sanctification, you next need to give them something to cast it on. To do this, copy the "sanctitem.txt" terrain script into your scenario folder, and then use the Place Terrain Script tool...



...to place this script on whatever item you wish to have sanctified.

The "sanctitem.txt" script responds to the sanctification by triggering something called a "town node", which is stored in your town script file. If you have not already done so, create a text file for your town script; the script should have the name of your town, followed by ".txt". Then, in the "Town Details" dialog box, enter the name of the town script (minus the ".txt") into the "town script" field.

At a minimum, your town script should look like this:

```
begintownscript;  
body;  
  
beginstate INIT_STATE;  
break;  
  
beginstate EXIT_STATE;  
break;  
  
beginstate START_STATE;  
break;
```

You can, of course, add more commands to your town script if appropriate.

Now, you need to create the town node which will respond to the sanctification. Firstly, you need to assign a number to your town node – any number greater than 2 is available, though you can't have two town nodes with the same number in the same town node.

Let's say that you decide to use the number 17 for your town node. In the town script, add the following lines:

```
beginstate 17;  
break;
```

Now, between these two lines you can place whatever code you like which will respond to the sanctification request. For example, here is a simple script which plays a suitable sound, shows a message to the user, and sets a Stuff Done Flag to indicate that the altar has been sanctified:

```
beginstate 17;  
    play_sound(24); // generic priest spell sound.  
    message_dialog("A strange humming sound fills the air.  
Then, after several moments, you sense the evil  
dissipating.", "The evil spirit has left this altar for  
good.");  
    set_flag(17, 2, 1);  
break;
```

You can use any Avernuscript commands you like in your town node; the sky's the limit!

One last point: before your town node will be called, you must tell the "sanctitem.txt" script which node number you have used; go into the scenario editor, and set memory cell 0 to the node number you have selected.

Special Encounters

You can have a script called whenever the party enter a particular area within a town or outdoor section. To do this, select the “Create Special Encounter” tool in the BoA editor...



...and click on the top-left and bottom-right corners of the rectangular area where you want the encounter to be triggered. You will be asked for the number of a special node which will be called whenever the party enter the given area; choose a unique node number in the range 10 to 99.

You then need to create a suitable town script file (for town encounters) or an outdoor area script file (for outdoor encounters), if you haven’t already done so. If your encounter is in a town, you need to create a town script file; this file should have the name of your town, followed by “.txt”. Then, in the “Town Details” dialog box, enter the name of the town script (minus the “.txt”) into the “town script” field.

At a minimum, your town script should look like this:

```
begintownscript;  
body;  
  
beginstate INIT_STATE;  
break;  
  
beginstate EXIT_STATE;  
break;  
  
beginstate START_STATE;  
break;
```

You can, of course, add more commands to your town script if appropriate.

For outdoor encounters, you need to create an outdoor area script and give it some appropriate name ending in “.txt”. Then, in the “Outdoor Area Details” dialog box, enter the name of the outdoor area script (minus the “.txt”) into the input field labelled “Section Script”.

Outdoor area scripts are much simpler than town scripts; all you need to put in the script is:

```
beginoutdoorscript;  
body;
```

Once you have created your town or outdoor area script, you need to define the node which is called when the special encounter is activated. To do this, enter the following into the town or area script:

```
beginstate <node-number>;  
break;
```

where <node-number> is the number of the node you selected for this encounter. Between the “beginstate” and “break” lines, you enter the Avernuscript commands which will be called when the party enter this area. For example, the following commands create a custom dialog box which asks the player if they want to do something

```
beginstate 17;  
    reset_dialog();  
    add_dialog_str(0, "The crystal hums slightly.", 0);  
    add_dialog_str(1, "Smash it?", 0);  
    add_dialog_choice(0, "Yes");  
    add_dialog_choice(1, "No");  
    if (run_dialog(0) == 1) {  
        set_flag(3, 7, 1);  
    }  
    block_entry(1);  
break;
```

Notice how you create a custom dialog box by specifying the individual pieces of text and the buttons which make it up, you display the dialog box by “running” it, and can perform different actions depending on which choice the user selected. In this example, we set a Stuff Done Flag to 1 if the user chooses to smash a crystal. Note that we use the **block_entry()** command to stop the party from entering the space.

Of course, you have the full power of Avernuscript available to you when you tell the computer how to respond to the special encounter. While many of your encounters will be as simple as the one above, you can do far more complicated things with your scripts if you wish to do so.

Non-Player Characters

The term “Non-Player Character” (or NPC for short) refers to any person or creature in your scenario other than the player’s party members. Obviously, interacting with NPCs forms a huge part of any scenario.

To place an NPC within a town, select the type of character you want from the **C1**, **C2**, **C3** or **C4** menu. If all you want is a simple monster or townspeople, all you need to do is select the type of creature, and then choose an appropriate starting attitude in the “Edit Placed Creature” dialog box. These starting attitudes are:

| | |
|------------------------|---|
| Friendly | Won’t attack the party, but will attack the party’s enemies. |
| Neutral | Won’t attack either the party or the party’s enemies. |
| Hostile, Type A | Will attack both the party and any “Hostile, Type B” creatures. |
| Hostile, Type B | Will attack both the party and any “Hostile, Type A” creatures. |

The creature won’t talk or have a name, but this is often sufficient for the minor NPCs within your scenario.

Notice that each type of creature has a “level”, which indicates how hard the creature is to fight or kill. To prevent your scenario from being too easy or too hard, you should try and select monsters of about the same level as the party you expect will be playing your scenario. You can also change the level of a predefined creature to be more in line with the level of the party who will be playing the scenario; to do this, you need to place a **set_creature_type_level()** command within the **LOAD_SCEN_STATE** portion of your scenario script. For more information on this, please refer to the BoA editor documentation.

Creature Scripts

For more complicated characters, you will probably want to use one of the predefined creature scripts provided with BoA. To use a creature script, you enter the name of the script (minus the “.txt” extension) into the “Creature Script” field in the “Edit Placed Creature” dialog box. You then enter values into the script’s memory cells as appropriate for that type of script.

Let’s look at how you can use some of the most common creature scripts to bring your NPCs to life.

Firstly, there is the “basicnpc.txt” script, which provides a very basic sort of NPC, but is probably sufficient for most of your characters. When you use this script, you should fill in the script’s memory cells (in the “Edit Placed Creature” dialog box) as follows:

- If you want the NPC to wander at random, put the value 0 into memory cell 0.
- If you want the NPC to stand still until an enemy appears, enter 1 into memory cell 0.
- If you want the NPC to remain completely still, even if an enemy is visible, set memory cell 0 to 2.
- If you want to have a Stuff Done Flag set to 1 when this NPC is killed, place the coordinates of the desired Stuff Done Flag into memory cells 1 and 2.
- If you want the NPC to be able to talk with the players, enter the number of the starting dialogue node into memory cell 3. If this is left at zero, the NPC won’t be able to talk.

A slightly more advanced form of non-player character can be created by using the “guard.txt” script instead of “basicnpc.txt”. The guard works exactly the same as the basic NPC, except that it will hunt down the party if the town is angered (eg, because the party stole an item or attacked a friendly NPC). The memory cells are identical to the basic NPC, except for the following additional choice:

- If you want the NPC to patrol an area by walking from its starting point to a given waypoint and then back again, set memory cell 0 to 3, and place the desired waypoint number into memory cell 4.

Other, more advanced NPC scripts are provided in the “Sample Scripts” folder provided with the BoA editor:

| | |
|------------------------------|---|
| <code>abilnpc.txt</code> | An NPC who can use the spells curse, weaken, slow, shield, bless and haste while in combat. |
| <code>assassin.txt</code> | an NPC who will hunt the party down and then attack one character until he or she is dead before starting to attack the next character. |
| <code>farhunter.txt</code> | Similar to <code>hunter.txt</code> , except that it can hunt down the party from far away. |
| <code>fleeoncenpc.txt</code> | An NPC who flees to a given waypoint when wounded. |
| <code>flocknpc.txt</code> | An NPC which follows another NPC. |
| <code>groupnpc.txt</code> | An NPC who alerts the other members of its group when it first spots an enemy, so that all the NPCs in the same group will join in the fight. |
| <code>healnpc.txt</code> | An NPC who heals wounded allies. |
| <code>hunter.txt</code> | an NPC which will hunt the party down if it is hostile. |
| <code>patrolguard.txt</code> | A guard who walks endlessly between two points. When it encounters an enemy, it fights for one round and then runs back to its starting point, where it alerts the other members in its group |
| <code>spawner.txt</code> | an NPC which doesn't move, but which spawns a number of hostile creatures every few moves. |
| <code>splitter.txt</code> | an NPC which spawns other monsters when it dies. |
| <code>twopts.txt</code> | an NPC who walks endlessly between two waypoints. |

If you want to use any of these scripts, copy the script file into your scenario folder, and edit the script to see which memory cell values it requires.

Custom Creature Scripts

If one of the predefined creature scripts doesn't do what you want, you can create your own creature script instead. Most of the time, you can simply take one of the existing scripts and change it to do what you want. Make sure you copy the existing script and give it a different name, so that the original creature script is still available if you want to use it elsewhere.

The most common thing you'll want to use a custom creature script for is to have something special happen when the creature is killed. To do this, decide on which of the existing creature scripts you want to use, and copy it, giving it an appropriate name. For example, let's say that you have a rogue NPC captain by the name of Garth who drops a key when he is killed by the party. Let's base Garth's creature script on the `basicnpc` script; copy `basicnpc.txt` and change its name to `garth.txt` (you can use whatever name you like; this is just a suggestion).

Now, edit the **garth.txt** creature script in your favourite text editor. Most of the stuff in this script you can leave unchanged; however, to make something special happens when Garth is killed, you need to replace the existing DEAD_STATE code with your own instructions.

DEAD_STATE is a special state in the creature script called when the creature is killed by the party. Let's replace the assassin's default DEAD_STATE (which sets a Stuff Done Flag to a given value) with our own code:

```
beginstate DEAD_STATE;  
    message_dialog("Garth drops a key as he dies.", "");  
    change_spec_item(4, 1);  
break;
```

This gives the party special item #4 (the key) when Garth is killed. Obviously, you can do far more complicated things in response to a character dying if you wish, but this gives you the basic idea of how to have special actions occur when a creature is killed.

When you do use a custom creature script, don't forget to enter the name of your custom script (minus the ".txt" extension) into the "Creature Script" field in the "Edit Placed Creature" dialog box.

Limiting where your NPCs can Go

For a number of reasons, you may want to prevent your NPCs from passing through certain areas of your town. A simple example might be to prevent a shopkeeper from leaving his or her shop. You can also prevent a monster from standing behind a secret door, which would make it impossible for the party to pass through that door.

To prevent your NPCs from entering a particular space, click on the "Make Space Blocked" tool in the BoA editor...



...and then click on the desired space(s). A small **B** will appear in the corner of the space, indicating that no NPCs can enter that space. To remove a block, click on the "Clear Space" tool...



...and then click on the space(s) you want to clear.

Names and Bubble Text

By default, your NPCs will take their names from the type of creature you selected when you created the NPC. For example, “Priest” or “Guard”. If you want to give your NPC a more specific name, you need to place a **set_name()** command into your town script’s INIT_STATE node. For example:

```
beginstate INIT_STATE;  
    set_name(7, "Captain Greaves");  
break;
```

Note that the number to use in the **set_name()** command is the “creature number” for the NPC, as displayed in the “Edit Placed Creature” dialog box.

One of the more interesting features of Blades of Avernum is the ability to have “text bubbles” appear above a character to represent what they are saying. To have a text bubble appear above your NPC, you use the **text_bubble_on_char()** command. This command takes two parameters: the NPC’s “creature number,” and the text you want to appear. For example, the command:

```
text_bubble_on_char(7, "Where's Me Parrot?");
```

will cause the text “Where’s Me Parrot?” to appear for a short time above creature number 7.

Because text bubbles only appear for a short time, you need to make use of the START_STATE node in your town script, which is called every turn while the party is in this town. You can use an “if” command to only show a text bubble occasionally, like this:

```
if (get_ran(1, 0, 100) < 10)  
    if (get_flag(7, 3) == 0)  
        text_bubble_on_char(7, "Where's Me Parrot?");  
    else  
        text_bubble_on_char(7, "I'm happy.");
```

In this case, there is a 10% chance ($\text{get_ran}(1, 0, 100) < 10$) a text bubble will be displayed in any given turn. As you can see, you can also use Stuff Done Flags to control which message is displayed depending on what else has happened in the game.

Allowing NPCs to Join the Party

In Blades of Avernum, it is possible for up to two NPCs to join the player's party and accompany them for a while. To allow this, you need to add an:

```
enable_add_chars(1);
```

command to the town's INIT_STATE node. This tells the game to allow you to add characters to the party in this town. Then, in an appropriate place such as one of your NPC's dialogue nodes, you need to add an **add_char_to_party()** command, like this:

```
beginstalknode 16;  
    state = 6;  
    nextstate = -1;  
    question = "Would you like to join our party?";  
    text1 = "_Sure._";  
    text2 = "_Sorry, your party is full._";  
    code =  
        if (add_char_to_party(16) == FALSE) {  
            remove_string(1);  
        } else {  
            remove_string(2);  
        }  
    break;
```

Notice that the **add_char_to_party()** command will return FALSE if the party already has two extra characters; you can use this to change the text which is displayed, as in the above example.

If you allow an NPC to join your party, you may want to provide a way of having the character leave the party again. An obvious way of doing this is to set up a special encounter at the place where the party met the character, and link this encounter to a script like the following:

```
beginstate 18;  
    if (character_in_party(16) >= 0) {  
        reset_dialog();  
        add_dialog_string("Should I stay with you?");  
        add_dialog_choice(0, "Stay with us.");  
        add_dialog_choice(1, "Bye!");  
        if (run_dialog(1) == 2) {  
            erase_char(character_in_party(16));  
            spawn_creature(16);  
        }  
    }  
    break;
```

This script checks to see if the NPC is in the party, and if so asks if he/she should stay with the party or leave. If the player chooses to have the NPC leave, the NPC is removed from the party using the **erase_char()** command, and then the NPC is recreated back in his/her original position in the town by using the **spawn_creature()** command.

Changing a Character's Attitude

In a typical scenario, an NPC may start out friendly, but then change his or her attitude and attack the party, for example in response to a particular comment or event occurring. For example, a dragon may initially be friendly or neutral, but will quickly turn hostile if the party attempt to steal his treasure.

To change an NPC's attitude, you need to make use of the **set_attitude()** command:

```
set_attitude(27, 10);
```

The first number is the creature number for the NPC, while the second number indicates the character's attitude:

| | |
|-----------|------------|
| 3 | Friendly. |
| 4 | Neutral. |
| 10 | Hostile A. |
| 11 | Hostile B. |

A typical place to do this would be in a dialogue node or a special encounter.

You can also see what an NPC's current attitude is by using the **get_attitude()** command. For example:

```
if (get_attitude(27) == 10)  
    message_dialog("The dragon is not happy!", "");
```

This can be useful in certain circumstances, for example within a special encounter.

Dialogue

Dialogue is basically a two-way conversation between the party and a non-player character within the game. To support dialogue within a town, you must create a "dialogue script" for the town, and then fill that dialogue script with the conversation you want to take place.

As well as simply talking with NPC's, you can use a dialogue script to perform other actions such as unlocking a door, bestowing a special item on the party, buying and selling, etc.

Before you can create a dialogue script for a town, you must create a town script, even if your town script doesn't do anything. By convention, a town script is given the name of the town with the file extension ".txt", though you can actually use any name you like.

Create the town script file within your scenario folder, and enter at least the following text to define the town script:

```
begintownscript;  
body;  
  
beginstate INIT_STATE;  
break;  
  
beginstate EXIT_STATE;  
break  
  
beginstate START_STATE;  
break
```

You can, of course, add more commands to your town script if you wish.

Now that you have created your town script, run the BoA editor and select the **Town Details** command from the **Town** menu. Then type the name of your town script (minus the ".txt" extension) into the "Town Script" field. This will tell BoA to use the town script you have specified.

Once you have created the town script, you can finally start to create your town's dialogue script. The dialogue script must have the same name as the town script, except that instead of ".txt" at the end, your file must end in ".dlg.txt". For example, if your town is named "Hamner", your town script would be "Hamner.txt", and your town's dialogue script would be named "Hamnerdlg.txt".

Create the dialogue script, and place the following at the start of the file:

```
begintalkscript;
```

After this, you should place each of the "dialogue nodes" you want to have in your town, one after the other. A dialogue node is basically a question and its associated answer; a typical dialogue node might look like this:

```
begintalknode 8;  
    state = 3;  
    nextstate = -1;  
    question = "Have you seen Joe?";  
    text1 = "_No._";
```

There are several things to notice about a dialogue node. Firstly, the number at the end of the 'begintalknode' line (in this case 8) is a unique 'node number' identifying this node within the conversation. There can be at most 200 dialogue nodes in a script, numbered in the range 0 to 199; apart from telling the dialogue nodes apart, the node number can be used to start the conversation at a particular point, as described in the section on non-playing characters, above.

The lines of text following the 'begintalknode' line are indented, and provide the details of the dialogue node. You should then have a blank line before the start of the next 'begintalknode' line, to separate the individual dialogue nodes so that you can see them easier.

Notice that the question and answer are defined as text, using quote marks to identify the start and end of the text. So, in the above example, the question is:

Have you seen Joe?

By default, the quote marks are not displayed. Since you need to use quote marks to identify the start and end of your text, you can't include quote marks as part of your response. You can, however, trick the game into displaying quote marks by using underscore characters ("_") to represent the quote mark. For example:

```
text1 = "_No._";
```

is actually displayed as:

"No."

So far, we have used dialogue nodes to represent a single question and its associated answer. There is another use for dialogue nodes, however: to describe the character being talked to. For example, a typical conversation with a guard might start with something like the following:

```
begintalknode 7;  
    state = -1;  
    nextstate = 3;  
    question = "Town Guard";  
    text1 = "The guard eyes you suspiciously. _What do you  
want?_";
```

For these 'opening' dialogue nodes, the 'state' should be set to -1, the 'question' is actually the name of the character you are talking with, and the 'answer' text is the description of the character that is displayed when you first start conversing with that character.

Critical to understanding dialogue scripts is to understand the idea of a "dialogue state". Each node has a state associated with it; only those dialogue nodes which belong to the current dialogue state will be shown at any one time; as the player selects a dialogue node's question, the answer is displayed and the conversation moves to the dialogue state specified by the dialogue node's 'nextstate' value.

It is important that you understand the idea of a dialogue state; the dialogue state is quite separate from the dialogue node number; each state can have multiple nodes associated with it, representing the various choices the player has at any given point in the conversation. Furthermore, each dialogue state number must be unique within the dialogue script; you cannot re-use dialogue state numbers for different conversations within the same town.

Notice that the above dialogue node, which starts a conversation with a town guard, sets the 'nextstate' value to 3. This means that all dialogue nodes which belong to dialogue state 3 will be displayed as possible choices within the conversation. For example:

```
begintalknode 8;
    state = 3;
    nextstate = -1;
    question = "Have you seen Joe?";
    text1 = "_No._";

begintalknode 9;
    state = 3;
    nextstate = 4;
    question = "Can you let me in, please?";
    text1 = "_What's the password?_";

begintalknode 10;
    state = 4;
    nextstate = -1;
    question = "Um, dunno.";
    text1 = "The guard turns his back on you.";

begintalknode 11;
    state = 4;
    nextstate = 5;
    question = "_open sesame_";
    text1 = "The guard scowls.  _All right, you can come
in._";
```

In this example, there would be two questions available when you first start conversing with the guard:

Have you seen Joe?

Can you let me in, please?

As you can see, the answer to the first question is simply "No", while the answer to the second question is another question, which must be answered correctly before you can proceed.

Notice that the response is identified by a line labelled 'text1'. You can actually have up to eight lines of responses, labelled 'text1', 'text2', 'text3', 'text4', 'text5', 'text6', 'text7' and 'text8'. each one of these response lines can be up to 255 characters long; be aware, though, that if your response may be too long to show in the window, especially if several possible choices have to be shown as well.

As well as simply answering questions, you can use dialogue nodes in much more powerful ways. Firstly, you can specify a condition which must apply if the dialogue node is to be shown. For example:

```
begintalknode 18;  
    state = 5;  
    nextstate = 6;  
    condition = get_flag(3, 5) == 1;  
    question = "Why?";  
    text1 = "Why not?";
```

You can use any Avernuscript function to decide which condition must be met if the dialogue node is to be shown. Common conditions are:

```
get_flag(x, y) == z
```

The node will only be shown if the Stuff Done Flag at coordinate (x, y) has the value z.

```
has_special_item(x) > 0
```

The party has at least one of special item number x.

There are a number of predefined actions which you can also include in a dialogue node. These are identified using an 'action =' line:

- You can vary the text which is displayed based upon the value of a Stuff Done Flag, by using the "DEP_ON_SDF" action, like this:

```
action = DEP_ON_SDF 7 12 1;
```

The first two numbers are the coordinates of the Stuff Done Flag, while the third number is the value to compare that SDF against. If the Stuff Done Flag has a value less than or equal to the given number, text 1 and 2 will be displayed; otherwise text3 and text4 will be shown.

- You can set the value of a Stuff Done Flag using the "SET_SDF" action:

```
action = SET_SDF 7 12 0;
```

The first two numbers are the coordinates of the Stuff Done Flag, while the third number is the value you want the SDF set to.

- You can change the value of a Stuff Done Flag and vary the text displayed depending on the value of that flag, all at once, by using the "SET_TO_1" action:

```
action = SET_TO_1 7 12;
```

The two numbers are the coordinates of the desired Stuff Done Flag. If that SDF has the value 0, it will be set to 1 and text1 and text2 will be displayed. Otherwise, text3 and text4 will be displayed.

- You can force the party to pay for a response by using the "PAY" action:

```
action = PAY 7 12 1 200;
```

The first two numbers are the coordinates of a Stuff Done Flag, while the third number is the value that Stuff Done Flag will be set to once the party has paid, and the fourth number is the amount of gold the party must pay for the response.

If the given SDF already has the given value, then the party has already paid for this response. In this case, text5 and text6 is displayed. Otherwise, the party still has to pay for the response; if they can afford it, they are charged the given amount, the SDF is set to the given value, and text1 and text2 are displayed. If they can't afford to pay, text3 and text4 are displayed.

- You can stop the conversation immediately by using the "END_TALK" action:

```
action = END_TALK;
```

- You can vary the text displayed depending upon whether the party have previously talked with this character or not. Doing this is a two-step process; firstly, you need to include an "INTRO" action in your dialogue node, like this:

```
action = INTRO;
```

Then you need to give this character a "personality number", which identifies this character within the game. If the party have talked in the past with a character with the given personality number, text1, 2, 3, and 4 will be displayed. Otherwise, text5, 6, 7, and 8 will be shown.

Personality numbers are in the range 0 to 3999, and must be unique within the game. You enter a creature's personality number into the "Edit Placed Creature" dialog box in the editor, and you must also identify the personality within the creature's dialogue, by adding a:

```
personality = x;
```

line to each of the creature's dialogue nodes.

- You can have the NPC act as an innkeeper, charging the party for a room where they can rest up. To do this, you use the "INN" action:

```
action = INN 50 7 12;
```

The first number is the amount the innkeeper charges for a room, while the remaining two numbers are the X and Y coordinates of the room the party is moved to if they choose to stay at the inn. The party are healed, their spell points are recharged, and they are teleported to the given coordinates.

The text displayed depends upon two conditions: whether the party is on horseback, and whether the party can afford to pay. If the party can afford to pay and is not on horseback, text1 and text2 are displayed and the conversation ends. If the party cannot afford to pay, text3 and text4 are displayed. If the party is on horseback, text5 is displayed.

- You can have the NPC offer to identify the party's items by using the "ID" action:

```
action = ID 10;
```

The number is the amount the NPC charges to identify each item.

Finally, you can include arbitrary commands which will be performed when the given dialogue node is selected, for example:

```
begintalknode 19;  
  state = 6;  
  nextstate = -1;  
  question = "I want to be your slave.";  
  text1 = "The lich smirks.  _Very well._";  
  code =  
    set_flag(3, 2, 1);  
  break;
```

All of the Avernuscript code between the 'code =' and 'break;' lines will be executed when the given question is selected; in this example, the Stuff Done Flag at coordinate (3, 2) will be set to the value 1.

Within your code, you can use the following commands to choose which parts of the response text are displayed:

```
clear_strings() ;
```

This hides all of the response text associated with this dialogue node.

```
add_string(n) ;
```

This shows response text 'n' (a number in the range 1 to 8). You should use this after a **clear_string**() command to make a particular response text visible again.

```
remove_string(n) ;
```

This hides response text 'n' (a number in the range 1 to 8).

For example, the following commands:

```
clear_strings() ;  
add_string(3) ;
```

would cause all the response text to be hidden, except for 'text3'.

Shops

To create a shop, you need to make use of the **begin_shop_mode()** command. You would normally do this within the "code" section of a dialogue node, so that a particular response to a conversation will start the buying and/or selling process. However, you can use the **begin_shop_mode()** command in any script you like, for example as part of a special encounter in the outdoors.

Before you can buy or sell items, you need to set up a shop. To do this, you must first create a scenario script.

A scenario script is a text file stored in your scenario folder. The script must have the same name as your main scenario file, with the extension '.txt'. At a minimum, your scenario script must contain the following text:

```
beginscenarioscript;  
body;  
  
beginstate LOAD_SCEN_STATE;  
break;  
  
beginstate START_SCEN_STATE;  
break;
```

You can, of course, have lots more in your scenario script; this is the minimum you need to get started.

Each shop in your scenario must have a unique number in the range 0 to 129. Once you have assigned a number to your shop, you can start to fill it by adding **add_item_to_shop()** commands to your scenario script, between the 'beginstate START_SCEN_STATE;' line, and the 'break;' which follows it.

The **add_item_to_shop()** command requires three numbers; the first number is the number of the shop, while the remaining two items identify the item to add to the shop.

To add an ordinary item to your shop, you first need to find the number of the item you want to add. The **I1**, **I2**, **I3**, **I4** and **I5** menus in the BoA editor show the number associated with each item. Once you know the item number, you should place an **add_item_to_shop()** command into your scenario script, like this:

```
add_item_to_shop(1, 26, 3)
```

This will give three suits of leather armour (item #26) to shop number one. As you can see, the third number is the number of this item the shop has in stock; if you give a shop more than 500 of a particular item, that shop's supply will be infinite.

Note that an individual shop can have no more than 25 unique items.

As well as ordinary items, shops can sell mage and priest spells, alchemy and skills. To add a mage spell to a shop, use one of the following **add_item_to_shop** commands:

| | |
|--|--|
| <code>add_item_to_shop(x, 2000, y);</code> | adds Bolt of Fire, max level y, to shop x. |
| <code>add_item_to_shop(x, 2001, y);</code> | adds Light, max level y, to shop x. |
| <code>add_item_to_shop(x, 2002, y);</code> | adds Call Beast, max level y, to shop x. |
| <code>add_item_to_shop(x, 2003, y);</code> | adds Spray Acid, max level y, to shop x. |
| <code>add_item_to_shop(x, 2004, y);</code> | adds Haste, max level y, to shop x. |
| <code>add_item_to_shop(x, 2005, y);</code> | adds Slow, max level y, to shop x. |
| <code>add_item_to_shop(x, 2006, y);</code> | adds Ice Lances, max level y, to shop x. |
| <code>add_item_to_shop(x, 2007, y);</code> | adds Unlock Doors, max level y, to shop x. |
| <code>add_item_to_shop(x, 2008, y);</code> | adds Create Illusions, max level y, to shop x. |
| <code>add_item_to_shop(x, 2009, y);</code> | adds Far Sight, max level y, to shop x. |
| <code>add_item_to_shop(x, 2010, y);</code> | adds Lightening Spray, max level y, to shop x. |
| <code>add_item_to_shop(x, 2011, y);</code> | adds Capture Mind, max level y, to shop x. |
| <code>add_item_to_shop(x, 2012, y);</code> | adds Simulacrum, max level y, to shop x. |
| <code>add_item_to_shop(x, 2013, y);</code> | adds Dispel Barrier, max level y, to shop x. |
| <code>add_item_to_shop(x, 2014, y);</code> | adds Summon Aid, max level y, to shop x. |
| <code>add_item_to_shop(x, 2015, y);</code> | adds Forcecage, max level y, to shop x. |
| <code>add_item_to_shop(x, 2016, y);</code> | adds Fireblast, max level y, to shop x. |
| <code>add_item_to_shop(x, 2017, y);</code> | adds Arcane Summon, max level y, to shop x. |
| <code>add_item_to_shop(x, 2018, y);</code> | adds Arcane Shield, max level y, to shop x. |
| <code>add_item_to_shop(x, 2019, y);</code> | adds Arcane Blow, max level y, to shop x. |

To add a priest spell:

| | |
|--|--|
| <code>add_item_to_shop(x, 3000, y);</code> | adds Healing, max level y, to shop x. |
| <code>add_item_to_shop(x, 3001, y);</code> | adds Curing, max level y, to shop x. |
| <code>add_item_to_shop(x, 3002, y);</code> | adds War Blessing, max level y, to shop x. |
| <code>add_item_to_shop(x, 3003, y);</code> | adds Terror, max level y, to shop x. |
| <code>add_item_to_shop(x, 3004, y);</code> | adds Repel Spirit, max level y, to shop x. |
| <code>add_item_to_shop(x, 3005, y);</code> | adds Smite, max level y, to shop x. |
| <code>add_item_to_shop(x, 3006, y);</code> | adds Summon Shade, max level y, to shop x. |
| <code>add_item_to_shop(x, 3007, y);</code> | adds Enduring Barrier, max level y, to shop x. |
| <code>add_item_to_shop(x, 3008, y);</code> | adds Unshackle Mind, max level y, to shop x. |
| <code>add_item_to_shop(x, 3009, y);</code> | adds Move Mountains, max level y, to shop x. |
| <code>add_item_to_shop(x, 3010, y);</code> | adds Mass Healing, max level y, to shop x. |
| <code>add_item_to_shop(x, 3011, y);</code> | adds Mass Curing, max level y, to shop x. |
| <code>add_item_to_shop(x, 3012, y);</code> | adds Radiant Shield, max level y, to shop x. |
| <code>add_item_to_shop(x, 3013, y);</code> | adds Divine Fire, max level y, to shop x. |
| <code>add_item_to_shop(x, 3014, y);</code> | adds Control Foes, max level y, to shop x. |
| <code>add_item_to_shop(x, 3015, y);</code> | adds Cloud of Blades, max level y, to shop x. |
| <code>add_item_to_shop(x, 3016, y);</code> | adds Return Life, max level y, to shop x. |
| <code>add_item_to_shop(x, 3017, y);</code> | adds Divine Retribution, max level y, to shop x. |
| <code>add_item_to_shop(x, 3018, y);</code> | adds Divine Restoration, max level y, to shop x. |
| <code>add_item_to_shop(x, 3019, y);</code> | adds Divine Host, max level y, to shop x. |

To add alchemy recipes:

| | |
|--|---|
| <code>add_item_to_shop(x, 4000, 1);</code> | adds Healing potion recipe to shop x. |
| <code>add_item_to_shop(x, 4001, 1);</code> | adds Curing potion recipe to shop x. |
| <code>add_item_to_shop(x, 4002, 1);</code> | adds Hasting Potion recipe to shop x. |
| <code>add_item_to_shop(x, 4003, 1);</code> | adds Energy Potion recipe to shop x. |
| <code>add_item_to_shop(x, 4004, 1);</code> | adds Strength Potion recipe to shop x. |
| <code>add_item_to_shop(x, 4005, 1);</code> | adds Graymold Salve recipe to shop x. |
| <code>add_item_to_shop(x, 4006, 1);</code> | adds Balm of Life recipe to shop x. |
| <code>add_item_to_shop(x, 4007, 1);</code> | adds Healing Elixir recipe to shop x. |
| <code>add_item_to_shop(x, 4008, 1);</code> | adds Hasting Elixir recipe to shop x. |
| <code>add_item_to_shop(x, 4009, 1);</code> | adds Energy Elixir recipe to shop x. |
| <code>add_item_to_shop(x, 4010, 1);</code> | adds Rogue's Elixir recipe to shop x. |
| <code>add_item_to_shop(x, 4011, 1);</code> | adds Strength Elixir recipe to shop x. |
| <code>add_item_to_shop(x, 4012, 1);</code> | adds Bliss Elixir recipe to shop x. |
| <code>add_item_to_shop(x, 4013, 1);</code> | adds Restoration Brew recipe to shop x. |
| <code>add_item_to_shop(x, 4014, 1);</code> | adds Protection Brew recipe to shop x. |
| <code>add_item_to_shop(x, 4015, 1);</code> | adds Heroic Brew recipe to shop x. |
| <code>add_item_to_shop(x, 4016, 1);</code> | adds Knowledge Brew recipe to shop x. |

To add a skill:

| | |
|--|--|
| <code>add_item_to_shop(x, 5000, y);</code> | adds Strength, max level y, to shop x. |
| <code>add_item_to_shop(x, 5001, y);</code> | adds Dexterity, max level y, to shop x. |
| <code>add_item_to_shop(x, 5002, y);</code> | adds Intelligence, max level y, to shop x. |
| <code>add_item_to_shop(x, 5003, y);</code> | adds Endurance, max level y, to shop x. |
| <code>add_item_to_shop(x, 5004, y);</code> | adds Melee Weapons, max level y, to shop x. |
| <code>add_item_to_shop(x, 5005, y);</code> | adds Pole Weapons, max level y, to shop x. |
| <code>add_item_to_shop(x, 5006, y);</code> | adds Bows, max level y, to shop x. |
| <code>add_item_to_shop(x, 5007, y);</code> | adds Thrown Missiles, max level y, to shop x. |
| <code>add_item_to_shop(x, 5008, y);</code> | adds Hardiness, max level y, to shop x. |
| <code>add_item_to_shop(x, 5009, y);</code> | adds Defense, max level y, to shop x. |
| <code>add_item_to_shop(x, 5010, y);</code> | adds Assassination, max level y, to shop x. |
| <code>add_item_to_shop(x, 5011, y);</code> | adds Mage Spells, max level y, to shop x. |
| <code>add_item_to_shop(x, 5012, y);</code> | adds Priest Spells, max level y, to shop x. |
| <code>add_item_to_shop(x, 5013, y);</code> | adds Arcane Lore, max level y, to shop x. |
| <code>add_item_to_shop(x, 5014, y);</code> | adds Potion Making, max level y, to shop x. |
| <code>add_item_to_shop(x, 5015, y);</code> | adds Tool Use, max level y, to shop x. |
| <code>add_item_to_shop(x, 5016, y);</code> | adds Nature Lore, max level y, to shop x. |
| <code>add_item_to_shop(x, 5017, y);</code> | adds First Aid, max level y, to shop x. |
| <code>add_item_to_shop(x, 5018, y);</code> | adds Luck, max level y, to shop x. |
| <code>add_item_to_shop(x, 5019, y);</code> | adds Quick Strike, max level y, to shop x. |
| <code>add_item_to_shop(x, 5020, y);</code> | adds Parry, max level y, to shop x. |
| <code>add_item_to_shop(x, 5021, y);</code> | adds Blademaster, max level y, to shop x. |
| <code>add_item_to_shop(x, 5022, y);</code> | adds Anatomy, max level y, to shop x. |
| <code>add_item_to_shop(x, 5023, y);</code> | adds Gymnastics, max level y, to shop x. |
| <code>add_item_to_shop(x, 5024, y);</code> | adds Pathfinder, max level y, to shop x. |
| <code>add_item_to_shop(x, 5025, y);</code> | adds Magery, max level y, to shop x. |
| <code>add_item_to_shop(x, 5026, y);</code> | adds Resistance, max level y, to shop x. |
| <code>add_item_to_shop(x, 5027, y);</code> | adds Magical Efficiency, max level y, to shop x. |
| <code>add_item_to_shop(x, 5028, y);</code> | adds Lethal Blow, max level y, to shop x. |
| <code>add_item_to_shop(x, 5029, y);</code> | adds Riposte, max level y, to shop x. |
| <code>add_item_to_shop(x, 5030, y);</code> | adds Sharpshooter, max level y, to shop x. |

Once you have defined your shop in the scenario script, you now need to tell the game when you want the party to be able to go shopping. As described above, you will often do this as part of the "code =" section of a dialogue node, but you can also turn on shopping at any point where a script is run.

To start shopping, use the **begin_shop_mode()** command, like this:

```
begin_shop_mode("Purdee's Resort", "The meals here are very  
good but expensive", 7, 5, 3);
```

The first item is the name of the shop, and the second item is a brief description. The third item (in this case, 7) is the number of the shop, while the fourth item adjusts the price and the last item adjusts the amount the shop will pay for items.

The price adjustment value should be one of the following:

- 0** charges 60% of the regular cost.
- 1** charges 80% of the regular cost.
- 2** charges 100% of the regular cost.
- 3** charges 130% of the regular cost.
- 4** charges 180% of the regular cost.
- 5** charges 220% of the regular cost.
- 6** charges 280% of the regular cost.

The buy adjustment value should be one of the following:

- 1** this shop doesn't buy any items.
- 0** will pay 35% of the item's value.
- 1** will pay 32.5% of the item's value.
- 2** will pay 30% of the item's value.
- 3** will pay 27.5% of the items' value.
- 4** will pay 25% of the item's value.
- 5** will pay 22.5% of the item's value.
- 6** will pay 20% of the item's value.

Note that if you use the **begin_shop_mode()** command in a dialogue node, the response text in your dialogue node will appear after the party has finished shopping.

Boats and Horses

In Blades of Avernum, boats and horses are very similar, the only real difference being that boats travel on water while horses travel on land. Both boats and horses can be used by the party when travelling, and a boat or a horse can either belong to the party or to somebody else; the party can only use those boats and horses which it owns.

Boats and horses are controlled entirely through scripts. You can have a maximum of 30 boats and horses in a single scenario.

To define a boat or horse in your scenario, you first need to create a scenario script if you haven't already got one. A scenario script is a text file with the name of your main scenario script, with ".bas" replaced by ".txt". At the very least, your scenario script should contain the following:

```
beginscenarioscript;  
body;  
  
beginstate LOAD_SCEN_STATE;  
break;  
  
beginstate START_SCEN_STATE;  
break;  
  
beginstate START_STATE;  
break;
```

You define your boats and horses by placing appropriate commands into the START_SCEN_STATE node, like this:

```
beginstate START_SCEN_STATE;  
    create_horse(0, 3, 17, 14, 1);  
    create_boat(0, 7, 12, 36, 0);  
break;
```

Both the **create_horse()** and **create_boat()** commands take five parameters:

- The number of the horse or boat to define. Both horses and boats are numbered in the range 0 to 29.
- The number of the town where the boat or horse should first appear.
- The X and Y coordinates where you want the boat or horse to appear within the town.
- A flag indicating the ownership of the boat or horse; this should be 1 if the party initially own the boat/horse, and 0 if they don't.

Once you have defined your boat or horse, the party will be able to make use of it by walking on to the boat or horse. If the party don't own the horse or boat, they will first have to obtain ownership; you can do this within a script (for example a dialogue node) by using the **set_horse_property()** or **set_boat_property()** commands. Both of these commands take two parameters:

- The number of the horse or boat to change ownership of.
- 1 if the party now own the boat or horse, 0 if they don't.

You can make use of these commands to do quite sophisticated things within a dialogue script. For example, the following dialogue node will allow the party to purchase a boat from an NPC:

```
beginstalknode 3;
    state = 1;
    nextstate = -1;
    condition = get_flag(5, 3) == 0;
    question = "I want to purchase your boat.";
    text1 = "Smithers takes your money.  _Thanks._";
    text2 = "Smithers laughs.  _You can't afford it!_";
    code =
        if (coins_amount() < 500) {
            remove_string(1);
        } else {
            change_coins(-500);
            play_sound(15); // cash register sound.
            set_boat_property(0, 1);
            set_flag(5, 3, 1);
            remove_string(2);
        }
    break;
```

If the party can afford it, they are charged 500 coins to purchase the boat. If they can't afford it, an appropriate message is displayed instead. Finally, a Stuff Done Flag is used to ensure that the party can't purchase the same boat twice.

Quests

In your scenario you are likely to have a number of quests which the party can complete. Quests can be given to a party, and can be completed, when certain conditions are met.

Each quest in your scenario has to have a unique number in the range 0 to 99. To define a quest, you first need to create a scenario script if you haven't already got one; a scenario script is a text file with the same name as your main scenario file, except that the ".bas" is replaced by ".txt". At the very least, your scenario script should contain the following:

```
beginscenario script;  
body;  
  
beginstate LOAD_SCEN_STATE;  
break;  
  
beginstate START_SCEN_STATE;  
break;  
  
beginstate START_STATE;  
break;
```

You need to define your quests by placing **init_quest()** commands into the LOAD_SCEN_STATE node, like this:

```
beginstate LOAD_SCEN_STATE;  
    init_quest(1, "Kill Dragon.", "You've been asked to  
kill the dragon.");  
break;
```

The **init_quest()** command takes three parameters: the number of the quest, the quest's name (up to 29 characters long), and the quest's description (up to 199 characters long).

Once you've defined your quest, the next step is to give the quest to the party at the appropriate time, and to mark the quest as completed. For example:

```
begintalknode 12;
    state = 2;
    nextstate = -1;
    question = "I will kill the dragon for you.";
    text1 = "Excellent!";
    code =
        toggle_quest(1, 1);
    break;

...

begintalknode 26;
    state = 4;
    nextstate = -1;
    condition = get_flag(2, 3) == 1;
    question = "I killed the dragon!";
    text1 = "Thanks! Your quest has been fulfilled.";
    code =
        toggle_quest(1, 0);
    break;
```

As well as just marking the quest as being completed, you may want to do things such as give the party gold, a special item, or experience (or maybe all three!). To do this, the following commands may be useful:

```
change_coins(amount);
```

Adds the given amount to the party's coins. If 'amount' is negative, the party's wealth is reduced by the given amount.

```
change_spec_item(itemNum, howMany);
```

Gives 'howMany' copies of special item 'itemNum' to the party. If 'howMany' is negative, the items are removed from the party.

```
put_item_on_spot(x, y, itemNum);
```

Places an item of type 'itemNum' at the given position within the current town. This allows you to have a special item appear at a particular point once a quest has been completed.

```
reward_give(itemNum);
```

Attempts to give an item of type 'itemNum' to the party. If nobody can hold the item, it is placed at the party's feet (indoors only).

```
award_party_xp(amount, level);
```

Award each member of the party the given number of skill points, adjusted as if they were awarded while killing a monster of the given level.

Outdoor Encounters

An outdoor encounter is a group of creatures which the party can encounter while moving through an outdoor section. The encounter usually results in a battle, though it is possible to have an encounter consisting only of friendly characters who do things like display a message when the party meets them.

There are three basic types of outdoor encounters:

- Groups of monsters that spawn randomly and hunt down the player's party. These are called "Wandering Encounters."
- Encounters which are triggered by a script, usually a special encounter. We will call this type of encounter a "Programmed Encounter."
- Encounters which are spawned when the party enters the outdoor section, and do not respawn. These are called "Preset Encounters."

Wandering Encounters

Wandering encounters are groups of monsters which spawn randomly to chase the party through the outdoors. To create a wandering encounter, you first have to set the points in your outdoor section where the wandering monsters can spawn. Do this by clicking on the "Set Spawn Point" tool...



...and click on the four points in your outdoor section where you want the wandering monsters to spawn. Then you create the party of wandering monsters by selecting the **Outdoor Wandering Monsters** command from the **Outdoors** menu.

The dialog box which appears allows you to enter up to four different wandering monsters per outdoor section. Each group of monsters can have up to four different types of hostile creatures, and up to three types of friendly creatures who fight on the party's side (though it doesn't make sense to have friendly creatures tagging along with a group of wandering monsters).

When the group of monsters is wandering through the outdoors, the game will show the first hostile creature's icon for the group. Thus, you should use this slot for the most common, or main, type of creature in the group.

The “move type” field lets you specify how the wandering monsters travel after they have been spawned; the most common values for wandering monsters are:

- 0** Seeks the party.
- 2** Moves randomly
- 10** Seeks the party, but doesn’t travel more than 8 spaces from where it starts.
- 12** Moves randomly, but doesn’t travel more than 8 spaces from where it starts.

You can also use the “Random Move Chance” field to set a percentage chance that the wandering group of monsters will move in a completely random direction. This only makes sense if the monsters are seeking the party.

There are more advanced options as well, for example to set a Stuff Done Flag when the encounter is killed, or to run a script in response to the party fleeing the encounter, but these are beyond the scope of what you need for a typical group of wandering monsters.

Programmed Encounters

Programmed encounters never occur just by themselves; you need to activate them by including an appropriate command in a script. For example, you might have a special encounter in the outdoors which, depending on the state of the game, triggers a programmed encounter.

The BoA editor confusingly calls this type of encounter a “special” encounter, even though the same name is used for scripts which are called in response to the party stepping on a particular part of a town or outdoor area. To avoid this confusion, we will use the term “Programmed Encounter” instead.

You can have at most four programmed encounters in each outdoor section. You set up your programmed encounters by selecting the **Outdoor Special Encounters** command from the **Outdoors** menu. As with wandering encounters, you can enter up to four types of hostile creatures and three types of friendly creatures who fight on the party’s side.

For programmed encounters, you should turn on both the “Party can’t evade” and the “Encounter is forced” option, as you want the encounter to always happen when the programmed encounter is triggered. For programmed encounters, the following move types are supported:

- 0** Hunts down the party.
- 1** Doesn’t move.
- 2** Moves at random.
- 3** Follows the road which it is placed on.
- 4** Runs away from the party.
- 10** seeks the party, but won’t travel more than 8 spaces away from where it starts.
- 12** Moves at random, but won’t travel more than 8 spaces from where it starts.
- 13** Follows the road it is placed on, but won’t travel more than 8 spaces from where it starts.
- 14** Runs away from the party, but won’t travel more than 8 spaces away from where it starts.

You can also use the “Random Move Chance” field to set a percentage chance that the group of monsters will move in a completely random direction. This only makes sense if the monsters are not already moving at random.

The other options in the “Outdoor Special Encounter” dialog box may be useful for doing things like triggering scripts when the encounter is defeated, setting Stuff Done Flags, etc.

Once you have defined your programmed encounter, you need to add an instruction to your script to trigger the encounter. A common way to do this would be to set up a special encounter in response to the party entering a particular part of the outdoor area. Then, in your outdoor area script, you add a command to trigger the programmed encounter when appropriate, like this:

```
beginstate 28;  
    message_dialog("Wolves attack your party.", "");  
    create_out_spec_enc(0);  
break;
```

The **create_out_spec_enc()** command takes the number of the programmed encounter (in the range 0 to 3) and places that programmed encounter next to the party so that it is triggered. You can also use the **place_out_spec_enc()** command to place the special encounter at a particular point in the outdoors, like this:

```
place_out_spec_enc(0, 17, 4);
```

The first value is the number of the programmed encounter (0-3), while the remaining two numbers are the coordinates within the outdoor section on which that programmed encounter should be placed.

When you create a programmed encounter, you can optionally set up a custom terrain where the fight takes place. This can be useful, for example, if the party is trying to pass a fortified camp or barricade; when the party approach, the fight can take place on terrain which looks like a campground or barricaded area. To use a custom terrain for an encounter, you must first create a town in which the fight takes place. Make sure the town has a size of 48 x 48 spaces, and fill the town with whatever details would make a good setting for the fight.

When the game uses a custom town for an encounter, it places the party and any friendly creatures as close to the coordinate (22, 23) as possible, and it places the hostile creatures near coordinate (22, 16). Make sure you design your terrain appropriately, and leave enough open space for the party and all creatures to fit.

Once you have created your “fight town”, you next need to tell the computer to use it for the encounter. To do this, you need to create an “encounter is met” state in your outdoor area script, like this:

```
beginstate 53;  
    set_out_fight_town_loaded(17);  
break;
```

In this case, town 17 would be used as the setting for the fight. The number of this state should be entered into the “Script State when Met” field in the “Edit Outdoor Encounter” dialog, so that this state is called when the encounter is met.

Preset Encounters

These are encounters which only occur once or are not intended to be fought, for example an ambush which the party walks into, or a group of friendly guards who patrol an area. There can be a maximum of eight preset encounters in each outdoor section.

To create a preset encounter, use the **Outdoor Preset Encounters** command in the editor’s **Outdoors** menu. As with the other encounter types, you can enter up to four types of hostile creatures and three types of friendly creatures who fight on the party’s side.

If you want, you can use the “Party can’t evade” option to prevent the party from using their Nature Lore skill to avoid the encounter. You shouldn’t use the “Encounter is forced” option for this type of encounter.

For programmed encounters, the following move types are supported:

- 0** Hunts down the party.
- 1** Doesn't move.
- 2** Moves at random.
- 3** Follows the road which it is placed on.
- 4** Runs away from the party.
- 10** seeks the party, but won't travel more than 8 spaces away from where it starts.
- 12** Moves at random, but won't travel more than 8 spaces from where it starts.
- 13** Follows the road it is placed on, but won't travel more than 8 spaces from where it starts.
- 14** Runs away from the party, but won't travel more than 8 spaces away from where it starts.

You can also use the "Random Move Chance" field to set a percentage chance that the group will move in a completely random direction. This only makes sense if the group is not already moving at random.

The other options in the "Outdoor Preset Encounter" dialog box may be useful for doing things like triggering scripts when the encounter is defeated, setting Stuff Done Flags, etc.

By default, a preset encounter is automatically reset whenever the party enters the outdoor area. If you want to have a preset encounter only happen once, you can make use of a Stuff Done Flag to remember whether the party has defeated this encounter. To do this, enter the same SDF coordinates into both the "Stuff Done Flag To Eliminate the Encounter" and the "Stuff Done Flag Set to 1 when Defeated" fields. This will have the effect of setting the given SDF to 1 when the party defeats the encounter, and so prevent the encounter from appearing again once that SDF has been set.

Finally, if you want the encounter to not be hostile, for example to represent a friendly party of guards who patrol an area, you have to make use of some tricks to make this happen. Firstly, the encounter must have a "Hostile 1" creature specified, or else the encounter won't appear in the game, so place your guard (or whatever) creatures into the "Hostile 1" slot, even though the encounter is friendly. Then, you need to create a script to prevent the encounter from trying to fight the party: enter the number of an outdoor node into the "Script State when Met" field, and in your outdoor section script create a node which looks like this:

```
beginstate 34;  
    outdoor_enc_result(1);  
break;
```

The **outdoor_enc_result()** command takes a number which indicates how the encounter should behave:

- 0** The encounter should fight the party as normal.
- 1** The encounter is friendly, and will continue to exist.
- 2** The encounter is friendly. The group will disappear, but can reappear later if appropriate.
- 3** The encounter is friendly. The group will disappear, and will set the “Stuff Done Flag Set to 1 when Defeated” flag for this encounter to 1, so that the encounter will never reappear.

Obviously, you can include more sophisticated commands in your script, for example to only attack if the party has a particular special item.

Splitting and Reuniting the Party

At times in a scenario, you might want to have just one member of a party go ahead, separate from the others, to perform some task. This is typically done using a teleporter which only accepts one member of the party, or a narrow passage which only one party member can squeeze through. Once the task has been completed, the character rejoins the rest of the party and the game continues.

There are two limitations with splitting the party:

1. You can only split the party when in a town.
2. The party member who goes ahead must not be able to leave town until he or she rejoins the rest of the party.

Whatever mechanism you use in the game to split the party, you need to use a script command to actually split, and then rejoin, the party. To split the party, you typically use the **try_to_split_party()** command within a special encounter, like this:

```
beginstate 41;  
    try_to_split_party(17, 12, 1);  
break;
```

This command takes three parameters: the X and Y coordinates of the point where the separated character will be teleported to, and a flag indicating whether to play a teleport sound (1) or not (0) when the character is teleported.

When this command is executed, the player will be asked to select a character who will go ahead. That character will be moved to the given coordinate within the town, and if the “teleport sound” flag is 1, the teleportation sound will be played.

Once the party has been split, you use the **reunite_party()** command to reunite the party. You would typically use this command within a special encounter which the character steps on to rejoin the rest of the party. For example:

```
beginstate 42;  
    reunite_party();  
break;
```

The party will be reunited at the point where they were split up.

Quickfire

To place quickfire within a town or dungeon, you must make use of the **put_field_on_space()** command, like this:

```
put_field_on_space(x, y, 6);
```

where 'x' and 'y' are the coordinates of the point where you want the quickfire to be placed, and '6' is the code for quickfire.

Unfortunately, you cannot place quickfire in a town or dungeon without using a script command like this. You will most likely want to place the command inside a town script for your town or dungeon; if you don't yet have a town script for the town or dungeon, create a text file with the name of the town followed by ".txt". Then, in the "Town Details" dialog box, enter the name of the town script (minus the ".txt") into the "town script" field.

The town script should contain at least the following entries:

```
begin towns script;  
body;  
  
begin state INIT_STATE;  
break;  
  
begin state EXIT_STATE;  
break;  
  
begin state START_STATE;  
break;
```

You can, of course, add more commands to your town script if you wish.

If you want the quickfire to be in the town all the time (eg, behind a magic barrier), you can place this command in the INIT_STATE section of the town script, like this:

```
begin state INIT_STATE;  
    put_field_on_space(23, 7, 6);  
break;
```

You can, of course, be far more sophisticated than this. Because you have the full power of Avernuscript available to you, you can do things like have the quickfire only appear if a certain condition is met, like this:

```
begin state INIT_STATE;  
    if (get_flag(7, 3) == 0)  
        put_field_on_space(23, 7, 6);  
break;
```

You can have the quickfire appear when the party walk onto a particular space by making use of a special encounter. Simply create a special encounter node within your town script, like this:

```
beginstate 17;
    message_dialog("Quickfire explodes behind you!", "");
    put_field_on_space(23, 7, 6);
break;
```

Then select the “Create Special Encounter” tool in the BoA editor...



...and click on the top-left and bottom-right corners of the rectangular area where you want the quickfire to be triggered. Enter the number of the special node you created, and the quickfire will be triggered when the party enter that area.

If you want to have a lever which, when pulled, triggers the quickfire, you could place the lever in the dungeon, and set up a special encounter which triggers when the party walk onto the lever. Your special node could then look something like this:

```
beginstate 4;
    reset_dialog_preset_options(2);
    if (run_dialog(0) == 1) {
        message_dialog("Oh no! Quickfire!", "");
        put_field_on_space(23, 7, 6);
    }
break;
```

Note that the **reset_dialog_preset_options(2)** command sets up the dialog box to prompt the player to pull the lever. If they pull the lever, the quickfire will be unleashed.

Finally, you could use dialogue nodes so that, for example, when the party says something which angers a powerful mage, the dialog node’s “code” section includes commands to hide the mage and fill the room with quickfire, like this:

```
begintalknode 40;
    state = 8;
    nextstate = -1;
    question = "Spit on the mage's outstretched hand.";
    text1 = "The mage scowls. _How dare you!_";
    code =
        erase_char(32);
        put_field_on_space(23, 7, 6);
break;
```

Hills and Heights

One of the main things that distinguishes Blades of Avernum (and the entire Avernum series) from the earlier Exile series is the fact that the player sees the world in simulated 3D. That is, stairs, sloping floors and hills, etc, are all portrayed semi-realistically on screen. When designing your scenario, this means that you can give your towns, dungeons and outdoor areas that third dimension of height – but the ability to do so comes at a price: you have to use heights sensibly, or the party will disappear behind a hill or cliff and can't be seen by the player. Let's examine this problem first, before we look at how you can add heights to your scenario.

The Line of Sight Problem

There's nothing more frustrating for the player than having his or her party walk down some stairs and be hidden behind a cliff as they try and fight a bunch of monsters – which is why it's important that you, as a scenario designer, avoid the line of sight problem when using height in your scenarios.

- By the way, this problem occurs in the scenario “Valley of Dying Things” which comes with BoA, so even the best scenario designers get it wrong sometimes. That's no excuse, though, for not trying your best to avoid this problem.

Technically speaking, Blades of Avernum uses something called “Isometric Projection” to display the BoA world in three dimensions. Unlike true 3D graphics, where objects get smaller the further away they are, isometric projection means that the objects stay the same size no matter how close or far away from the view they are. This makes the game much easier for the people who wrote BoA, as they only had to create one image for each creature, terrain element, object, etc. From your point of view, though, isometric projection imposes a critical limitation on your scenario: the world can only ever be seen from one angle, and so you need to avoid using height in such a way that the player can't see important things such as monsters or items from this angle. This is what I mean by the “line of sight” problem.

Let's examine this in more detail. Here's a typical BoA scene:



Notice how the player's viewpoint is always going from south-east to north-west (that is, the bottom-right corner of the map towards the top-left corner). Objects that are north-west of the party appear above the party in the game window, while objects that are south-east appear below the party, and so on. Also notice how the southern and eastern slopes of the hill takes up a lot more screen space than the northern and western slopes – this is because of the way hills are displayed in the BoA game, by “pushing” the ground up towards the top of the screen.

Let's see how this can cause problems if your scenario is not designed correctly. Here's another BoA scene:



This clearly shows the line-of-sight problem. Because the party started out at a higher level, and has moved north and west towards a lower level, the player's line of sight is blocked. You can just make out the monster attacking the front-most fighter, but if the fighter was to move further down the sloping passageway, he would be completely hidden.

Because of the way the game displays heights, the player will always be able to see things better if they are placed south, south-east, or east of a hill or other raised area, compared with things placed north, north-west or west of a raised area. In the picture above, the downward-sloping passageway the fighter is travelling along is north of the higher raised area at the bottom of the screen, which is why the fighter becomes obscured.

Of course, this is only a problem if the lower area is adjacent or nearly adjacent to the higher area. If the downward-sloping passageway in the above dungeon had been placed a space or two further north, the player would have been able to see it without a problem, as there would have been enough room for the game to display the lower area without the higher area obscuring it.

Now that you know the pitfalls, let's look at how you can go about adding hills and other height differences to your scenario.

Automatic Hills

The BoA editor contains an “automatic hills” feature which makes it easy to add things like hills and sloping passageways to your scenario. To use it, click on the “switch mode” button...



...or press the spacebar until the BoA editor is in “edit heights” mode. In this mode, a number will be displayed in the top-right corner of each space in the town or outdoor section being edited, like this:

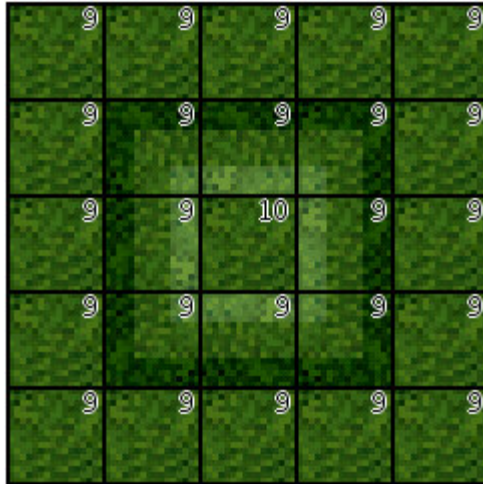
| | | | | |
|---|---|---|---|---|
| 7 | 7 | 8 | 8 | 9 |
| 7 | 7 | 8 | 8 | 8 |
| 7 | 7 | 7 | 8 | 8 |
| 6 | 6 | 7 | 7 | 8 |

These numbers represent the height of each space, in the range 0 to 99. For towns, the default height of the ground is set to 5, while for outdoor sections the default height is 9; bigger numbers represent higher ground, while smaller numbers represent lower ground. To increase the height of a space, simply click on it while in “edit height” mode; to decrease the height of a space, hold down the command (“apple”) key while clicking on that space.

When you are in the “edit height” mode, the bottom-right corner of the BoA editor window will show you things like the current drawing mode (“HEIGHT”), the coordinates of the space shown in the centre of the window, and whether automatic hills are turned on or off. To turn on automatic hills, click on the “automatic hills” button:



With this option turned on, the computer will change the terrain of your ground to show hills and slopes as you change the height of a space. For example, if you are in “edit heights” mode with automatic hills turned on and you click on a space to increase the height of that space, the computer will automatically place terrain around that space to make it a hill, like this:



The hills terrain (ie, sloping floors and hillsides) will automatically be added or removed as you change the height of each cell.

Automatic hill mode can be very convenient if all you want to do is create a few low hills to add visual interest to a town or outdoor area. Unfortunately there are some “gotchas” with using automatic hill mode which make it far less useful than you might think:

- When you use automatic hill mode, the computer will helpfully “smooth out” your terrain, destroying any sudden changes in height you may have created, such as cliffs or vertically-edged pits. This “smoothing out” happens *everywhere* in the current town or outdoor section, not just in the places where you are changing height, so if you have any sudden changes in height at all in your town or outdoor section, then you should avoid using automatic hill mode altogether.
- If two adjacent spaces in your town or outdoor section differ in height by more than one step, the computer will be unable to create the hill terrain that you would expect.
- The computer will automatically select the type of slope based on what it thinks you want the terrain to look like. Thus, you might end up with barren hills poking out of grassy flatlands, or cave floor where you would expect a bare dirt path to be.

For these reasons, you may want to learn how to create hills manually, as this gives you far more control as well as avoiding the above pitfalls. The procedure for creating hills by hand will be described in the next section.

Manual Hills

When creating hills by hand, you can do everything that the automatic hills feature will do for you, as well as having complete control over how and where the hills and sloping passageways, etc, will appear.

To create hills by hand, make sure that you have turned off the “automatic hills” feature. If this feature is on, click on the “automatic hills” button...



...to turn it off again. You can then use the various hill terrain elements to create your hills by hand. Here are what some of the hill terrain elements look like:

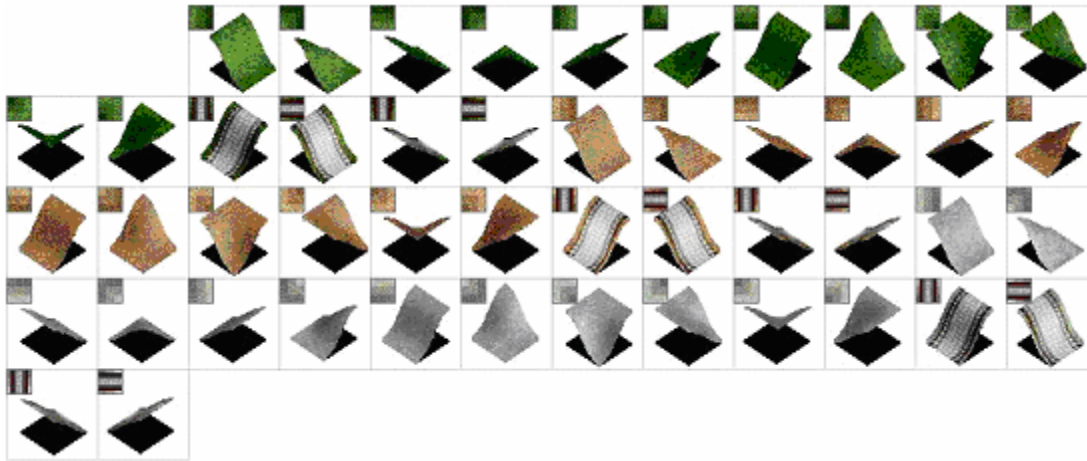


To create hills and other slopes by hand is a two-step process. Firstly, you need to decide on how the terrain should look, and place the appropriate terrain elements to make the hills and slopes look the way you want. Then, once you’ve placed the terrain elements, you then adjust the height of the spaces within your town or outdoor section so that the terrain elements line up correctly.

Let’s start by examining the types of terrain elements you can use to create hills, slopes, sloping passageways, etc. There are three basic types of slopes you can create:

- Grass-covered slopes.
- Bare dirt slopes.
- Cave floor or rocky slopes.

For each of these types of slopes, there are terrain elements which correspond to the given terrain sloping in various directions, as well as a copy of the terrain with roads running over it:



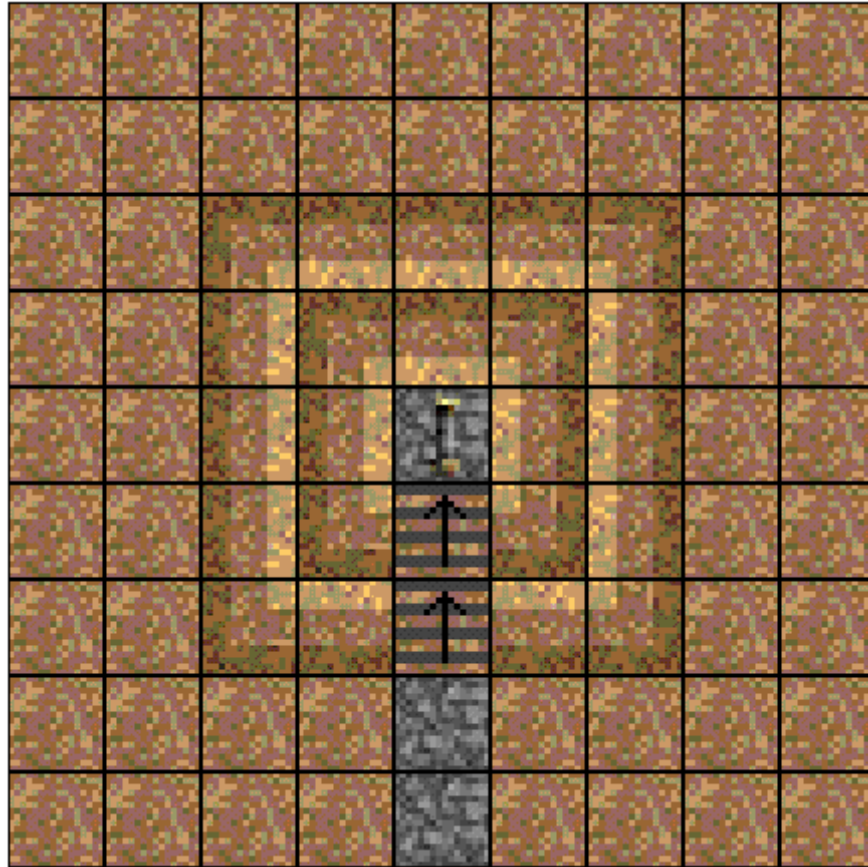
As well as actual slopes, you can also use the staircase terrain elements as part of a slope:



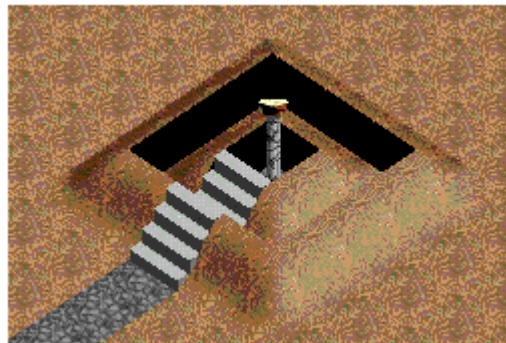
These can be useful, for example, where a path or walkway goes up or down a slope.

Notice how, in the BoA editor's icons for each of these terrain elements, the bottom end of the slope is darker than the top end. The icons are a bit confusing at first, but you'll soon get the hang of what they mean.

Once you have placed the various terrain elements, your town or outdoor section should look something like the following within the BoA editor:



If you were to load this into the BoA game itself, your terrain would now look something like this:



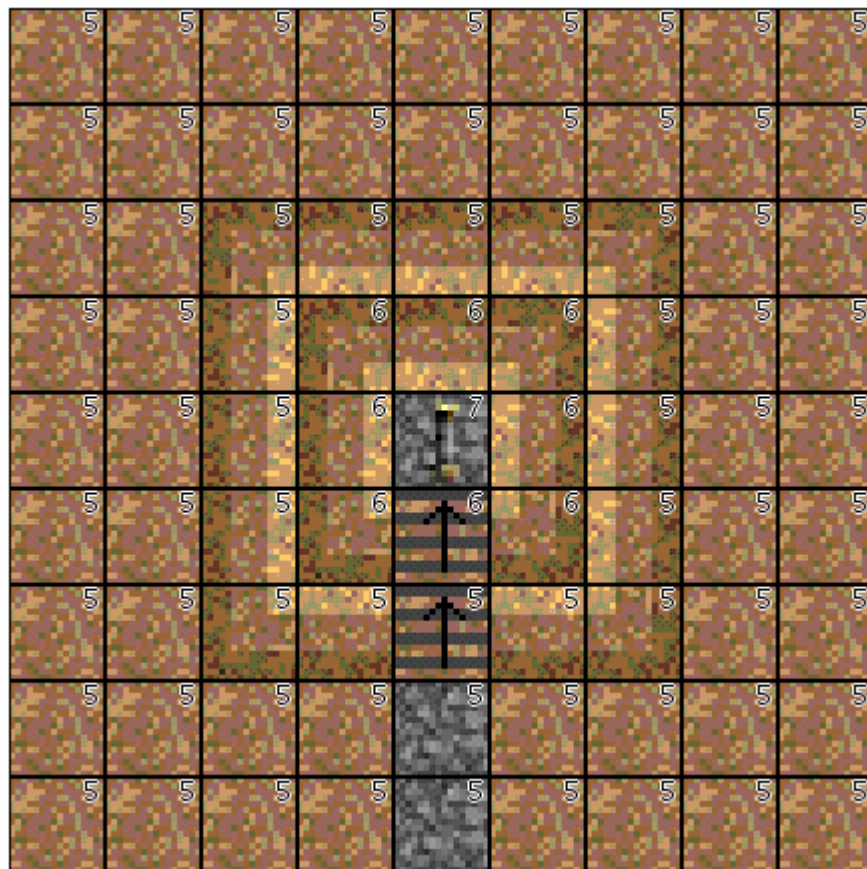
Notice how the terrain itself is correct, but the heights are all wrong. You now need to adjust the heights so that the terrain elements line up to make the hills and other slopes you want. To achieve this, use the “edit heights” mode of the BoA editor to set the height of each space as appropriate. As well as clicking on a space to increase the height, and holding down the command (“apple”) key while clicking on a space to decrease the height, you can use the “set height” button...



...to set the height of any rectangular area to the same value.

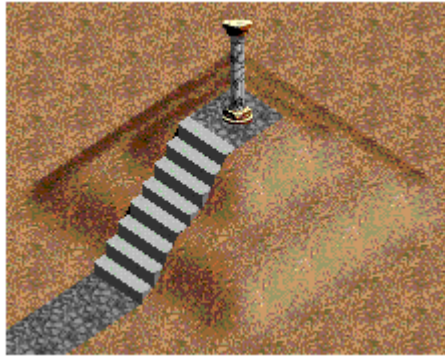
When setting the heights, make sure that adjacent spaces do not vary in height by more than 1 – unless, that is, you want to have a cliff between the two adjacent cells.

In the above example, you would want to set the heights to something like the following:



There's an important lesson to be learned from this picture: the terrain elements used to represent slopes and staircases go up from the specified height. For example, the first flight of steps has a height of 5, matching the height of the path which leads up to it. The second flight of steps has a height of 6, so that it will line up with the top of the first flight of steps.

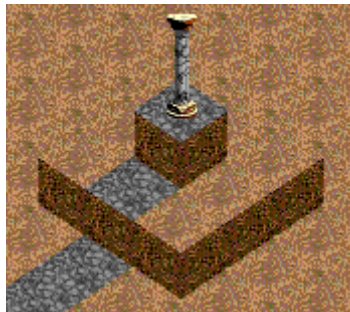
If you were to load the above terrain into the BoA game, it would look like this:



You should always load your town or outdoor section into the BoA game to see how it looks – when you draw hills by hand, it's easy to miss a few heights, but it's also easy to fix mistakes as soon as you spot them. With practice, you'll find that creating hills and other slopes by hand is almost as easy as creating them using the “automatic hills” feature, but with a lot more flexibility and none of the drawbacks.

Cliffs

Whenever there is a difference in height between two adjacent spaces, the computer will draw a cliff between those two spaces unless there is a slope terrain element which covers up the cliff. So, for example, if you removed the slopes in the above terrain, the result would look like this:



If you are creating cliffs within a town or dungeon, you can tell the computer what the cliff should look like by entering an appropriate number into the “Cliff Sheet” field within the “Town Details” dialog box. The various cliff sheet numbers are listed within the dialog box, so simply enter an appropriate sheet number and the cliff will be displayed appropriately.

For outdoor sections, your options are much more limited, as the computer selects an appropriate cliff style based on whether the outdoor section is underground or on the surface. You can change this by clicking on the “On Surface” checkbox within the “Outdoor Details” dialog box, but that’s the only sort of customisation you can do for cliffs in outdoor sections.

Finally, remember that you can have cliffs of any height you desire. Of course, if the party attempt to jump off a cliff, the higher the cliff the greater the damage done to the party.

Appendices

Editor Shortcuts



Move around the current town or outdoor section.



Move as far as possible in the given direction.



Move the selected object (item, creature or terrain script).

Delete

Activate the “Delete Object” tool.

Shift-R

Paint a rectangle with the current terrain.

Shift-S

Activate the “Select/Edit Object” tool.












































































Spacebar





















































Toggle between floor, terrain and height modes.









































































Default Items













The following pages show all the standard items found in the Blades of Avernum game. The items can be selected using the **I1**, **I2**, **I3**, **I4**, and **I5** menus in the BoA editor, where the items are listed in numerical order.












































































For more information about the characteristics of each item, please refer to the “corescendata2.txt” file, which you will find in your **Blades of Avernum Files** folder. The different codes used to define the item properties in the `corescendata2.txt` file are explained in the BoA Editor documentation, in the chapter on custom item types.




































































| | | |
|---|--|--|
|  1 - Copper Coins |  26 - Leather Armor |  51 - Iron Short Sword |
|  2 - Silver Coins |  27 - Fine Leather Armor |  52 - Steel Short Sword |
|  3 - Gold Coins |  28 - Drakeskin Armor |  53 - Blessed Short Sword |
|  4 - Bread |  29 - Cursed Studded Armor |  54 - Cursed Longsword |
|  5 - Mushrooms |  30 - Poor Studded Armor |  55 - Bronze Longsword |
|  6 - Greens |  31 - Iron Studded Armor |  56 - Iron Longsword |
|  7 - Steak |  32 - Steel Studded Armor |  57 - Steel Longsword |
|  8 - Dried Meat |  33 - Blessed Studded Armor |  58 - Blessed Longsword |
|  9 - Weird Meat |  34 - Cursed Chain Mail |  59 - Cursed Greatsword |
|  10 - Lizard Haunch |  35 - Poor Chain Mail |  60 - Bronze Greatsword |
|  11 - Fish |  36 - Iron Chain Mail |  61 - Iron Greatsword |
|  12 - Deli Sandwich |  37 - Steel Chain Mail |  62 - Steel Greatsword |
|  13 - White Tunic |  38 - Mithril Chain Mail |  63 - Blessed Greatsword |
|  14 - Red Tunic |  39 - Cursed Plate Mail |  64 - Cursed Spear |
|  15 - Green Tunic |  40 - Bronze Plate Mail |  65 - Crude Spear |
|  16 - Cloak |  41 - Iron Plate Mail |  66 - Iron Spear |
|  17 - Cloak |  42 - Steel Plate Mail |  67 - Steel Spear |
|  18 - Pants |  43 - Blessed Plate Mail |  68 - Blessed Spear |
|  19 - Pants |  44 - Cursed Dagger |  69 - Cursed Pike |
|  20 - Shirt |  45 - Crude Dagger |  70 - Crude Pike |
|  21 - Shirt |  46 - Iron Dagger |  71 - Iron Pike |
|  22 - Robe |  47 - Steel Dagger |  72 - Steel Pike |
|  23 - Dress |  48 - Blessed Dagger |  73 - Blessed Pike |
|  24 - Cursed Leather Armor |  49 - Cursed Short Sword |  74 - Cursed Halberd |
|  25 - Poor Leather Armor |  50 - Crude Short Sword |  75 - Bronze Halberd |

| | | |
|---|---|---|
|  76 - Iron Halberd |  101 - Iron Arrows |  126 - Wooden Small Shield |
|  77 - Steel Halberd |  102 - Steel Arrows |  127 - Iron Small Shield |
|  78 - Blessed Halberd |  103 - Blessed Arrows |  128 - Steel Small Shield |
|  79 - Stick |  104 - Cursed Bolts |  129 - Blessed Small Shield |
|  80 - Ceremonial Dagger |  105 - Crude Bolts |  130 - Cursed Large Shield |
|  81 - Kitchen Knife |  106 - Iron Bolts |  131 - Wooden Large Shield |
|  82 - Staff |  107 - Steel Bolts |  132 - Iron Large Shield |
|  83 - Hammer |  108 - Blessed Bolts |  133 - Steel Large Shield |
|  84 - Cursed Javelin |  109 - Rock |  134 - Blessed Large Shield |
|  85 - Crude Javelin |  110 - Cursed Sandals |  135 - Cursed Leather Helmet |
|  86 - Iron Javelin |  111 - Sandals |  136 - Poor Leather Helmet |
|  87 - Steel Javelin |  112 - Drakeskin Sandals |  137 - Leather Helmet |
|  88 - Blessed Javelin |  113 - Cursed Boots |  138 - Fine Leather Helmet |
|  89 - Cursed Bow |  114 - Boots |  139 - Wyrmskin Helmet |
|  90 - Crude Bow |  115 - Drakeskin Boots |  140 - Cursed Bronze Helmet |
|  91 - Ash Bow |  116 - Cursed Cloak |  141 - Bronze Helmet |
|  92 - Yew Bow |  117 - Blessed Cloak |  142 - Iron Helmet |
|  93 - Blessed Bow |  118 - Cursed Robe |  143 - Steel Helmet |
|  94 - Cursed Crossbow |  119 - Blessed Robe |  144 - Blessed Steel Helmet |
|  95 - Crude Crossbow |  120 - Cursed Buckler |  145 - Cursed Gloves |
|  96 - Ash Crossbow |  121 - Wooden Buckler |  146 - Gloves |
|  97 - Yew Crossbow |  122 - Iron Buckler |  147 - Drakeskin Gloves |
|  98 - Blessed Crossbow |  123 - Steel Buckler |  148 - Gauntlets |
|  99 - Cursed Arrows |  124 - Blessed Buckler |  149 - Bowl |
|  100 - Crude Arrows |  125 - Cursed Small Shield |  150 - Bowl |

| | | |
|---|--|---|
|  151 - Pitcher |  176 - Magic Lockpicks |  201 - Bar of Iron |
|  152 - Pot |  177 - First Aid Kit |  202 - Bar of Silver |
|  153 - Pot |  178 - Fine First Aid Kit |  203 - Amber |
|  154 - Bolt of Cloth |  179 - Poor Fur |  204 - Emerald |
|  155 - Bolt of Cloth |  180 - Nice Fur |  205 - Ruby |
|  156 - Bolt of Cloth |  181 - Beautiful Skin |  206 - Silver Ring |
|  157 - Cloak Clasp |  182 - Pillow |  207 - Gold Ring |
|  158 - Pincers |  183 - Pewter Mug |  208 - Jeweled Ring |
|  159 - Tongs |  184 - Mortar and Pestle |  209 - Silver Bracelet |
|  160 - Spearhead |  185 - Papyrus Sheet |  210 - Gold Bracelet |
|  161 - Bucket |  186 - Dice |  211 - Silver Necklace |
|  162 - Bucket of Water |  187 - Scalpel |  212 - Gold Necklace |
|  163 - Flagon |  188 - Needle |  213 - Torc |
|  164 - Horn |  189 - Thread |  214 - Healing Herbs |
|  165 - Jug |  190 - Trowel |  215 - Spiritual Herbs |
|  166 - Jug of Cheap Wine |  191 - Skull |  216 - Energetic Herbs |
|  167 - Golden Goblet |  192 - Bones |  217 - Graymold |
|  168 - Plate |  193 - Whip |  218 - Toadstools |
|  169 - Plate |  194 - Pen and Ink |  219 - Mandrake |
|  170 - Candle |  195 - Earrings |  220 - Healing Potion |
|  171 - Torch |  196 - Trash |  221 - Curing Potion |
|  172 - Lamp |  197 - Trash |  222 - Strength Potion |
|  173 - Spoon |  198 - Trash |  223 - Energy Potion |
|  174 - Lockpicks |  199 - Bar of Tin |  224 - Poison Potion |
|  175 - Fine Lockpicks |  200 - Bar of Lead |  225 - Haste Potion |

| | | |
|---|--|---|
|  226 - Invulnerability Potion |  251 - Scroll - Lightning Spray |  276 - Armor Ring |
|  227 - Resistance Potion |  252 - Scroll - Dispel Barrier |  277 - Nimble Band |
|  228 - Healing Elixir |  253 - Scroll - Summon Aid |  278 - Warrior's Ring |
|  229 - Curing Elixir |  254 - Scroll - Fireblast |  279 - Firewalker Band |
|  230 - Strength Elixir |  255 - Scroll - Repel Spirit |  280 - Quicksilver Ring |
|  231 - Energy Elixir |  256 - Scroll - Summon Shade |  281 - Archer's Ring |
|  232 - Poison Elixir |  257 - Scroll - Control Foes |  282 - Wand of Bolts |
|  233 - Haste Elixir |  258 - Scroll - Spray Acid |  283 - Wand of Ice |
|  234 - Invulnerability Elixir |  259 - Scroll - Force Cage |  284 - Wand of Slowing |
|  235 - Resistance Elixir |  260 - Scroll - Capture Mind |  285 - Wand of Lightning |
|  236 - Knowledge Brew |  261 - Scroll - Divine Host |  286 - Wand of Fire |
|  237 - Restoration Brew |  262 - Scroll - Bitter Loss |  287 - Wand of Acid |
|  238 - Protection Brew |  263 - Scroll - Withering |  288 - Wand of Forcecage |
|  239 - Heroic Brew |  264 - Ring of Vulnerability |  289 - Wand of Absorption |
|  240 - Graymold Salve |  265 - Ring of Idiocy |  290 - Rod of Arcana |
|  241 - Balm of Life |  266 - Clumsy Ring |  291 - Rod of Beast Call |
|  242 - Searing Balm |  267 - Ring of Exposure |  292 - Rod of Major Call |
|  243 - Draining Brew |  268 - Ring of Grief |  293 - Rod of Illusions |
|  244 - Scroll - Bolt of Fire |  269 - Magnetic Ring |  294 - Strength Bracelet |
|  245 - Scroll - Call Beast |  270 - Shield Ring |  295 - Dexterity Bracelet |
|  246 - Scroll - Arcane Blow |  271 - Ring of Health |  296 - Intelligence Bracelet |
|  247 - Scroll - Slow |  272 - Ring of Skill |  297 - Mage's Bracelet |
|  248 - Scroll - Ice Lances |  273 - Resistance Ring |  298 - Priest's Bracelet |
|  249 - Scroll - Create Illusions |  274 - Assassin's Band |  299 - Warrior's Bracelet |
|  250 - Scroll - Far Sight |  275 - Fletcher's Ring |  300 - Lucky Bracelet |

| | | |
|--|--|---|
|  301 - Monkey Bracelet |  326 - Piercing Crystal |  351 - Alien Blade |
|  302 - Chill Charm |  327 - Unshackling Crystal |  352 - Acid Arrows |
|  303 - Warmth Charm |  328 - Shattering Crystal |  353 - Bolts of Life |
|  304 - Health Charm |  329 - Shard of Release |  354 - Arrows of Light |
|  305 - Shielding Charm |  330 - Mithril Woven Robe |  355 - Acid Bolts |
|  306 - Basic Charm |  331 - Archer's Cloak |  356 - Ruby Helm |
|  307 - War Charm |  332 - Dragonskin Cloak |  357 - Emerald Helm |
|  308 - Freedom Charm |  333 - Pants of Power |  358 - Helm of Speed |
|  309 - Crystal Charm |  334 - Pants of Sorrow |  359 - Runed Helm |
|  310 - Knowledge Charm |  335 - Robe of the Magi |  360 - Nimble Gloves |
|  311 - Sticky Charm |  336 - Radiant Robe |  361 - Ogrish Gauntlets |
|  312 - Harm Charm |  337 - Rogue's Leather |  362 - Giantish Gauntlets |
|  313 - Cursed Sling |  338 - Mauling Leather |  363 - Glue Gauntlets |
|  314 - Leather Sling |  339 - Shadow Leather |  364 - Aspskin Gloves |
|  315 - Blessed Sling |  340 - Icy Chain Mail |  365 - Micah's Gloves |
|  316 - Broom |  341 - Polished Plate Mail |  366 - Dancing Shoes |
|  317 - Mop |  342 - Assassin's Dagger |  367 - Boots of Apollo |
|  318 - Shovel |  343 - Diamond Dagger |  368 - Slippers of Speed |
|  319 - Pick |  344 - Flaming Sword |  369 - Wyrmsbane |
|  320 - Broken Sword |  345 - Icy Longsword |  370 - Slith Spear |
|  321 - Leather Bracer |  346 - Demonslayer |  371 - Fine Slith Spear |
|  322 - Drakeskin Bracer |  347 - Ghoulbane |  372 - Scissors |
|  323 - Bronze Bracer |  348 - Giantslayer |  373 - Pot |
|  324 - Iron Bracer |  349 - Jade Halberd |  374 - Glass |
|  325 - Blessed Bracer |  350 - Obsidian Spear |  375 - Rolling Pin |

| | | |
|--|---|---|
|  376 - Towel |  401 - Basket |  426 - Shield of Klud |
|  377 - Hatchet |  402 - Clay Pot |  427 - Martyr's Shield |
| 378 - <i>Unused</i> |  403 - Teapot |  428 - Runeshield |
|  379 - Cursed Waveblade |  404 - Mug |  429 - Shield of Kron |
|  380 - Waveblade |  405 - Teacup |  430 - Ring of Warmth |
|  381 - Fine Waveblade |  406 - Boardgame |  431 - Icy Ring |
|  382 - Cursed Razordisk |  407 - Potted Plant |  432 - Ring of Speed |
|  383 - Razordisk |  408 - Hourglass |  433 - Ring of Will |
|  384 - Fine Razordisk |  409 - Carved Stone Spiral |  434 - Aescal's Ring |
|  385 - Vahnatai Cloak |  410 - Lovely Stone Spiral |  435 - Dust of Shadows |
|  386 - Crystal |  411 - Saw |  436 - Dust of Choking |
|  387 - Fine Crystal |  412 - Plank |  437 - Cleansing Powder |
|  388 - Gold Ore |  413 - Stone Blocks |  438 - Basic Powder |
|  389 - Silver Ore |  414 - Bottle of Ale |  439 - Powder of Lethe |
|  390 - Rough Diamond |  415 - Bottle of Wine |  440 - Helm of Alertness |
|  391 - Rope |  416 - Unicorn Horn |  441 - Archer's Bow |
|  392 - Fine Meal |  417 - Incense |  442 - Mithril Blade |
|  393 - Cheap Wine |  418 - Gremlin Wine |  443 - Plate Boots |
| 394 - <i>Unused</i> |  419 - Sack of Meal |  444 - Titanium Boots |
|  395 - Bliss Elixir |  420 - Bar of Gold | |
|  396 - Rogue's Elixir |  421 - Crystal Shield | |
|  397 - Pears |  422 - Iceshield | |
|  398 - Apples |  423 - Shield of Khar | |
|  399 - Cake |  424 - Shield of Klin | |
|  400 - Meat on Spit |  425 - Lifeshield | |

Default Creatures

The following pages show the various creatures (both non-player characters and monsters) built into the BoA game. The first list groups the creatures by level, while the second list shows a picture of each creature and lists the creatures in ID number order.

The various creatures are colour-coded to show their default attitude to the party:

Friendly creatures have black text.

Neutral creatures have blue text.

Hostile creatures have red text.

Creatures by Level

Level 1 Creatures

Beggar (#3)
Vahnatai Child (#148)
Cat (#80)
Chicken (#189)
Child (#86)
Dog (#186)
Lizard (#67)
Goblin (#21)

Level 2 Creatures

Townsmen (#74)
Townsmen (#76)
Townsmen (#78)
Townswoman (#75)
Townswoman (#77)
Townswoman (#79)
Cave cow (#68)
Sheep (#190)
Amber Slime (#195)
Bat (#56)
Brigand (#7)
Brigand Archer (#9)
Emerald Slime (#197)
Giant rat (#53)
Kobold (#129)
Ochre Slime (#196)
Thug (#73)

Level 3 Creatures

Merchant (#1)
Merchant (#2)
Cow (#188)
Mauve Slime (#198)
Serpent (#62)
Skeleton (#42)
Unicorn (#231)
Wolf (#191)

Level 4 Creatures

Acolyte (#10)
Apprentice mage (#14)
Official (#81)
Chitrach Larva (#160)
Giant lizard (#51)
Goblin Fighter (#134)
Spore Beast (#171)
Zombie (#43)

Level 5 Creatures

Official (#18)
Cave slime (#50)
Evil acolyte (#88)
Ghoul (#44)
Giant spider (#69)
Nephari (#27)
Nephari (#22)
Nephari archer (#24)
Shade (#120)

Level 6 Creatures

Cockroach (#200)
Hordling (#145)
Large Roach (#201)
Mung Slime (#233)
Nephari archer (#29)
Nephari shaman (#25)
Nephari warrior (#23)
Slime Zombie (#232)
Spirit (#46)
Vapor rat (#54)

Level 7 Creatures

Soldier (#6)
Ghast (#45)
Ghost (#48)
Giant Roach (#202)
Lava bat (#57)
Nephari shaman (#30)
Nephari warrior (#28)
Nephari chieftain (#26)
Viscous Goo (#143)
Worg (#192)

Level 8 Creatures

Archer (#8)
Asp (#63)
Basilisk (#84)
Bear (#193)
Fire lizard (#52)
Mung rat (#72)
Mung Roach (#203)
Wight (#47)

Level 9 Creatures

Big Roach (#205)
Baby Hydra (#164)
Imp (#59)
Nephar chieftain (#31)
Ogre (#32)
Troglodyte (#206)

Level 10 Creatures

Captain (#5)
Pixie (#65)
Chitrach (#159)
Greater Shade (#121)
Gremlin (#130)
Guardian Roach (#204)
Poison fungus (#55)
Salamander (#166)
Slith (#34)
Witch (#17)

Level 11 Creatures

Talking spider (#70)
Aranea (#71)
Ice Lizard (#141)
Ice Slime (#142)
Wyrmling (#183)

Level 12 Creatures

Monk (#133)
Priest (#11)
Good Witch (#82)
Black shade (#91)
Evil priest (#13)
Living statue (#66)
Null Bug (#169)
Rogue Archer (#124)
Rogue Mage (#125)
Rogue Warrior (#123)
Royal Slime (#199)
Slith mage (#37)
Slith priest (#36)
Troglodyte Shaman (#208)
Troglodyte Warrior (#207)

Level 13 Creatures

Mage (#15)
Slith warrior (#35)
Ursag (#229)

Level 14 Creatures

Dryad (#226)
Assassin (#19)
Ruby Skeleton (#136)

Level 15 Creatures

Bladesman (#131)
Hill Giant (#212)
Hydra (#161)
Troglodyte Defender (#210)

Level 16 Creatures

Ice Pudding (#168)
Quickghast (#137)
Troglodyte Khazi (#209)

Level 18 Creatures

Sorcerer (#106)
Empire Archer (#132)
Evil high priest (#89)
Guardian (#99)
Hill Giant Fighter (#213)
Hill Giant Shaman (#214)
Slith high priest (#85)
Spectre (#138)
Vahnnavoi (#140)

Level 20 Creatures

Vahnatai (#146)
Vahnatai (#147)
Deva (#165)
Cave giant (#39)
Centaur (#102)
Elder aranea (#83)
Foul Rat (#176)
Gazer (#93)
Giant shaman (#40)
Hraithe (#139)
Ogre mage (#33)
Shambler (#144)
Slith chief (#38)
Ur-Basilisk (#90)
Vengeful Shade (#122)

Level 21 Creatures

Doomguard (#158)
Fire Golem (#217)
Golem of Blades (#216)
Ice Golem (#218)

Level 22 Creatures

Unicorn (#94)
Flame Hydra (#162)
Hill Giant Chief (#215)
Ice Drake (#167)
Mung demon (#95)
Spawned Fungus (#185)

Level 23 Creatures

Rabid bat (#98)

Level 24 Creatures

Eye beast (#64)

Ice Hydra (#163)

Jeweled Golem (#219)

Slith Archmage (#182)

Vampire (#87)

Level 25 Creatures

Elite Soldier (#20)

High priest (#12)

Wizard (#16)

Demon (#60)

Drake (#58)

Giant chief (#41)

Gorgon (#230)

Infiltrator (#184)

Mad Monk (#187)

Slith Avatar (#100)

Spine Beast (#228)

Troglodyte Lord (#211)

Level 26 Creatures

Demon Golem (#220)

Level 28 Creatures

Giant Slug (#170)

Naga (#173)

Skeletal Warrior (#127)

Level 30 Creatures

Guard (#4)

Centaur Chief (#103)

Dark Wyrn (#225)

Efreet (#174)

Fetid Zombie (#128)

Fierce Shade (#177)

Granite Golem (#92)

Mind Crystal (#222)

Power Crystal (#221)

Rakshasa (#172)

Royal guard (#96)

Royal mage (#97)

Level 33 Creatures

Vahnatai Shaper (#151)

Vahnatai Shaper (#152)

Vahnatai Warrior (#149)

Vahnatai Warrior (#150)

Mutant Lizard (#126)

Level 34 Creatures

Alien Beast (#223)

Level 35 Creatures

Augmented Giant (#135)

Level 36 Creatures

Vahnatai Keeper (#153)

Vahnatai Keeper (#154)

Drake Lord (#227)

Level 40 Creatures

Soul Crystal (#157)

Vahnatai Blade (#155)

Triad mage (#105)

Divine Shade (#178)

Empire Dervish (#175)

Haakai (#61)

Lich (#49)

Pack Leader (#224)

Level 45 Creatures


Vahnatai Lord (#156)

Level 50 Creatures

Erika (#104)

Dragon (#101)

Creatures by ID Number

| | | | | |
|---|---|---|---|--|
| 1  Merchant | 2  Merchant | 3  Beggar | 4  Guard | 5  Captain |
| 6  Soldier | 7  Brigand | 8  Archer | 9  Brigand Archer | 10  Acolyte |
| 11  Priest | 12  High priest | 13  Evil priest | 14  Apprentice mage | 15  Mage |
| 16  Wizard | 17  Witch | 18  Official | 19  Assassin | 20  Elite Soldier |
| 21  Goblin | 22  Nephil | 23  Nephil warrior | 24  Nephil archer | 25  Nephil shaman |
| 26  Nephil chieftain | 27  Nephar | 28  Nephar warrior | 29  Nephar archer | 30  Nephar shaman |
| 31  Nephar chieftain | 32  Ogre | 33  Ogre mage | 34  Slith | 35  Slith warrior |
| 36  Slith priest | 37  Slith mage | 38  Slith chief | 39  Cave giant | 40  Giant shaman |



41 L25

Giant chief



42 L3

Skeleton



43 L4

Zombie



44 L5

Ghoul



45 L7

Ghast



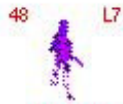
46 L6

Spirit



47 L8

Wight



48 L7

Ghost



49 L40

Lich



50 L5

Cave slime



51 L4

Giant lizard



52 L8

Fire lizard



53 L2

Giant rat



54 L8

Vapor rat



55 L10

Poison fungus



56 L2

Bat



57 L7

Lava bat



58 L25

Drake



59 L9

Imp



60 L25

Demon



61 L40

Haakai



62 L3

Serpent



63 L8

Asp



64 L24

Eye beast



65 L10

Pixie



66 L12

Living statue



67 L1

Lizard



68 L2

Cave cow



69 L5

Giant spider



70 L11

Talking spider



71 L11

Aranea



72 L8

Mung rat



73 L2

Thug



74 L2

Townsmen



75 L2

Townswoman



76 L2

Townsmen



77 L2

Townswoman



78 L2

Townsmen



79 L2

Townswoman



80 L1

Cat

81 L4



Official

82 L12



Good Witch

83 L20



Elder aranea

84 L8



Basilisk

85 L18



Slith high priest

86 L1



Child

87 L24



Vampire

88 L5



Evil acolyte

89 L18



Evil high priest

90 L29



Ur-Basilisk

91 L12

Black Shade

92 L30



Granite Golem

93 L20



Gazer

94 L22



Unicorn

95 L22



Mung demon

96 L30



Royal guard

97 L30



Royal mage

98 L23



Rabid bat

99 L18

Guardian

100 L25



Slith Avatar

101 L50



Dragon

102 L20



Centaur

103 L30



Centaur Chief

104 L50



Erika

105 L40



Triad mage

106 L18



Sorcerer

120 L5



Shade

121 L10



Greater Shade

122 L20



Vengeful Shade

123 L12



Rogue Warrior

124 L12



Rogue Archer

125 L12



Rogue Mage

126 L33



Mutant Lizard

127 L28



Skeletal Warrior

128 L30



Fetid Zombie

129 L2



Kobold

130 L10



Gremlin

131 L15



Bladesman

132 L18



Empire Archer

133 L12



Monk

134 L4



Goblin Fighter

135 L35



Augmented Giant

136 L14



Ruby Skeleton

137 L16



Quickghast

138 L18



Spectre

139 L20



Hraithe

140 L18



Vahnnavoi

141 L11



Ice Lizard

142 L11



Ice Slime

143 L7



Viscous Goo

144 L20



Shambler

145 L6



Hordling

146 L20



Vahnatai

147 L20



Vahnatai

148 L1



Vahnatai Child

149 L33



Vahnatai Warrior

150 L33



Vahnatai Warrior

151 L33



Vahnatai Shaper

152 L33



Vahnatai Shaper

153 L36



Vahnatai Keeper

154 L36



Vahnatai Keeper

155 L40



Vahnatai Blade

156 L45



Vahnatai Lord

157 L40



Soul Crystal

158 L21



Doomguard

159 L10



Chitrach

160 L4



Chitrach Larva

161 L15



Hydra

162 L22



Flame Hydra

163 L24



Ice hydra

164 L9



Baby Hydra

165 L20



Deva

166 L10



Salamander

167 L22



Ice Drake

168 L16



Ice Pudding

169 L12



Null Bug

170 L28



Giant Slug

| | | | | |
|--|---|--|--|--|
| 171 L4  Spore Beast | 172 L30  Rakshasa | 173 L28  Naga | 174 L30  Efreet | 175 L40  Empire Dervish |
| 176 L20  Foul Rat | 177 L30  Fierce Shade | 178 L40  Divine Shade | | |
| | 182 L24  Slith Archmage | 183 L11  Wyrn | 184 L25  Infiltrator | 185 L22  Spawned Fungus |
| 186 L1  Dog | 187 L25  Mad Monk | 188 L3  Cow | 189 L1  Chicken | 190 L2  Sheep |
| 191 L3  Wolf | 192 L7  Worg | 193 L8  Bear | | 195 L2  Amber Slime |
| 196 L2  Ochre Slime | 197 L2  Emerald Slime | 198 L3  Mauve Slime | 199 L12  Royal Slime | 200 L6  Cockroach |
| 201 L6  Large Roach | 202 L7  Giant Roach | 203 L8  Mung Roach | 204 L10  Guardian Roach | 205 L9  Big Roach |
| 206 L9  Troglydte | 207 L12  Troglydte Warrior | 208 L12  Troglydte Shaman | 209 L16  Troglydte Khazi | 210 L15  Troglydte Defender |

211 L25



Troglodyte Lord

212 L15



Hill Giant

213 L18



Hill Giant Fighter

214 L18



Hill Giant Shaman

215 L22



Hill Giant Chief

216 L21



Golem of Blades

217 L21



Fire Golem

218 L21



Ice Golem

219 L24



Jeweled Golem

220 L26



Demon Golem

221 L30



Power Crystal

222 L30



Mind Crystal

223 L34



Alien Beast

224 L40



Pack Leader

225 L30



Dark Wyrn

226 L14



Dryad

227 L36



Drake Lord

228 L25



Spine Beast

229 L13



Ursag

230 L25



Gorgon

231 L3



Unicorn

232 L6



Slime Zombie

233 L6

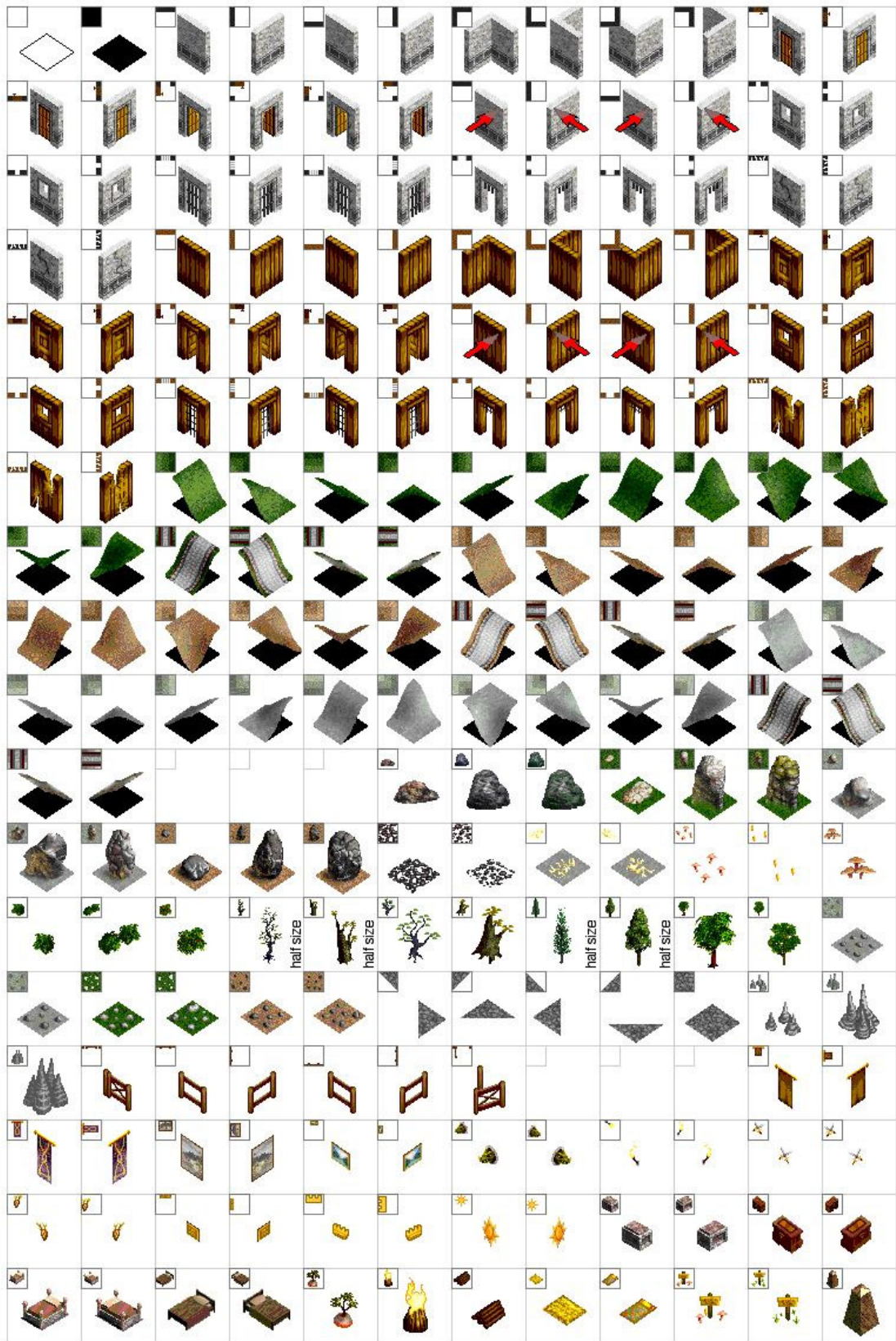


Mung Slime

Default Terrain Graphics

The following two pages show how the various terrain icons in the editor's terrain elements list translate into actual graphics within the BoA game. The BoA editor includes a simple description of each terrain icon, but the icons are still confusing and rather overwhelming to a new user. Finding something like a bookcase or secret door is extremely difficult until you know what all these icons mean – which is where these charts come in. The terrain graphics are listed in the same order as they appear in the editor; the small box in the corner shows the icon as it appears in the editor, while the larger box shows the same piece of terrain as it appears in the game itself.

Please note that these charts only show the *default* terrain graphics; by changing the values entered into the “appearance settings” section of the “Edit Town Details” dialog box, you can have different sets of graphics used for things such as walls and cliffs. However, this chart should still be useful as a starting point for building a town, even if you do use customised graphics.





Default Floor Graphics

The following table shows the default types of floor graphics available in Blades of Avernum.

