Ultima: Quest of the Avatar Save State Hacking Guide

by jdratlif Updated to v1.2 on Mar 21, 2006

```
--------------------------------------------------------------------------------
| Ultima: Quest of the Avatar
| Save State Hacking Guide
| by John Ratliff
|
| The most recent version of this guide can always be found at
| http://games.technoplaza.net/hack4u/sram-hacking.txt
|
| Copyright (C) 2004-2005 emuWorks (http://games.technoplaza.net/)
|   Permission is granted to copy, distribute and/or modify this document
|   under the terms of the GNU Free Documentation License, Version 1.2
|   or any later version published by the Free Software Foundation;
|   with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
|   Texts.  A copy of the license can be found at
|   http://www.gnu.org/licenses/fdl.html
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
| Table of Contents
--------------------------------------------------------------------------------

  1.0 Introduction
  2.0 Revision History
  3.0 Credits & Thanks
  4.0 Copyright
  5.0 SRAM Offsets
    5.1  Phases of the Moon
    5.2  The Hero's Name
    5.3  Virtues
    5.4  Party Members
    5.5  Stones & Runes
    5.6  Magic
    5.7  Herbs, Gold, & Tools
    5.8  Inventory
    5.9  Character Stats
    5.10 Starting Location
    5.11 Pirate Ships & The Balloon
    5.12 The Whirlpool
    5.13 Treasure Offsets
    5.14 Miscellaneous Offsets
  6.0 The Sanity Algorithm
  7.0 Contact Information

--------------------------------------------------------------------------------
| 1.0 Introduction
--------------------------------------------------------------------------------

  This document is a guide to hacking the save state (SRAM or battery backed
  RAM) of Ultima: Quest of the Avatar for the Nintendo Entertainment System.

  Ultima: Quest of the Avatar was released for the NES in 1990. The copyrights
  are attributed to Origin Systems (creators of the Ultima series) in 1985 for
```

the original version, and to FCI and Pony Canyon in 1990 for the NES version.

This guide refers to the NES version released in the US in 1990. I do not know if it was released in Japan or Europe. If it was, this information may or may not apply to those games. Since I only owned the US version, I can only comment on that version.

This guide is not specific to any one NES emulator. It modifies the SRAM (aka battery backed RAM, non-volatile RAM, etc.) that the game used internally to save the game on the real cartridge. Because of this, these methods should work on any emulator that supports SRAM. Any decent emulator should do this. A short list includes FCE Ultra, Nester, Nestopia, Nesten. I have never used Jnes, Fakenes, or VirtuaNES, but I would guess they do this also as they are highly regarded.

I have only tested this information with FCE Ultra, which is the emulator I use. This is primarily because FCE Ultra is one of two current emulators that runs in Windows and includes a nice debugger which is very useful for finding the information here.

Modifying the information directly (by hand with a hex editor for instance) is difficult because of a complicated sanity algorithm Ultima uses to check the integrity of the SRAM. Because the original battery in the cartridge can only last so long, it was necessary to develop an algorithm they were sure would confirm the data in the SRAM was legit. Unfortunately, it is not a simple algorithm to compute by hand. Fortunately, I've written a nice program that will do it for you. You may be reading this guide as part of that program, but if not, you can find it at http://games.technoplaza.net/hack4u/. It is available for Windows, Linux, Mac OS X, and FreeBSD.

The sanity algorithm is described in section 6.0.

2.0 Revision History

Version 1.2 - March 21, 2006
- minor typo corrections
- updated the Ultima alphabet table
- added information on joined party members
- added information on starting location
- added information on the balloon location
- added information on the pirate ship locations
- added information on the whirlpool location
- added information on treasure offsets

Version 1.1 - August 3, 2005
- minor corrections (typos mostly)

Version 1.0 - December 4, 2004
- initial version of this guide

3.0 Credits & Thanks

This guide would not be possible without the work of several people I feel should be awarded much accolates for their efforts.

- Tony Hedstrom
  Without Tony, I would have never discovered the sanity algorithm used by the game. I had originally given up on it until I talked with him about

it.

- Xodnizel
  Creator of FCE Ultra. Without FCE Ultra's debugger, it is quite likely I wouldn't have been able to find most of the offsets that lead to this guide and certainly not the sanity algorithm. It's a shame Xodnizel doesn't intend to continue FCE Ultra's development.

- Julian Smart, Vadim Zeitlin, Robert Roebling, and all the wxWidgets authors
  Without wxWidgets, hack4u would have been written in Java. While it would have been just as nice IMO, many people dislike Java and it wouldn't be as useful to people.

  In addition, Vadim and Julian have been invaluable resources in learning the wxWidgets nuances and I wouldn't have gotten very far without their guidance.

- sp, bbitmaster, DahrkDaiz, and Vystrix Nexoth
  Creators of FCEUXD SP. The PPU viewer was very helpful in learning the remainder of the Ultima character alphabet.

| 4.0 Copyrights

It really disgusts me when authors of documents like this say you can't post their stuff, or that you can do so only with their prior approval. Maybe this is solely because they want to make sure their guide is up-to-date in all the distributed locations, but I suspect far more evil intent.

Since I can't do anything about their copyrights, I'll just have to lead by example.

Basically, it is free documentation in much the same way software under the GNU General Public License is free software. You can modify it, redistribute it, sell it, publish it, etc.

| 5.0 SRAM Offsets

Finally, we get to the good stuff.

Ultima: Quest of the Avatar for the NES included, like many games for the NES, an SRAM (aka battery backed RAM).

This SRAM was a portion of writable memory inside the cartridge that when the NES was off was saved by a battery also contained inside the cartridge. This is similar the way a PC saves it's CMOS information today. That's the reason your PC's clock still works when your computer is off.

The SRAM was an area of memory $2000 bytes long. All numbers in this guide

that begin with a $ are hex numbers, i.e. numbers in the hexadecimal number system, base 16. Decimal numbers, like we use in practice, are base 10. I don't plan to include a decimal->hex tutorial here, so if you don't understand hex, you'll probably need to find one. There are many on the web.

Fortunately, the parts of the SRAM we care about are in three small $200 byte sections, and 9 other bytes contained elsewhere.

The NES accessed the SRAM at the CPU addresses $6000-$7FFF. When I refer to the CPU address, it will always be in this range. The SRAM address with be in the range $0000-$1FFF. $0000 SRAM = $6000 CPU.

The SRAM was used by Ultima to save three different games. Each game used $200 (512) bytes of memory. The three SRAM addresses start at $1A00, $1C00, and $1E00. They end at $1BFF, $1DFF, and $1FFF. The nine other bytes are used to sanity check the SRAM, and will be discussed later.

Since each game is identical in storage location, I will use relative SRAM offsets. In other words, if I refer to address $27 in game 1, I really mean SRAM address $1A27. Just add the start address + the relative offset to get the SRAM address. You can then add the starting CPU address if you want the actual CPU address, but this is probably not useful unless you're just curious.

Here are the relative offsets in the SRAM that I know about. There are many bytes unaccounted for. Some may be unused, and some may just be uninteresting for purposes of save state hacking. There will be a more detailed explanation after I present the list.

$00	Sanity Byte
$01	Moon Phases
$02-$06	Hero's Name
$0B	Characters Who've Agreed to Join the Party
$0C-$13	Virtues
$14	Avatarhood
$15-$18	Party Members
$19	Stones
$1A	Runes
$1B-$1E	Magic
$1F-$26	Herbs
$27-$28	Gold
$29	Torches
$2A	Gems
$2B	Key
$2C	Oil
$2D	Sextant
$2E	Scale
$2F	Flute
$30	Candle
$31	Book
$32	Bell
$33	Wheel
$34	Horn
$35	Skull
$36	Key of Truth
$37	Key of Courage
$38	Key of Love
$39-$3E	Mage Inventory
$3F-$44	Bard Inventory
$45-$4A	Fighter Inventory

$4B-$50 Druid Inventory
$51-$56 Tinker Inventory
$57-$5C Paladin Inventory
$5D-$62 Ranger Inventory
$63-$68 Shepherd Inventory
$69-$70 Levels
$71-$80 Current HP
$81-$90 Max HP
$91-$98 Current MP
$99-$A0 Max MP
$A1-$A8 Strength
$A9-$B0 Intelligence
$B1-$B8 Dexterity
$B9-$C8 Experience
$D2-DD   Found Treasure
$ED   Capture Pirate Ships
$EE   Next Pirate Ship to Overwrite
$F0-$F1 Balloon Location
$F2-$F3 Whirlpool Location
$F4-$FB Pirate Ship Location(s)
$107   Starting Location

These are all the ones I know about, or cared enough to make a note of anyway.

I haven't noted the spells you can learn in Spells Unlimited. I thought it was enough just to have the spells.

5.1 Phases of the Moon

There are two moons in the world of Ultima: Trammel and Felucca. Trammel is the one on the left, while Felucca is on the right. Respectively, they determine where the Moongate is, and where it will lead.

There are $18 (24 = 8 cities * 3 destinations each) possible phases. The relative offset $01 determines which of these it is.

Here are all the possible values the offset $01 can have and their meanings.

$00 Moonglow - Moonglow (Black Stone)
$01   - Britain
$02   - Jhelom

$03 Britain - Yew
$04   - Minoc
$05   - Trinsic

$06 Jhelom - Skara Brae
$07   - Magincia
$08   - Moonglow

$09 Yew - Britain
$0A   - Jhelom
$0B   - Yew

$0C Minoc - Minoc (Shrine of Spirituality)
$0D   - Trinsic
$0E   - Skara Brae

```
  $0F Trinsic - Magincia
  $10         - Moonglow
  $11         - Britain

  $12 Skara Brae - Jhelom
  $13            - Yew
  $14            - Minoc

  $15 Magincia - Trinsic
  $16          - Skara Brae
  $17          - Magincia
```

This value is not that useful since the moon phases change often enough it is usually easier to wait for a specific phase in-game than to alter the state.

5.2 The Hero's Name

In Ultima, like most RPG's, you have a Hero and several supporting characters. The Hero is the only character that can have a name selected by the player. The game determines that the person who leads the party is the Hero. Other than that, the Hero is just like any of the other character classes. If your hero is the Paladin class, then the game's built-in Paladin (Dupre) will not join you. This is because the game stores information according to class, not according to individual player.

Well, enough about the hero, how do you change his (or her) name.

The offsets $02-$06 (5 characters) will do this. You can also use $07 for a sixth character (like the name Mariah for the built-in Mage), but the game will only allow you to give the character 6. Some number of offsets past $07 can also be used for the hero, but this will mess up the status screen, so I don't recommend more than 6, 5 if you want to stay within the game's limits, but the only real reason I see for this is that there's no space between the last character in the name and the status info in battle. The mage Mariah has this problem, too, so it doesn't seem to hurt the game any.

Ultima's alphabet for names is simple. I will list all the symbols I know, but some may not be valid for names. The legitimate alphabet for character names includes the letters A-Z and the lowercase a-z, the numbers 0-9, the dash -, the exclamation point !, a space (represented by an underscore _), and the NUL character which signifies termination of a name with less than 5 characters. My save state editor program hack4u limits you to the game provided character alphabet.

Legitimate character alphabet:

```
A-Z = $91-$AA
a-z = $D1-$EA
-   = $D0
!   = $90
_   = $BC     (SPC in the game becomes an underscore)
NUL = $00     (this signifies a name with less than 5 (or 6) characters).
```

Other possible symbols:

```
Circle (small)                = $BF
/                             = $C0
.                             = $C1
```

```
,                                = $C2
Dot                              = $C3
"                                = $C4
Degree mark (circle)             = $C5
,                                = $C6
Key symbol                       = $C7
Star                             = $C8
+ (Plus Symbol)                  = $C9
Ankh Symbol                      = $CA
x (not the same as alphabet x)   = $CB
:                                = $CC
;                                = $CD
```

Status Screen Alphabet:

These are the capital letters used on the status screen to indicate class, MP, HP, and status ailments. These letters are distinct from the name slphabet.

```
N                     = $63
E                     = $64
S (Shepherd)          = $65
W                     = $66
G                     = $67
P (Paladin or Poison) = $68
D (Druid)             = $69
M (Mage or MP)        = $6A
B (Bard)              = $6B
F (Fighter)           = $6C
H (HP)                = $6D
T (Tinker)            = $6E
R (Ranger)            = $6F
```

I'm not sure what N, E, W, and G are used for. The other letters may have additional uses I'm unaware of.

I always name my hero either Dominic or Liam, depending upon how many characters I'm allowed. Since we are restricted to 5, Liam it is.

Liam is spelled $9C $D9 $D1 $DD $00. That fills the offsets $02-$06.

5.3 Virtues

The virtue system in Ultima introduced with Quest of the Avatar was extremely popular and unique.

There are eight virtues: Honesty, Compassion, Valor, Justice, Sacrifice, Honor, Spirituality, and Humility. They are always numbered from 0-7.

The game represents everything in refernece to these eight virtues. The cities Moonglow, Britain, Jhelom, Yew, Minoc, Trinsic, Skara Brae, and Maginicia. The cities are always numbered 0-7, too. The characters Mage, Bard, Fighter, Druid, Tinker, Paladin, Ranger, and Shepherd are also represented 0-7. All these values are tied together. Moonglow is the city of Honesty and Mages.

The virtues are tracked all throughout the game in the offsets $0C-$13. $0C = Honesty, $13 = Humility. Here is the full list:

$0C Honesty
$0D Compassion
$0E Valor
$0F Justice
$10 Sacrifice
$11 Honor
$12 Spirituality
$13 Humility

I'm not 100% what the exact values represent, but the higher the number, the better (more moral) your character is.

I believe the values range between $00-$64 (0-100). $64 means you have partial avatarhood, $63 is the best you can be without having partial avatarhood, $32 (50) is where you start, and so on.

However, avatarhood is measured two ways (probably for simplicitiy in the program). Offset $14 is also used to measure partial avatarhood in a virtue.

Offset $14 is what is known as a bit list or flagset, etc. It has many names. Basically, $14 is a single byte, which has 8 bits of data. Since there are eight virtues, this is perfect. We can turn a bit on (1) to represent you have partial avatarhood in this virtue, and off (0) to represent you do not have partial avatarhood in this virtue.

Bits are represented from right to left. So if we have a list of bits 10011010, the bit 7 is on (1), and bit 0 is off (0). Knowing that virtues are represented 0-7, we know that Honesty is bit 0, and Humility is bit 7.

If all the bits are on, you are the avatar. But if you virtue values don't correspond, you will lose your avatar status very easily. My program, hack4u, will keep both values in sync automatically.

Let's say the game has just started. You have $32 (50) in offsets $0C-$13 and $0 (no partial avatarhoods) in $14. To give yourself partial avatarhood in honesty, we can change $0C to $64 and $14 to $1. If after that, we want to get partial avatarhood in Honor, we can set offset $11 to $64 and offset $14 to $21.

Here is how I got the value for $14.

Starting value = $1 (%00000001 in binary -- binary numbers start with % signs) We want to turn on bit 5 (honor = 5). So we binary OR the starting value with (1 LSH 5 = %00100000). Then we get %00100001 which is $21 in hex. If you don't understand binary math operations, you should look for a tutorial, or just use my program. That's even easier.

5.4 Party Members

Ultima IV can support four characters in your party as one time. The characters are one of the following: Mage, Bard, Fighter, Druid, Tinker, Paladin, Ranger, and Shepherd. You can also have no one in your party at a certain location, but to avoid confusing the game, it's best if the empty slots are at the end.

If you read the section on Virtues, you know that the characters, like the virtues are always numbered 0-7 (Mage - Shepherd). But as a special case where there can be no one, 0 becomes no one. So each number is shifted up by 1.

Mage = 1, Bard = 2, ..., Shepherd = 8.

The first member is the hero, so $15 not just the leader, but is also the hero. You cannot rearrange the hero's location. If you put the hero's character in another slot, he (or she) will stop being the hero and will lose their hero abilities such as being able to equip Exotic Armor or Sword of Paradise if they are the Avatar.

You can also have more than one of the same character in your party. However, they are all tracked with the same stats. This means if one character dies, they all die. But on the plus side, if one gains experience, they all gain experience. Other than this quirk, I didn't notice any adverse affects in the game, but that doesn't mean there aren't any. There are generally some issues to be expected when the game is forced to use invalid data it never expected was possible.

Since members in your party are not tracked by the same value as people who have agreed to join you, if you put characters who have not joined your party in your formation, you will see them in their hometowns. You can even ask them to join you as if they were not already in your party. If your party is not full and they agree to join you, you will have a duplicate in your party. If your party is full, they will go to the hostel in Castle Britannia. But since that member is already in your party, they won't really be there.

In addition, if you part with people at the hostel who didn't join you, they will not show up in the hostel until you've asked them to join you in their hometown. However, if you split with them, they will appear in the hostel until you leave the room and can rejoin your party immediately.

Just set the values to 0-8 (0 = no one, 1-8 = Mage - Shepherd)

$15 First Member (Hero)
$16 Second Member
$17 Third Member
$18 Fourth Member

As for members who have agreed to join your party, this is tracked by another flagset variable, $0B. Since no one is not an option here, we go back to the standard values 0-7 Mage - Shepherd. This affects which characters will appear in their hometown or in the hostel if they are not in your party. Assuming you had picked the Mage as your hero, and asked Iolo (Bard), Dupre (Paladin), and Jaana (Druid) to join you, $0B would have a value of $2B.

5.5 Stones & Runes

As with character's, virtues, and towns, there are eight stones and eight runes. They all correspond to a particular virtue.

Blue Stone of Honesty
Yellow Stone of Compassion
Red Stone of Valor
Green Stone of Justice
Orange Stone of Sacrifice
Purple Stone of Honor
White Stone of Spirituality
Black Stone of Humility

The stones are at offset $19 while the runes are at offset $1A. Just like

partial avatarhood, they are represented by a bitlist. You can read more about bitlists in the Virtues section 5.3.

5.6 Magic

Magic in the game is tracked using four bitlists at $1B, $1C, $1D, and $1E. They can each hold eight spells, but only some values are used. You can read more about bitlists in the Virtues section 5.3.

Offset $1B represents the following spells:

- Bit 0 Light
- Bit 1 Missile
- Bit 2 Awaken
- Bit 3 Cure
- Bit 4 Wind
- Bit 5 Heal
- Bit 6 Fire
- Bit 7 Exit

Offset $1C

- Bit 0 Dispel
- Bit 1 View
- Bit 2 Protect
- Bit 3 Ice
- Bit 4 Blink
- Bit 5 Energy
- Bit 6 Quick
- Bit 7 (Unused)

Offset $1D

- Bit 0 Sleep
- Bit 1 Reflect
- Bit 2 Negate
- Bit 3 (Unused)
- Bit 4 Destroy
- Bit 5 Jinx
- Bit 6 Squish
- Bit 7 Gate

Offset $1E

- Bit 0 Tremor
- Bit 1 Life
- Bit 2 (Unused)
- Bit 3 Defeat
- Bit 4 (Unused)
- Bit 5 (Unused)
- Bit 6 (Unused)
- Bit 7 (Unused)

If you think I've missed a spell or this information is incorrect, please contact me.

5.7 Herbs, Gold, & Tools

The number of herbs you can have is represented in the offsets $1F-$26. There are eight herbs: Ash, Ginseng, Garlic, Silkweb, Moss, Pearl, Fungus, and Manroot. Ash = $1F, Manroot = $26. The rest are in-between, respectively.

Just change the number to how much you want. $00 = 0, $63 = 99. This is the most the game would normally let you have, though I haven't seen any adverse affect if you have more, say $FF = 255.

Gold is one of the more difficult values because it's one of the few values that is represented by a word (2 bytes = 16 bits). Anytime you have a value greater than a single byte (usually), you start needing a way to store such a number in memory. There are two primary ways of doing this. They are known as little-endian, and big-endian. The NES (MOS 6502) and the PC (Intel x86) are little-endian. Macs (MC68000 and PPC) and Suns (Sparc) are big-endian.

What is endian? It's how you store a multi-byte number. Do you store the most significant part first (like humans do normally, 8001), which is big-endian, or do you store the least significant part first, which is little-endian. Though I would personally prefer big-endian since it looks more correct for us, the NES used little-endian, and we can't do anything about that now.

So, if the NES has a word value (2 bytes) starting at $26, it might read something like $9001 ($27 = $90, $28 = $01). Now, $0190 is 400, the amount of gold you start with. So, to look correct for us, we need to swap $27 and $28. So, if we want 9999 gold, which is $270F hex, we want to put $0F in $27 and $27 in $28.

Complexities like this are just another reason to use my hack4u program.

Tools come in two formats, have/have not, and quantity. The quantity tools are torches, gems, and oil. You can have 0 (none) up to 99 (max). You might be able to have 255 like Herbs, but we haven't tested it.

So, for the three quantity tools, just change the offset to the number you want, $00 - $63 (or possibly $FF).

$29 = Torches, $2A = Gems, and $2C = Oil

All the other tools are have/have not tools. You either have them, or you don't. If you don't have them, there's a $0 value in the offset. If you have them, there is a $1.

The have/have not tools are $2B, and $2D-$38. You can look them up in section 5.0.

5.8 Inventory

In Ultima, there are eight characters. Each ones inventory is tracked separately. They can have 6 items (weapons, bows, and armors) and they can have one of each kind equipped.

So there are six offsets for each character, and they start at $39. Refer to section 5.0 for the full list for each character.

Here are the possible values.

$00 Nothing
$01 Sling
$02 Bow
$03 X-Bow
$04 +1 Bow
$05 Dagger (Has nyone ever heard of this? I haven't)
$06 Staff
$07 Club
$08 Axe
$09 Sword
$0A +1 Sword
$0B +2 Sword
$0C +1 Axe
$0D Wand
$0E +2 Axe (Axe of Legend)
$0F Sword of Paradise
$10 Cloth
$11 Leather
$12 Chain
$13 Ring
$14 +1 Cloth
$15 Plate
$16 +1 Chain
$17 +1 Plate
$18 Robe
$19 Exotic Armor

That will give your character the item. If you want it equipped, add $80 to the value. $81 = Equipped Sling. $99 = Equipped Exotic Armor.

5.9 Character Stats

Character stats include current level, current/max HP, current/max MP, strength, intelligence, dexterity and experience.

Of those values, current/max HP and experience are word values. See section 5.7 on Gold for more information on the implications of word values as opposed to single byte values.

For each range of offset, they apply to the characters in sequence Mage, Bard, Fighter, Druid, Tinker, Paladin, Ranger, and Shepherd. This means the level offset for the Mage is $69, for Bard $6A, for Fighter, $6B, etc.

I'm not entirely sure, but I believe valid HP values are between 200 and 800, while valid experience values are between 0 and 9999. All the rest of the values range between 0 and 99. Going outside these ranges may yield unexpected results.

5.10 Starting Location

The starting location value determines in which town the player starts out. There are eight inns in the game, one for each city of virtue except Magincia, and the village of Vesper.

Offset $107
  Moonglow   = $05

```
    Britain    = $06
    Jhelom     = $07
    Yew        = $08
    Minoc      = $09
    Trinsic    = $0A
    Skara Brae = $0B
    Vesper     = $0E
```

Any other value puts you in Britain's inn.

5.11 Pirate Ships & The Balloon

In Ultima, you can find a balloon to fly in, and steal ships from pirates. The game has one balloon, and keeps track of up to 4 stolen pirate ships you've comandeered. For each ship after 4 you take, one will vanish. You may lose all your ships if you travel by moongate or use the Gate spell. I'm not sure exactly when the game erases your ships during gate travel.

The locations of the balloon and the pirate ships are stored in two bytes using the game's internal latitude and longitude coordinates.

```
Balloon Longitude = $F0
Balloon Latitude  = $F1
```

The Balloon will go back to its starting location anytime you exit the dungeon Hyloth from the Britannia entrance. The starting location is at (243,233).

The four pirate ship locations are stored as follows:

```
Ship 1 = ($F4,$F5) (Longitude,Latitude)
Ship 2 = ($F6,$F7)
Ship 3 = ($F8,$F9)
Ship 4 = ($FA,$FB)
```

Because there are four potential comandeered pirate ships, there are some extra offsets for pirate ships. To have one of the ships, the offset $ED must have one of it's lower 4 bits set. For ship 1 to exist, bit 0 must be on, ship 2 is bit 1, and so on for the four possible ships.

The offset $EE keeps track of the next ship. In other words, if you already have ship 1, then the next ship you steal should be ship 2. This number varies between 0 and 3 for the 4 possible pirate ships.

5.12 The Whirlpool

There are three ways to get to the village Cove in the game. One is the balloon. Another is by ship without a whirlpool (yes, there are places you can catch a pirate ship and sail to Cove sans whirlpool). Finally, there is taking your ship into the Whirlpool.

Before today, I had never seen the whirlpool, though I've heard of it. I've always used one of the other two methods for getting to Cove.

I'm not entirely sure how the whirlpool works. It moves, but it has a starting location that may or may not change. Since I've never seen it before, I don't know if it changes or not. I assume that it does because it seems stupid to

save a value in SRAM that never changes.

The starting location of the whirlpool is at ($F2,$F3) (Longitude, Latitude). It moves around this spot, but doesn't seem to stray very far. I'm not sure how useful it is to change the location or if there are any side effects. I didn't notice any when I tested it. But I only tried it out for a short time.

5.13 Treasure Offsets

Treasure offsets determine whether or not a found item will be there when you go to look for it. For example, if you've already found the rune of honor, then it won't be there if you research the field in Trinsic. Nearly every treasure has a single bit that tracks whether it will be there or not. The only exceptions I have found are the three keys.

```
Rune of Honesty      = $D4 bit 4
Rune of Compassion   = $D4 bit 5
Rune of Valor        = $D4 bit 6
Rune of Justice      = $D4 bit 7
Rune of Sacrifice    = $D5 bit 0
Rune of Honor        = $D5 bit 1
Rune of Spirituality = $D5 bit 2
Rune of Humility     = $D5 bit 3

Blue Stone           = $D6 bit 7
Yellow Stone         = $D7 bit 0
Red Stone            = $D7 bit 1
Green Stone          = $D7 bit 2
Orange Stone         = $D7 bit 3
Purple Stone         = $D7 bit 4
White Stone          = $D7 bit 5
Black Stone          = $D7 bit 6

Horn                 = $D6 bit 3
Flute                = $D5 bit 5
Scale                = $D5 bit 4
Candle of Love       = $D5 bit 7
Book of Truth        = $D6 bit 0
Bell of Courage      = $D6 bit 1
Mondain's Skull      = $D6 bit 4
Sword of Paradise    = $DD bit 7
Exotic Armor         = $DE bit 0
Robe                 = $D5 bit 6

Gave Scale to Zircon = $D2 bit 3
Axe of Legend        = $D2 bit 5
```

5.14 Miscellaneous Offsets

These offsets don't seem to fit in anywhere else. I'm not sure how useful they are, but here they are anyway.

Dungeon Keeper's Wall Smashed:

For each of the 6 dungeons that have stones (all except Hyloth), there is a guardian who keeps the stone behind a wall. If you answer the question

correctly, the wall crumbles. These offsets track whether the wall is still present or not.

```
Deceit (Honesty)     = $DD bit 1
Despise (Compassion) = $DD bit 2
Destard (Valor)      = $DD bit 3
Wrong (Justice)      = $DD bit 4
Covetous (Sacrifice) = $DD bit 5
Shame (Honor)        = $DD bit 6
```

6.0 The Sanity Algorithm

Because the battery can wear out, the SRAM values cannot be trusted. They may not retain their proper values, which would result in invalid or corrupt saves. Unfortunately, there is no way to prevent it. But we can attempt to detect is by using what are known as sanity algorithms. Sometimes they are called checksums, CRCs, MD5 sums, etc, but each of those other names is specific to unique process. None of these processes are used by ultima. They did something far more complex.

To keep the game thinking the SRAM is good, since it doesn't let you play the game if it thinks it's not, we need to duplicate the process it uses to create sanity values that it can check. Here is how it works.

Take the $200 bytes used by the particular save game.

You need a value which will be the sanity byte. Set this value to 0. You also need a counter value. Set this value to 0.

For every byte $001 to $1FF ($200 bytes = $000 - $1FF), do the following

1) XOR (binary exclusive OR) this number with one of the following numbers $55, $AA, $33, $CC, $A5, $5A, $BB, $99. The number depends upon your counter. Counter at 0 = $55, Counter at 7 = $99.
2) Add the XORed value to the sanity value.
3) If the sanity value exceeds $FF (255), drop the higher numbers. $123 = $23. $1FE = $FE. etc.
4) Increment the counter.
5) If the counter is > 7, set the counter to 0.
6) Repeat steps 1-5 for each byte $001 - $1FF

Once you are done, you will have the primary sanity value. This value should be stored in the SRAM at offset $00.

But you're not done yet. We need two secondary sanity values. To get the first, XOR the primary with $AA. To get the second, XOR the primary with $55.

Now, you have three sanity values. For game one, store then in the SRAM at addresses $1900, $1903, and $1906. For game two, it's $1901, $1904, and $1907. For game three, it's $1902, $1905, and $1908. These are the nine bytes thar are not part of the $200 byte game data.

If you want to hack the SRAM, but don't want to generate the SRAM values, you can use this program I wrote. It will take an NES Ultima SRAM file, generate the sanity values and write them to the SRAM file. Then you can see your modifications. Of course I would recommend my hack4u program as a complete solution, but if you want to play with values you can't modify with hack4u, this is a good alternative.

```
------------------------------ begin sanity.c ---------------------------------
/*
 * sanity
 * Copyright (C) 2004 emuWorks
 * http://games.technoplaza.net/
 *
 * sanity is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * sanity is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with sanity; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 */

#include <stdio.h>

#define SRAM_SIZE 0x2000
#define SANITY_OFFSET 0x1900

#define GAME_SIZE 0x200
#define GAME_OFFSET 0x1A00

static const unsigned char SANITY_XORS[] = {
    0x55, 0xAA, 0x33, 0xCC,
    0xA5, 0x5A, 0xBB, 0x99
};

unsigned char getSanity(const char *game) {
    int counter = 0;
    int offset;
    unsigned char sanity = 0;

    /* For every byte $001 to $1FF, do the following */
    for (offset = 1; offset < GAME_SIZE; offset++) {
        /* XOR the number with one of the following numbers */
        char xor = (game[offset] ^ SANITY_XORS[counter]);

        /* Add the XORed value to the sanity value */
        sanity += xor;

        /* If the sanity value exceeds $FF, drop the higher numbers */
        /* this is done automatically since our sanity value is a char */

        /* increment the counter */
        counter++;

        /* if the counter > 7, set the counter to 0 */
        counter &= 0x7;

        /* repeat steps 1-5 for each byte */
    }

    return sanity;
```

```
}

int main(int argc, char *argv[]) {
    FILE *f;
    char sram[SRAM_SIZE], *game;
    int game_number;
    unsigned char primary, first, second;

    /* check for the SRAM argument */
    if (argc != 2) {
        fprintf(stderr, "syntax: sanity ultima_sram_file\n");
        fprintf(stderr, "example: sanity ultima.sav\n");

        return -1;
    }

    /* make sure we can open the file for reading */
    if ((f = fopen(argv[1], "rb")) == NULL) {
        fprintf(stderr, "error: unable to open %s for reading.\n", argv[1]);

        return -1;
    }

    /* make sure we were able to read a full SRAM file */
    if (fread(sram, SRAM_SIZE, 1, f) != 1) {
        fprintf(stderr, "error: unable to read SRAM file data.\n");
        fclose(f);

        return -1;
    }

    fclose(f);

    /* print the three sanity values for each game */
    for (game_number = 0; game_number < 3; game_number++) {
        /* find the game data within the SRAM */
        game = (sram + GAME_OFFSET + (game_number * GAME_SIZE));

        /* generate sanity values */
        primary = getSanity(game);
        first = (primary ^ 0xAA);
        second = (primary ^ 0x55);

        /* alter the SRAM with the new sanity values */
        game[0] = primary;
        *(sram + SANITY_OFFSET + game_number) = primary;
        *(sram + SANITY_OFFSET + game_number + 3) = first;
        *(sram + SANITY_OFFSET + game_number + 6) = second;

        /* display those sanity values */
        printf("primtary sanity for game %d = $%02X\n",
               (game_number + 1), (unsigned int)primary);
        printf("first sanity check for game %d = $%02X\n",
               (game_number + 1), (unsigned int)first);
        printf("second sanity check for game %d = $%02X\n",
               (game_number + 1), (unsigned int)second);

        if (game_number < 2) {
            printf("\n");
        }
```

```
    }

    /* write these values back to the save file */
    if ((f = fopen(argv[1], "wb")) == NULL) {
        fprintf(stderr, "error: unable to open %s for writing.\n", argv[1]);

        return -1;
    }

    if (fwrite(sram, SRAM_SIZE, 1, f) != 1) {
        fprintf(stderr, "error: unable to write SRAM file data.\n");
        fclose(f);

        return -1;
    }

    fclose(f);

    return 0;
}
------------------------------- end sanity.c ---------------------------------

  Written in ANSI C and under the GNU GPL, it should compile under any platform.
  I have put up a windows binary at
  http://games.technoplaza.net/hack4u/sanity.zip

-------------------------------------------------------------------------------
| 7.0 Contact Information
-------------------------------------------------------------------------------

  The author, John Ratliff, can be contact at
  http://www.technoplaza.net/feedback.php.
```