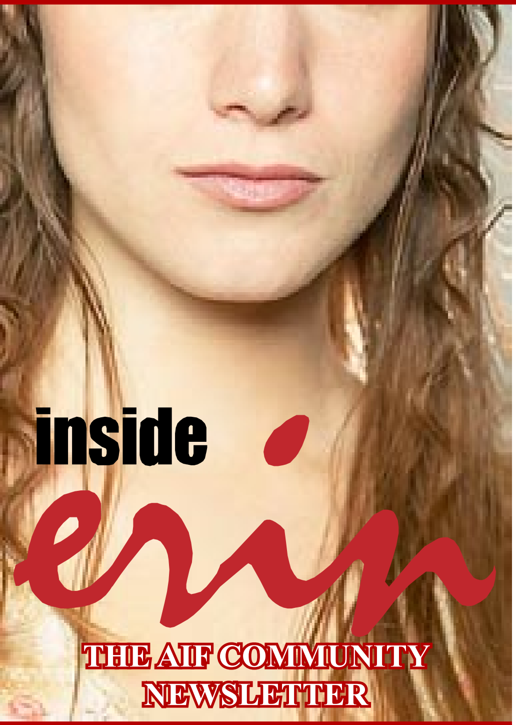
Volume 5 Number 8 September 2009



Contents

Mission Statement

Inside Erin is written and published by people who enjoy AIF. It is done for fun, but we also have some goals that we seek to achieve through the newsletter:

1. To encourage the production of more quality AIF games by providing advice from game developers, and by offering constructive criticism that is specifically relevant to AIF.

2. To encourage activity and growth in the AIF community. We aim to generate a constant level of activity so that there aren't long periods in which people can lose interest in AIF.

3. To help document and organize the AIF community. This is done through reporting on games and events, as well as by helping to organize community-wide activities such as competitions and the yearly Erin Awards.

A Letter From the Editor

Purple Dragon

Welcome all. First off let me explain to anyone who might not have heard, or to anyone reading this months or years down the line that no, you didn't miss last month's issue. Due to lack of content and an incredibly lazy editor there was no August 2009 edition of *Inside Erin*. I'm sorry for that, and sincerely hope that it will be a one time anomaly. Only time will tell.

This month I got off my butt and wrote a couple of reviews to go alone with the one for Raul's new game sent in by ExLibris. A. Ninny graces us with a new installment of *The Aphrodite Chronicles*, and we have a brand new edition of *Coder's Corner*.

I'm also throwing in a little something extra this month. On occasion I like to do logic puzzles in my free time. Yeah, I know (*coughnerd*). Well, I wrote one up this month and am including it just in case anyone wants to give it a try. If you don't know what I'm talking about there is more information about it later in the issue.

That's it for this month. As always, feel free to drop us a line and let us know what you think. There's no way for us to know if we're getting it right or wrong unless we hear from you. I hope you enjoy your stay, and we'll see you next month. ◆

This Month In AIF

by BBBen

This is the first newsletter for two months, since we missed last month's issue. I'm reasonably sure Purple Dragon will be addressing that in his editorial, but I thought I'd just mention it here as well. We've talked before about how the drop in game releases and community involvement has made a monthly newsletter seem inappropriate, but there was a lot of backlash against the idea at the time. The penny really drops, however, when we just don't have anything much to say and we can't convince anyone to write anything much for the edition. I've made all these arguments before, of course, so I won't keep carrying on.

Anyway, despite the fact that I'm writing for two months, there's still not heaps to say, since the reason we didn't publish last month was that there wasn't anything to write about. The creative side of the community was pretty quiet after the mini-comp, with the main appreciable activity taking place on AIFGames.com with Holet continuing his live AIF *The Rival*. This turned around with the release of a new game, *The Last Week Before the Wedding,* the long anticipated second game by Raul (he was talking about this game long before releasing his first).

In the discussions on the boards, apart from the discussion of the new game, there also seemed to be quite a few people digging up older games to play (not that that's a bad thing). And that's about it for now, so have fun until next month (and if you keep the place busy then maybe we'll have enough for another issue of the newsletter!). ◆

The toughest thing any writer faces is a blank screen. I mean it may be different if you're punching a clock, but sometimes it's hard to come up with something to say.

Now I can't speak for everyone, but with my semi-busy life I often think to myself, well I'll just sit down for an hour of so and create a bit of my latest epic game. Then nothing, not a sausage. The ideas just won't flow.

It's worst when you're collaborating, because even if no one's saying anything, and I've never had anyone complain, you feel that you should be doing something. And that makes it worse because then you force yourself to create something, and that just doesn't work.

It's my major bugbear, I'm just not disciplined enough to work to a routine. It's also why I bounce from project to project. If I was a little more disciplined I could probably write a lot more.

In some ways you're lucky, no really you are, that I even mange to write these articles. It's mostly because I've promised I'd do them and I hate to let people down. Even if that means going over the same thing again, as I'm sure I've done before. Not that I don't enjoy writing them, it's just they'd never get done without a deadline. And even that slips now and again. I'm just lucky I've got an understanding editor.

The irony is that when I sat down to write this article, with ideas already formed, I couldn't think of what to write. So I've just waffled on about how I've got no idea of what to say and vola, a whole article.

Continued on page 3

It's been a busy couple of months at the Collective in our Games Section. Recently there has been a discussion on people posting other people's games. You can follow that discussion here: http://hypnopics-collective.net/viewtopic.php?f=11&t=16127 It's a discussion that I think should be read by both game designers and players alike. But the one thing that I want to say is, if you like a game, why don't you send the game's creator a note of thank you or something? If you encourage them, you'll see more games. If you don't, then why would they? Everyone likes a compliment. Please think about giving them some?

In other news…

Sirgiggles did post in his thread about the S.U.B.Mission Spa and Resort, that he hasn't abandoned the game but is looking for a Java programmer to assist him. You can apply in this thread: http://hypnopics-collective.net/viewtopic.php?f=11&t=10222

Dragoon93041 posted a new version of his game Slime Kingdom. He also posted that he is pondering another version of the game for the near future. You can see the discussion of this game here: http://hypnopics-collective.net/viewtopic.php?f=11&t=15851 You can find the newest version (1.3) here: http://rapidshare.com/files/253973142/slime_kingdom_v1.3.rag

Lydia02 posted that she is not going to be as involved in the creation of games as she has been in the past. She is still working on her University Sim 2 game with a select number of people she has invited to do so. Remember that comment I made about supporting your game designers at the beginning? Something you should be thinking about.

Matrix1807 posted that his game N.T.J. is, thanks to help from Guntag and his magical paysite, the game is once more being worked on. He is working on finishing two full zones before releasing the updated version. The puzzles will be more complex and with varied outcomes, stay tuned…

Orcha posted that his game Demon Town has a new version in the works, but its appearance could be, in his words, longer than a month but less than a year.

Continued on page 3

TFGames, continued from page 2

Due to *Inside Erin* taking a bit of a break last month, this is what happened in July.

First Gunny finally revealed to the world what he's currently working on with the WWYS (RAGS) a nice start to a college based game, but he's decided to work on other projects first.

On the RAGS side, games have also come out with unusual themes, *Mboy* by Rrr5sg has Age Reduction; *Auntie* by Sayder has a little Weight Gain but mostly Breast Expansion. Whilst strangest of all *Slime kingdom* by Dragoon, well, it does exactly what it says on the tin.

On the more 'traditional' we have Moradin's *After the Fall*, a man who suffers a nervous breakdown and wakes up a woman. And Cry of Pain's *Saw*, about someone being punished, by being turned into a girl. Add to that TinaB multimedia extravaganza that is *Haunted Mansion* and MadisonX's poser rich *The Institute* and you have a rich source of demos to fill your evenings. Finally, Shrimpy returns with a little Inform demo called *Vials*, with promise of a new game built up module by module.

This month has if anything been quieter than the last, so embarrassingly so that I've had to release two whole games of my own just to fill the gap. Both are for RAGS and are *Office Politics*, for's Vengeance's *Back to School Challenge*, and *In HAARmS way*, which isn't.

The other entries for Vengeance's little comp have also surfaced, both RAGS games, Lydia's rerelease of *University Sim*, and Robyn's *The Body Builder*. Closing date is the 10th of September so you might just be able to squeeze something in if you hurry.

Things have been so quiet that TinaB, with a little prompting, has been doing a little spring cleaning of the old topic. That and a new chat function should make it the board to watch in September.

So another month another random column, look forward to more in the near future. ◆

Collective, continued from page 2

Dromdri posted a new game called For Fear of Little Men. It's a short little game with a lot of potential! You can find the game here: http://hypnopics-collective.net/cpg132/displayimage.php?pos=-94199 And if you have ideas, please add to the thread here: http://hypnopics-collective.net/viewtopic.php?f=11&t=15809

DragonTrainer posted an update to his game Jigsaw Town II! You can find it here: http://hypnopics-collective.net/cpg132/displayimage.php?pos=-94955 There were a lot of battle bugs, so he simplified the game battle system to produce this version. If you have bug notes or other ideas, please post here: http://hypnopics-collective.net/viewtopic.php?f=11&t=13871

Tilde is working on a game and needs image help. A lot of it. I mean tons and tons. He notes that, "I am programming a RAGS game of epic proportions, with 20 enemy ladies, 7 or so bosses, 5 worlds...you get the picture." You can follow that thread here: http://hypnopics-collective.net/viewtopic.php?f=11&t=15658

Windsongbard noted that, he received permission to post Slave Maker on his 4shared site. If you want to check it out just click this link: http://www.4shared.com/dir/18947900/3fbaaa56/Slave_Maker.html These are the exact same files that are on the Master Bloodfer site which as of this post which have not been updated since July 28th. This is still version 14 so if you downloaded it within the past 3 weeks you should already have the latest version. The updates for the game are on his 4shared site as well. He will try to keep it updated with the latest version. Just check it out and also drop by the Master Bloodfer Site and please let Cmacleod know what you think about it the game and feel free to make any suggestions to him. The game thread can be found here: http://masterbloodfer.creatuforo.com/1-tema5140.html?start=0

Darstan posted a new version of his game Bodywerks! You can find it here:
The discussion thread is here: http://hypnopics-collective.net/viewtopic.php?f=11&t=14623 And he has announced that version 1.2 of Bodywerks is scheduled for release in the fall of this year, but that could extend to Christmas depending on real life.

A NEW GAME! Dollmistress has released a game called Space Ditz! You can find the beta game here: http://www.megaupload.com/?d=E3PK8UC2 Discussion of the beta game is here: http://hypnopics-collective.net/viewtopic.php?f=11&t=15986 If you like latex, hive minds, mind control and all of that good stuff, this is a game you have to try! The v1.1 version can be found here: http://www.megaupload.com/?d=E3PK8UC2 and http://rapidshare.com/files/265930864/Space_Ditz_1.1.rag.html here! Discussion of the V1.1 version can be found here: http://hypnopics-collective.net/viewtopic.php?f=11&t=16031 Dollmistress has indicated that there is no plans for a version 1.2, but maybe with some encouragement something else might happen?

NandiBear posted a game called In HAARM'S Way. Nandi remembers one day ready one of the forum's about Xiriels Genie Gone Wild, when someone pointed out a plot hole. And thought yes they were right. And Xirel kindly let Nadi poke around the code, so as a thank you, Nandi created this little game. You can find it here: http://rapidshare.com/files/271046845/In_HAARMS_way.rag Discussion thread here: http://hypnopics-collective.net/viewtopic.php?f=11&t=16116

Wightwashed and NandiBear are also working on The House That Jack Built's next version release. According to Wightwashed, that release should be very soon indeed…

Blau posted a game called Harem Collection. You can find it here: http://hypnopics-collective.net/cpg132/thumbnails.php?album=4233 and the discussion thread is here: http://hypnopics-collective.net/viewtopic.php?f=11&t=15636 It's a game about… Well, Harem collecting of course… It's a short quick but fun idea!

Finally, Benmbedlam has posted Rough Landing 2! You can find the game here: http://www.megaupload.com/?d=4R4JSO9V and http://www.mediafire.com/?kmrm3zrdezz If you haven't played Rough Landing 1, you need to play that first! That game (V1.3) can be found here: http://rapidshare.com/files/129206297/Rough_Landing_1.3.rag.html Discussion of Rough Landing 2 can be found here: http://hypnopics-collective.net/viewtopic.php?f=11&t=16144 This is really an amazing piece of work that Benmbedlam has created. In his words: The idea with RL2 when I started making it over a year back was to try and move away from the popular "gauntlet" style of play that most games on this site go with. Not to disparage this style, but to see if something a bit more open ended was possible to make in RAGS. Course since then there have been games that have really pushed the boundaries like Bodywerks and Space Ditz, but hey competition is a good thing right? Even so I can say that RL2 does introduce gameplay elements that haven't been used before. There are four possible player characters to choose from that while all play through the same game have some drastically different encounters, options and items. Even a unique character in one case. There's animated gifs and real time decision making in a couple of places. The main thing I wanted to have in the game and the one rule I always tried to follow is that every single encounter has more than one way of resolving, and I believe I've achieved this. Some have far more than two and some of the other options are significantly harder to find but they're in there. There are some random elements to the game, though this is primarily to determine the order in which events take place rather than locking the player out of content.

Play this game and help him make it better!!!

And that's all from the Collective!! ◆

Dear Mortal Men and Women,

I apologize for the gap of a few months between my last letter and this one. When you're thousands of years old as I am, a few months just don't have a lot of meaning - it's an incredibly minute portion of the span of my life.

This month I thought I'd change pace and expound on something a little different. When people learn that I'm the goddess of love, the first thing they always ask me (Well, actually, the second thing right after 'will you have sex with me?') is 'what is love?' It seems that humans have an incredibly hard time defining what love is, even though it shows up in every cultural artifact humanity creates. Art, movies, books, music... everything.

Humans consider love to be a magical thing, something that happens to them (when they fall in love, for instance). They consider it something that has to be broken into little "love is . . ." fragments, and still it's not fully understood.

Believe it or not, I have an actual definition for what love is. This definition is very wise, and goes back thousands of years, but for some reason isn't widely described or used, probably because it is very prosaic. It's prosaic, but that makes it extremely useful.

But before I tell you the definition, I want to demonstrate something. Try this: think about someone you love. Close your eyes and think really about them. I'll wait . . . . OK, now name some of the things you love about that person. I'll wait again. . . . Chances are all the things you're naming are qualities and virtues of that person. He or she is beautiful. He or she has a great sense of humor. He or she has integrity, is caring, is great in bed... whatever it is. And how does it feel for you to think about that person and name those qualities? Good, right? Well, what you're doing when you name the qualities of the person you love is actively engaging in love for that person. And that brings me to the definition of love, which is:

Love is the feeling that we get when we focus on and appreciate the virtues of another person.

That's it. Read it again. Love is the feeling that we get when we focus on and appreciate the virtues of another person.

So when we focus on and appreciate the virtues of another person we are creating love. And when we focus on the shortcomings of another person we're creating something that's - well, not love.

One of the reasons I'm still needed here on Earth is that it's so much easier for people to focus on shortcomings than on virtues. Part of that could be innate to humans' psyches, but most of it is cultural. We're being programmed to feel like we fall short. Popular media barrages with images of ideal people who we're never going to live up to (and don't really exist in the first place without airbrushing), and they use these feelings of inadequacy that they have deliberately created to sell us stuff. I know: rant: modern capitalism; rant; sexist advertising, etc.

The point is that since love is the feeling we get when we focus on and appreciate the virtues of another person, we can and should use this very simple technique to create feelings of love *all the time*. Very simply, the next time you see the person you thought of in our little exercise a few paragraphs back, focus on and appreciate one of his or her virtues and then tell him or her, 'you are' and say whatever the virtue is. That will make you feel love and he or she will feel loved and appreciated. Think of it as a game; love is supposed to be fun, after all. You are! No, you are!!

I apologize that this month's entry is very much full of boring Love Goddess business. I hope it was at least a little helpful and promise to get back to the juicy stuff soon.

Until then, I wish you all wonderful love.

Aphrodite

So what is a logic puzzle anyway? A logic puzzle gives you some basic information, a list of clues, and then asks you to use deduction to sort all the information into your answer. For instance, if I told you that x=1, and that y=x, I'm sure that everyone reading this would be able to tell me what y equals. The answer, of course, is 1, but how do you know that? I didn't actually tell you that, but from the two statements I did give you there is no other possibility. Obviously that's a ridiculously simple example, but it illustrates the concept. If you want some more specific help on how to solve these things just jump on Google. There are hundreds of sites out there that will tell you more than you ever wanted to know.

Although the puzzle here is obviously more complicated than my example above, I still don't think you will find it all that taxing. I've solved probably hundreds of these things in my lifetime, but I've only written a few. The ones I've written tend to be either very easy, or very hard, and I tried to err on the side of easy in this case. A lot of times you will see these puzzles accompanied by a grid, but in this case, I don't think a grid would be of much help. The chart I added below the puzzle is all that I would use to solve this one, and it should be all you need.

I doubt that this will become a regular feature of the newsletter for two reasons. First, they're kind of hard to write, much harder than solving them in my opinion. Second, I can't imagine that more than a couple of you will even bother to try it. Of course, if I'm wrong about that and you like it then I'll certainly try to do another one some time. I'll print the answer to this one in next month's newsletter. If anyone wants to show off their smarts, just solve the puzzle and e-mail me your solution. I'll print your name, along with everyone else who got it right, before the solution next month. And now, on with the puzzle. Good luck.

All in a Day's Work

Bob Studmuffin is quite the ladies man. Last Saturday he managed to bag five of the lovelies on the same day, and they were a sight to behold. Each had a particularly sexy occupation (including one who was still just a school girl). Of course, it wasn't all fun and games. Before the girls would give it up, they all required a very specific gift from poor Bob (including a bottle cap, and a cool, refreshing glass of water), but in the end he managed to get them all, implanting his seed a different number of times (10, 15, 20, 25, or 30) in each girl before she begged him to stop. From this information and the clues below can you determine each girl's name (one was Amanda), occupation, and gift received as well as how many times Bob came during the encounter?

1. The five women are Lisa, Sheri, the one who received a pencil, the maid, and the one that Bob came 25 times with.
2. Bob came five more times with Sheri than he did with the girl who received the candy and five more times with her than with the secretary.
3. The number of times Bob came with the nurse was five more than with the girl he gave the stamp to, but five less than during his encounter with Erin.
4. During his time with the cheerleader he came more times than with Heather, but less than with the girl who got the pencil.
5. Neither Erin nor the secretary received an edible gift.

Girl	Occupation	Gift Received	Number of Orgasms

Last Week Before the Wedding

Review by ExLibris

Title:	*Last Week Before the Wedding*
Author:	Raul
Release Date:	August 16, 2009
Platform:	ADRIFT 4.0
Size:	512kb
Content:	mf, ff, mff, voyeurism, nc, underage, toys, spanking (all sexual content is optional)
Type:	Puzzle-based romp
Length:	Long, approx. 3-4 hours.
Reviewed:	September 2009
Extras:	Winning the game gives cheat commands and suggestions for funny things to try.

Basic Plot/Story

It takes a brave man to release the first full-length game since *School Dreams 3*, all the more so since it seems inevitable that *SD3* will sweep the Erins this year (even in the categories it doesn't qualify for). But Raul is that man. His second game, *Last Week Before the Wedding*, represents the culmination of three years work and is the largest non-graphical AIF ever created for ADRIFT.

The player takes on the role of Travis as he discovers that he will have to completely reorganise his wedding a week before it's supposed to happen. The game is non-linear in the sense that although Travis has a list of tasks he has to complete it is largely up to the player in which order he tackles them.

Overall Thoughts

Plot

As a motivation for the protagonist I found organising Travis and Samantha's wedding to be too abstract, especially since Samantha is absent from most of the game and her relationship with Travis is never described in any detail. As a result it's unclear why they want to get married. You might assume that they're in love, but for a game supposedly about a wedding the L word is largely conspicuous by its absence.

Of course, in reality the wedding is only a device to get Travis to interact with the other characters since, like most AIF protagonists, his actual goal is to have as much sex as possible. While that doesn't necessarily decrease enjoyment of the game, I did find it disappointing that the plot of *Last Week* was so secondary to the sex.

Despite these misgivings, I think that the actual execution of the plot works well. The player never feels railroaded into a particular course of action, though it has to be said that a lot of the time his path is perhaps too well signposted. I also liked the way that certain actions trigger events, such as the arrival of Samantha's family. This gives the impression that the story as a whole is moving forward, as opposed to the protagonist meandering through a static plot.

Character

Characterisation is a strength of *Last Week*. Most of the characters have a large amount of implemented dialogue, and many of the responses change to reflect events in the game. Given how many characters there are in *Last Week* this is an impressive amount of work. This high level of implementation helps to establish the characters as real people rather than simply objects. It also helps to make them distinct in the player's mind, an important consideration given how many characters there are.

Unfortunately I think the author is guilty of trying too hard in some places. For example, I learned that Christine had been homeless because she wedged it into the conversation the first three times I asked her about something. "Aha!" you say, putting on your deductive reasoning cap, "if she's been homeless perhaps bribing her with a few luxuries might be the means of getting that passkey she has." Except that you don't need to do any thinking because she tells you all that herself, as well as half a dozen stories about how grateful she's been for gifts in the past. In fact you almost never have to deduce anything about a character, because most of them tell you all about themselves. In the rare cases where they don't, e.g. Holly, some other character will tell you. This makes the game quite straightforward, but it also makes the characters seem superficial as there's nothing left below the surface for you to discover. "Show, don't tell" is a maxim of writing fiction that could have been used to better effect here. However, in the grand scheme of things "trying too hard" isn't exactly a major flaw. I'd be quite pleased if more games were as 'flawed' in this respect as *Last Week* is.

Writing

In general the quality of the writing is reasonable. In particular the environments are well described and implemented. The descriptions of the locations also reflect the people who normally inhabit them. For example, Kelly's home furniture is barely used since she's a workaholic who's always in the bakery. This is a source of characterisation that is often neglected, so it was good to see it put to use here.

However, after reading a sentence I occasionally found myself thinking that it didn't sound quite right. This was more common when it came to dialogue, with some lines sounding unnatural and stilted if I imagined them being spoken out loud.

Puzzles/Gameplay

Part of the satisfaction of playing IF in general comes from solving the puzzles. *Last Week* has a large number of puzzles, but unfortunately it tends to veer towards making them too easy by giving clear indications of not only what the player is supposed to do next, but also the exact command in many cases. This is probably a lesser sin than making the puzzles frustratingly hard, but it did reduce my sense of accomplishment.

The puzzles themselves are quite well constructed and generally make sense as things that Travis would have to do in order to make sure that his wedding takes place. However, they do occasionally call attention to themselves as puzzles by blocking alternate solutions. For example, Stacey will only accept a $150 coupon from Jenny's hair salon as payment for doing the flowers for the wedding; Travis can't just give her the money. It's also quite easy to block off certain paths without realising it (e.g. forgetting to pick up the bra in the dressing room, or giving the schnapps to Christine before meeting Kirsten), which led to me quitting the game in frustration at least once. This is mitigated somewhat by the efforts that have been made to create multiple solutions to certain puzzles.

Sex

The author's writing is strongest when it comes to description, and strongest of all when describing sex. *Last Week* has eleven interactive scenes, as well as a number of non-interactive scenes that Travis can enjoy if he is in the right place at the right time, or has found the right object. Mechanically each of the scenes is very similar, with two responses implemented for each of up to fifteen actions. However, despite this each scene has something that makes it unique and distinctive.

Like the puzzles, many of the sex scenes fall into the trap of being almost too easy. The women virtually hurl themselves at Travis and afterwards tell him how great he was. I know that AIF protagonists are wish-fulfilment figures in this regard, but the extent to which it's taken with Travis strained my suspension of disbelief. The other minor criticism I had was that few of the characters attach any emotion to their encounters with Travis. It's more "How can I thank you for finding that DVD? Oh, I know..." This happens so frequently that after a while you get the feeling that none of the characters actually know how to express gratitude except through sex.

If you can maintain your suspension of disbelief, most of the scenes are very enjoyable. One of my favourites was the Amanda/Maggie scene, which whets your appetite with a teaser scene, and then keeps cranking up the anticipation by forcing Travis to remain a spectator as the two girls enjoy themselves, until they finally let him join in. Another scene I enjoyed involved Samantha's little sister, Kirsten. Although it's another case of a girl hurling herself at Travis, the fact that Kirsten is a virgin stops her from hurling herself too far, and Travis has to gently guide her to her destination. Again this has the effect of increasing the anticipation and getting the player interested in the scene. The only scene I really didn't like was Holly's. An NC scene feels out of place, and frankly having her say "you made me feel like I've never felt before" after Travis has effectively raped her just disturbs me.

Technical

Given its size, *Last Week* is comparatively bug free. I noticed only a handful, such as the game's assumption that the PC will only strip in the presence of some other person, or the fact that it's apparently impossible to distinguish any regular telephone from the PC's cell phone. These bugs and glitches are annoying rather than game-breaking. There are also a few typos and spelling mistakes, as well as formatting errors. The fact that they are uncommon probably makes them more noticeable when you do encounter them, but again they're an annoyance rather than being a major issue.

On the plus side there are a number of nice programming touches, such as the way character descriptions and responses vary according to circumstances. Completing the game also reveals a number of meta-functions that have been built into the game for those who care to use them, such as the ability to summon objects or view the status of certain characters.

Intangibles

I have to admit that I was initially a little disappointed when I first played *Last Week*. Not, I hasten to add, because it isn't a good game, but because it wasn't quite what I was anticipating. I really enjoyed the author's first game, *Riding Home*. It was a pleasant change of pace from the key/lock puzzles that frequently characterise relationships in AIF, and was the sort of character focused game that I enjoy. Consequently, I was hoping that the author's second game would be similar to his first, just on a larger scale. Of course it wasn't, and in hindsight it's difficult to see how such a game could be carried off (though that doesn't stop me hoping).

Comparing the two games, one of the things that struck me was the difference in the protagonists. The PC in *Riding Home* came across as intelligent and mature, which is something of a rarity in AIF (and sadly didn't carry over to his cameo in *Last Week*). By contrast Travis is an almost stereotypical AIF protagonist, which feels like a step backwards. Something else that disappointed me a little was how irrelevant the actual wedding was. It doesn't even take place ingame, and could have been changed to someone else's wedding or some other event entirely without affecting the overall game very much.

However, it's very heartening to see an author who has released a strong minicomp game go on to release a full-length game that's equally strong (albeit in different ways). In theory the minicomp is supposed to be a catapult that launches the careers of new authors, so it's nice to see it working in practice

Final Thoughts

Overall, *Last Week* is a very solid game where the good considerably outweighs the bad. There are some things that I would have done differently, but those quibbles don't stop *Last Week* from being a very enjoyable piece of AIF. While I don't quite consider it a classic in the same league as *SD3* there are some areas where *Last Week* is arguably stronger. Given that this is only Raul's second game, I'm greatly looking forward to his third. I only hope we don't have to wait another three years for it.

Rating: A-

The Corruption of Alex

Review by Purple Dragon

Title:	The Corruption of Alex
Author:	Dr. Fanfic
Release Date:	April 1, 2004
Platform:	ADRIFT 3.9
Size:	29KB
Content:	mf
Type:	ANW
Length:	Short
Reviewed:	September 2009
Extras:	None

Basic Story

You are Alexandra Sullivan, a biochemist working on an orbiting space station. During your time there you have developed a crush on your coworker, Mark Rockway, but deem it inappropriate to do anything about it while working with him. When Mark is infecting by a strange powder you realize that you may not have the option of keeping things on a professional level.

Overall Thoughts

I liked the idea of this game. It seemed a nice premise, and should have worked well within the bounds of the rules for the mini-comp that it was written for. In addition, there are so few games with female PCs, that it is nice to see one from time to time. However, for several reasons, the game just fell flat and didn't live up to my expectation or hopes.

Puzzles/Game Play

There are several different puzzles in this game, or perhaps it would be more accurate to say several different steps in the one main puzzle. The puzzles themselves are logical, but there are still some major problems, which I'll discuss under technical below.

Sex

This is possibly the only AIF game that I have ever played where to get the 'best' ending, and the most points, it necessary to have absolutely no sex of any kind. When Mark is infected his sexual drive kicks into overdrive and he begs you to help him relieve the tension. You can do just that, but each time you do you lose points. Still, you will probably want to do so since helping him out in this way before you cure him is the only sex in the game. Once he is cured the game ends, which makes it feel like I just went through all the work for no reward.

The game could have been much better if a sex scene was added at the end, after Mark has been cured. Or the author could have really gone the extra mile and implemented a full scene for when Alex is infected as well. How it stands now there is just a cutscene showing the two of them fucking each other's brains out until they die. Of course, this being a mini-comp entry I wouldn't expect all that, but it still would have been nice.

Technical

Well, here is the main problem with the game. When you load up the game, play through the first few turns, and have a look around the room it will probably be pretty clear what you need to do. The problem is getting there from here. Yep, I'm talking about guess the verb problems, and quite a few of them. The only thing that (almost) saved the game is that the author has included hints. I say almost, because one of the first things that I used the hints to find out gave me a solution that still didn't work. I had to look up a walkthrough to find the right command. Some of the others are just as bad, but at least the hints give the right commands for those.

Another thing that I found rather annoying is that while the environment is implemented fairly well, with descriptions of all the objects mentioned in the room description, there are no body parts to examine. "X tits"? "Nothing Special." Poor girl.

Final Thoughts

As I said, I liked the idea for the game, and the writing is actually pretty good. I haven't seen anything more from this author, and I don't know if he is still around, but if so I hope he tries again because he has potential. However, for this game it just felt like maybe he ran out of time and cut some corners. Unfortunately, the corners that were cut were things like sex (rather important in AIF) and betatesting (which would have caught at least some of those guess the verb problems).

Rating: C-

The Oval Office

Review by Purple Dragon

Game Info:	The Oval Office
Author:	Faraday
Release Date:	April 1, 2004
Platform:	ADRIFT 4
Size:	21KB
Content:	mf
Type:	ANW
Length:	Short
Reviewed:	September 2009
Extras:	None

Basic Story

You are the president of the United States and one of the interns catches your eye and your interest enough to see if something more might come from it. Sound familiar?

Overall Thoughts

I think that the idea here is a good one for a mini-comp game, and it is implemented rather well. The stage is set well, complete with little tidbits of historical background that help to flesh it out.

Puzzles/Game Play

There are a few minor puzzles to get the game moving in the right direction, but I doubt they will give anyone any problems. They are fairly logical and intuitive and don't suffer from guess the verb problems that sometimes crop up.

Sex

This game is all about the sex, and there is nothing wrong with that. I really liked how the scenes progress and escalate through different rooms and situations, building the tension in the player as the tension builds in the characters themselves. A bit more could have been added, allowing more of the tasks that were completed earlier to be redone in the new situation where more freedom is available, but I suppose that is usually the case.

Technical

Although certainly not an overly ambitious game from a technical standpoint, what is here is clean and well done. Very, very few typos and no guess the verb problems make the game easy and enjoyable to play.

Final Thoughts

All in all, I enjoyed the game. The characters and environment have enough depth to allow you to get into the swing of things, and the sex writing is, if not incredibly deep, still well done and pretty hot. Well worth a play when you get a chance.

Rating: B-

For the last few months we have been examining a few different things that anyone writing an IF game of any type might want to know how to do. I think it has been good, and hopefully helpful, but I also think it's time to put the 'A' back into the equation. In that spirit, this month's topic is layered clothing.

Some of the pros and cons of working with layered clothing have been discussed a lot over the years. There is no doubt that it makes for a more realistic experience (although you have to make sure that 'realistic' doesn't equate to 'tedious'). There is also no doubt that making this type of system work, and work well, is not easy. There are a lot of things to consider and a lot of things that can go wrong if you are not very careful. I'm sure we won't be able to hit on all of those, but hopefully this will at least give you a good starting point if you are considering making layered clothing a part of your new game.

Goals:
Demonstrate how to set up a clothing system with at least two layers (obviously). Descriptions of the examined person and his/her body parts should change depending on what clothing is worn. There should also be some mention of how the basic sexual commands would be affected by the clothing. Either not allowing it, or even better, changing the description to mention working around or through the barrier.

TADS 2 Segment by A. Bomire

In this edition of Coder's Corner, we will be creating some clothing for an NPC. That is no trick in TADS, as clothing is already defined in TADS. So, we are going to create layered clothing.

Layered clothing implies that there are multiple articles of clothing, some of which is worn over top of other articles. Providing for layered clothing means addressing some basic issues when writing descriptions of characters and actions. These issues include:

-Hiding/revealing articles of clothing as layers are added/removed
-Changing descriptions of certain body parts as layers are added/removed
-Allowing/disallowing the wearing and removing of clothing depending upon what is worn "on top" of it

I have worked with a lot of layered clothing systems. There are many TADS 2 AIF libraries that all handle multiple layers of clothing, as well as a module written specifically just for layered clothing (this is an "IF" module, in that it doesn't handle body parts). They all work very similarly, but they are also a little more complex than what we might want to tackle here. So, I'm going to go over a simpler solution that is more specific to a single situation. The above-mentioned libraries have really broad solutions that can handle any clothing in any situation.

First, let's set up our test game. In this one, the player is with his girlfriend, Erin, in their bedroom. We could outfit her with a whole closet full of clothes, but to demonstrate layered clothing I'll just limit her to a shirt and bra. Other layered clothing works much the same. The limitations we'll impose is that Erin cannot wear her bra over her shirt, nor can she take off her bra unless she has removed her shirt. (These same limitations of course apply to the player as well.) When describing Erin, we will list her as wearing a shirt if it is worn, or her bra if that is worn and her shirt is not, or nude when she is wearing nothing. When describing Erin's breasts, we will alter the description depending upon which clothing item is worn: One description when her shirt is worn, one for when her bra is worn, and one final one for when she is nude. Last but not least, when describing her bra, we will change its description if Erin is also wearing her shirt.

This all sounds fairly complicated, but in practice it isn't all that difficult. It basically amounts to using some *if-conditions*to alter text. I would not recommend using TADS embedded expressions, as they are designed to handle only one conditional test and our output will probably require multiple conditions. But, in the right situation they would work as well and feel free to use them. (For more information on using TADS embedded expressions, please consult the TADS manual, Chapter 8 – Language Reference.)

That's enough talking, let's get to coding! First, we'll create the bedroom and its occupant, Erin:

```
startroom: room
 sdesc = "Erin's Bedroom"
 ldesc = "This is Erin's bedroom, which as the name implies contains
   her bed. "
;

bed: beditem
  location = startroom
  sdesc = "bed"
  ldesc = "This is Erin's bed. "
  noun = 'bed'
;

Erin: Actor
  location = startroom
  sdesc = "Erin"
  adesc = "Erin"
  thedesc = "Erin"
  ldesc =
  {
    "This is Erin, your girlfriend. She is a very attractive
     young woman. ";
    if (shirt.isworn)
```

```
        “Erin is currently wearing a white shirt. “;
       else if (bra.isworn)
        “Erin is sexily dressed in just her bra. “;
       else
        “Erin is topless! “;
     }
     noun = ‘Erin’
     isHer = true
    ;
```

As I say every month, in a real game this would be expanded with much more detail. You’ll note that I have altered Erin’s *ldesc*property to be a full method that checks to see what clothing articles are worn and alters her description to match. In a fuller game, there would be more checks for other clothing she might have, such as jeans, panties, sweaters, dresses, etc. Since her shirt “takes precedence”, so to speak, I check for that first, then her bra and finally list her as topless. If she had a sweater that she could wear over her shirt, for example, then I would place its check before checking for her shirt.

That takes care of Erin. Let’s look at her body, specifically her breasts (as they are the only body parts affected by the clothing in this demonstration). Similar to how I altered the *ldesc*property of Erin, I will also conditionally alter the description of her breasts:

```
    Erinsbreasts: fixeditem
     location = Erin
     sdesc = “Erin’s breasts”
     adesc = self.sdesc
     thedesc = self.sdesc
     ldesc =
     {
      if (shirt.isworn)
       “You can make out the rounded shape of her breasts through
        her white shirt. “;
      else if (bra.isworn)
       “You can see the creamy swells of her breasts in the
        cups of her bra. “;
      else
        “Erin has wonderful breasts. “;
     }
     noun = ‘breast’ ‘breasts’ ‘tit’ ‘tits’
     isThem = true
    ;
```

Once again, in a full game this would be much more detailed. For other body parts, you could simply extend the above example. That seems simple enough, so let’s move on to the actual clothing.

You can use TADS normal *clothingItem*class for your layered clothing, but there are some things you will have to override. For one thing, TADS by default doesn’t allow the player to remove or take off articles of clothing that the player isn’t wearing. Because of this, the description of clothing being removed is written with the premise that is the player who is wearing the clothing. Both of these will need to be altered so that it takes into account the player or the NPC (i.e., Erin) who is removing the clothing. This is done by altering the verification and action methods for the *removeVerb*verb. Another thing to keep in mind is that once removed, TADS doesn’t know that a bra can’t be worn by male players (for example). Or that Erin’s bra might be too small for Jackie. So, any such distinctions will have to be caught in the verification and action methods for wearing the clothing, which is the *wearVerb*.

```
    shirt: clothingItem
     location = Erin
     sdesc = “white shirt”
     ldesc = “A very nice white shirt. “
     isworn = true
```

```
        noun = 'shirt'
        adjective = 'white'
        verDoUnwear(actor) =
        {
          if (not self.isworn)
           "The shirt is not being worn right now! ";
        }
        doUnwear(actor) =
        {
          "%You% remove%s% <<self.thedesc>>. ";
          self.isworn := nil;
        }
        verDoWear(actor) =
        {
          if (self.isworn)
           "The shirt is already being worn! ";
          else if (actor <> Erin)
           "The shirt is too small for you! ";
        }
        doWear(actor) =
        {
          "Erin puts on the white shirt. ";
          self.isworn := true;
          self.moveInto(Erin);
        }
      ;
```

You'll note that I am taking advantage of TADS format strings to automatically substitute either a "You" if the player removes the shirt, or "Erin" if Erin removes the shirt. (For more information regarding the use of format strings, please read the TADS manual, Chapter 5 – Advanced Parsing Techniques.) These descriptions are very ...well, non-descriptive. In a real game, you would probably spice them up a bit.

The last piece of clothing we are going to look at is Erin's bra. In most respects, the code for Erin's bra is very similar to her white shirt. The difference is that the bra can be worn under the white shirt, and cannot be worn over it. So, we need to alter some of the descriptions to account for this.

```
      bra: clothingItem
        location = Erin
        sdesc = "bra"
        ldesc =
        {
         if (shirt.isworn and self.isworn)
          "You can't really see the bra through Erin's shirt. ";
         else
          "It is a pretty little bra, with little hearts on it. ";
        }
        isworn = true
        noun = 'bra'
        verDoUnwear(actor) =
        {
          if (not self.isworn)
           "The bra is not being worn right now! ";
          else if (shirt.isworn)
           "%You% can't remove the bra - her shirt is in the way. ";
        }
        doUnwear(actor) =
        {
          "%You% remove%s% <<self.thedesc>>. ";
          self.isworn := nil;
        }
        verDoWear(actor) =
        {
          if (self.isworn)
            "The bra is already being worn! ";
          else if (actor <> Erin)
            "You can't wear the bra! ";
          else if (shirt.isworn)
            "Erin can't wear the bra - her shirt is in the way. ";
        }
        doWear(actor) =
        {
          "Erin puts on the bra. ";
          self.moveInto(Erin);
          self.isworn := true;
        }
      ;
```

As you can see, the additional code doesn't amount to much – just an extra check in removing or wearing the bra. If there were additional layers to worry about (a sweater, a coat, etc.) then these *if-conditions*could get rather long. But, they aren't that complicated – just additional conditions to check.

If you've tested this, you've probably discovered that there is a slight problem. Erin will not remove her own clothing – the player has to do it. This is because by default TADS prevents characters from following the player's orders. This is simple enough to correct, and we addressed this in more detail back in "Coder's Corner" #2. We simply have to alter Erin's *actorAction*method to allow her to respond to the "wear" and "take off" commands. To make it more specific, we'll only allow her to respond to these commands when the object being referenced is a clothing item. This way, she won't try to wear the bed, for example. For clarity, I won't repeat Erin's entire definition, but simply list the addition to the code described above:

```
      actorAction(verb, dobj, prep, iobj) =
      {
        if ( (verb=wearVerb or verb=removeVerb) and
           isclass(dobj, clothingItem) )
          return true;
        else pass actorAction;
      }
```

This makes use of the *isclass*function to determine whether or not an object is an article of clothing. To read more about this function, check the TADS manual, Chapter 8 – Language Reference.

One final bit of clean up: TADS associates the word "remove" with "take" or "get". This is fine for most objects in the game, but for clothing it is a different matter. You don't have to worry about it, because by default if the clothing item is worn then TADS treats "remove" as "take off", and treats it as "get" if the clothing isn't worn. However, you will have to add the *takeVerb*to Erin's *actorAction*method in order to get her to respond to "remove shirt":

```
      actorAction(verb, dobj, prep, iobj) =
      {
        if ( (verb=wearVerb or verb=removeVerb) and
           isclass(dobj, clothingItem) )
          return true;
        else if ( verb = takeVerb and isclass(dobj, clothingItem)
            and dobj.isworn )
          return true;
        else pass actorAction;
      }
```

This simple addition will have Erin "remove shirt" by trying to take it off if it is worn, and trying to pick up or get the shirt if it is not.

The above techniques would be extended to handling any sexual interaction with your character. I don't think it is necessary to go into details on this, since it is simply adding in conditional statements such as seen above when describing the sexual interactions in the same manner as I have used to describe Erin and her body parts.

That pretty much wraps up a simplistic way of handling layered clothing. You can use the same methods, extended with as many conditional statements as necessary, for as many layers as you need – and even for clothing worn at the same "layer" (a button up shirt and a t-shirt, for example).

For a little more sophistication, you can alter the clothing's visibility depending upon what clothing is worn over top of it. For example, for Erin's bra, we could add this little piece of code:

```
      isVisible(vantage) =
      {
        if (self.isworn and shirt.isworn) return nil;
        else pass isVisible;
      }
```

This would have the effect of making her bra invisible to the player while her shirt is being worn. For any commands the player directs towards her bra, he will get a message from TADS saying "I don't see any bra here." This has a two-fold benefit: 1) You don't have to worry about describing the bra when other articles of clothing are worn over it, and 2) You don't have to worry about trying to remove the bra when other articles of clothing are worn over it.

Another sophisticated technique is making partially worn clothing. An example of this would be a button-up shirt which has been unbuttoned – making the bra beneath visible, but not removable. This involves simply altering the above listed conditional statements to take the shirt's status into consideration. Or, you could have see-through clothing. For example, a t-shirt in a wet t-shirt contest. Again, this involves altering the above listed conditions. By extrapolating the techniques outlined above, you should be able to construct and describe any number of clothing items worn in almost any variation.

TADS 3 Segment by Knight Errant

This month's task is layered clothing. Coding for layered clothing is more difficult than it first seems because it has a wide range of effects. First, there has to be some sort of layering system so that the game can tell what clothing is "on top" of what other clothing. Clothing needs to be categorized, so that tops don't hide bottoms and vice-versa. Finally, body part descriptions need to check what clothing is worn and redirect their descriptions to the clothing. This task is a big part of why most AIF authors use AIF libraries, which provide pre-made code for these sorts of things.

There are a number of ways to create a layering system. Some library authors use a numerical scale to rank what clothes go on top of other clothes. In coding my Futuo AIF library, I broke down clothing types into categories: underwear, outerwear, and overwear (jackets, coats and cloaks). Underwear and outerwear had three types, top, bottom, and full. Overwear only had top and full. Although this does not code for the full range of possibilities, I feel it accounts for the majority of common AIF situations. I coded these as sub-classes of the TADS 3 class "Wearable".

First, we must make some adjustments to the Wearable class, to adjust things to AIF standard.

```
        modify Wearable
        // The removeMsg is the message displayed when the item is removed.
        // Override this for a more sensual description.
            removeMsg
            {
                if(gActor==gPlayerChar)
                    "You remove <<theNameWithOwner>>.  ";
                else
                "<<gActor.name>> removes <<theNameWithOwner>>.";
        }
    // This overrides the default handling to allow you to remove
    // clothes from other people and to drop clothes when removed, instead
    // of putting them in inventory.
        dobjFor(Doff)
         {
              verify()
              {
                  /*
                   *   Make sure any actor is actually wearing the item.  If
                   *   they're not, it's illogical, but if they are, it's an
                   *   especially likely thing to remove.
                   */
                  if (!isWorn)
                      illogicalNow('No one is wearing that.');
                  else if (isHidden)
                     illogicalNow('You can\'t get at that yet.');
                  else
                      logicalRank(150, 'worn');
              }
              action()
              {
                 /* un-wear the item and describe what happened */
                 makeWornBy(nil);
                 moveInto(gActor.location);
                 "<<removeMsg>>";
              }
         }
        canBeWornBy = [owner]
        dobjFor(Wear)
         {
              preCond = [objHeld]
              verify()
              {
                  /* make sure the actor isn't already wearing the item */
                  if (isWornBy(gActor))
                      illogicalAlready(&alreadyWearingMsg);
                  if (canBeWornBy.indexWhich({ cur: cur == gActor}) == nil)
                     illogical(cannotWearMsg);
              }
              action()
              {
              inherited;
              }
         }
        wornDesc = "<<self.owner>> is wearing <<self.theName>>.  "
        unWornDesc = "<<self.theNameWithOwner()>> is lying in a pile.  "
        cannotWearMsg = 'That isn\'t your size.  '
        desc {
         if(isWorn) "<<wornDesc>>";
         else "<<unWornDesc>>";
        }
    disambigName = (theNameWithOwner())
    ;
```

By now, most of this code shouldn't be too hard for people who have been following my TADS 3 section. First, we modify removeMsg to provide different messages based on who is doing the removing. Second, we change the Doff command to drop clothes instead of putting them in inventory, and to allow us to remove clothes from other people, which is not permitted in default TADS. We add a property called canBeWornBy, which is a list of characters who will be permitted to wear the item. By default, we set it to owner, which is the person whose inventory is in. We also add two properties, wornDesc and unWornDesc. The description property is what is called when someone types > LOOK AT ITEM, so we will use desc to determine which message should be called when someone tries to look at the item. Finally, we set disambigName = {theNameWithOwner()}, because I hate it when I type >X BRA and the game replies "Do you mean the bra, or the bra?" This way, it will reply with "Do you mean Erin's bra, or Julia's bra?"

Now, we code the various clothing layers as classes inheriting from the Wearable class. If you want to make things extra complicated, feel free to make them Openable as well. I'll leave that as an exercise for the reader. Here are the classes:

```
//OverwearTop is a class for clothing which is worn on top of normal
//outerwear, such as jackets and topcoats.
class OverwearTop: Openable, Wearable
// Since we append the clothing descriptions (if worn) to the character
// descriptions, we don't need to list them in NPC's inventories.
    isListedInInventory = (!isWorn || isWornBy(gPlayerChar))
    titsDesc = "You need to override titsDesc for <<self.theNameWithOwner()>
>.  "
;
```

In our schema, overwear will never be hidden, so this section is all pretty simple. We'll be adding the clothing descriptions to the character's descriptions, so we remove it from the inventory list. titsDesc is what the player will see if he types > X TITS while this item of clothing is being worn. OverwearFull is defined identically, with an added assDesc and pussyDesc. By default, we set it to a warning to the author about exactly which property he forgot to customize for his game. :)

Outerwear is the first layer that will be able to be hidden, so this one is slightly different.

```
// OuterwearTop is a class for outerwear which covers only the top half
// of the body, such as shirts
class OuterwearTop: Wearable
    isHidden
     {
  if(self.isWorn==true && location.topOverwear != nil)
            return true;
        return nil;
     }
    desc {
        if(isHidden) "<<hiddenDesc>>";
        else if(isWorn) "<<wornDesc>>";
        else "<<unWornDesc>>";
        // This shouldn't ever be called.
        return nil;
    }
    hiddenDesc = "If anyone is wearing <<gDobj.aName>>, {it's dobj/he's}
covered.  "
    titsDesc = "You need to override titsDesc for <<self.theNameWithOwner()>
>.  "
;
```

The isHidden property is used to determine if the item is currently visible. In this case, it's hidden if the item is worn by someone and if location (the person who is wearing it) has the property topOverwear set to be other than nil. topOverwear is a custom property we will have to add to Actor. Note, isHidden was used back in our custom Wearable class to prevent it from being removed if it's currently hidden. We also add a hiddenDesc to allow for people trying to look at the item when it's covered. This can be overridden by the author if he wants to allow for translucent items or other similar things. OuterwearBottom is coded virtually identical to OuterwearTop, except in isHidden we use location.bottomOverwear, and we code for an assDesc and pussyDesc. Finally, the underwear classes use topCovered and bottomCovered. We also code an underwearTop and underwearBottom to handle looking at body parts when underwear is worn.

At this point, the clothing would layer properly if only we had defined the location.topOverwear property and it's other versions. This is a little tricky, because we can't simply check to see if the character is wearing clothing of a certain class. There might be an easier way to deal with this, but this is the way I got it to work.

```
/*
 *   topOverwear and bottomOverwear are used to determine if the outerwear
 *   classes are revealed or hidden.  topOverwear is non-nil if the actor is
 *   wearing OverwearFull or OverwearTop and it's closed.  bottomOverwear is
 *   non-nil if the actor is wearing OverwearFull and it's closed.
 */
    topOverwear {
          return contents.indexWhich({ cur: cur.ofKind(OverwearFull) ||
cur.ofKind(OverwearTop) && cur.isWorn && !cur.isOpen });
         }

    bottomOverwear {
          return contents.indexWhich({ cur: cur.ofKind(OverwearFull) &&
cur.isWorn && !cur.isOpen });
         }
/*
 *   topCovered and bottomCovered are used to determine if the underwear
 *   classes are revealed or hidden.  These methods iterate through all the
 *   actor's contents and check for outerwear of the appropriate categories.
 *   Both are covered by OuterwearFull
 */
    topCovered {
     return contents.indexWhich({ cur: cur.ofKind(OuterwearFull) ||
cur.ofKind(OuterwearTop) && cur.isWorn });
         }
    bottomCovered {
     return contents.indexWhich({ cur: cur.ofKind(OuterwearFull) || cur.ofKi
nd(OuterwearBottom) && cur.isWorn });
         }
    underwearTop {
     return contents.indexWhich({ cur: cur.ofKind(UnderwearTop) && cur.isWorn
});
         }
    underwearBottom {
     return contents.indexWhich({ cur: cur.ofKind(UnderwearBottom) &&
cur.isWorn });
         }
```

I've included a check if the item is open to show how one could possibly add in openable clothing, but I haven't actually coded for openable clothes yet. Let me explain what's going on here. These properties are to be defined on the Actor object that will be wearing the clothes. Each property returns a value that's equal to the index (number in the list) of the first item of the specified class. The syntax { cur: cur.ofKind(class) } is an abbreviated syntax for looping through the list. For each item in the list, it is temporarily assigned to the variable (in this case, "cur") and the check (cur.ofKind(class) && cur.isWorn) is performed. The result is passed back to the indexWhich method. As a result, if the property underwearBottom returns nil, no item of class UnderwearBottom is being worn by the character. If the character is wearing an item of that class, the property will return the index of the item. That's not particularly useful for us, which is why in the definition of the classes I only check the value against nil.

Lastly, we can alter the description of the body parts based on what the character is wearing. Here's an example:

```
        class tits: BodyPart
            name = 'tits'
            vocabWords = 'breast/breasts/tit/tits/boob/boobs/tittie/titties/hooter/
        hooters/jug/jugs/melon/melons/nipple/nipples/areola/areolae'
            isPlural = true
            titsDesc ="<<theNameWithOwner>> need a real description.  "
            dobjFor(Examine)
            {
                action()
                {
                    return location.titsDesc;
                }
            }
```

So, if someone looks at these breasts, it returns the titsDesc of the Actor who has them. Of course, that's only useful if the character's titsDesc property does something useful ... which it does.

```
        titsDesc
            {
        // If the top is covered by overwear
                if(topOverwear)
                {
        // iterate all objects in the inventory
                    foreach(local cur in self.contents)
        // Is the object of class OverwearTop or Overwear full?
                    if(cur.isWorn && cur.ofKind(OverwearTop) || cur.ofKind(OverwearF
        ull))
        // If so, use the titsDesc from that object
                        return cur.titsDesc;
                }
                else if(topCovered)
                {
                    foreach(local cur in self.contents)
                    if(cur.isWorn && cur.ofKind(OuterwearTop) || cur.ofKind(Outerwea
        rFull))
                        return cur.titsDesc;
                }
                else if(contents.indexWhich({cur:cur.ofKind(UnderwearTop) &&
        cur.isWorn}))
                {
                    foreach(local cur in self.contents)
                    if(cur.isWorn && cur.ofKind(UnderwearTop))
                        return cur.titsDesc;
                }
                else
                {
                    foreach(local cur in self.contents)
                    if(cur.ofKind(tits))
                        return cur.titsDesc;
                }
        // Since the tits object contains a default titsDesc, it shouldn't ever
        return nil.
                return nil;

            }
```

The titsDesc property first checks the property topOverwear. Remember, if no overwear is worn, it returns nil. If it is, it returns the index number of that clothing ... since TADS 3 treats any non-zero number as TRUE, it will grab the titsDesc of the first OverwearTop or OverwearFull the character is worn. If not, it goes to topCovered, and then to UnderwearTop, and finally for any tits objects. We end with a nil just in case something goes stupidly wrong.

Now we have a basic layered clothing system ... as you can see, it's a pain in the ass. Until next month!

Inform 6 Segment by 'trix

Greetings, lowly ones.

What should layered clothing do? Here's some reasonable requirements:
1) clothes go on different areas
2) clothes go over other clothes
3) clothes may prevent the wearing and removing of other clothes
4) clothes affect a character's description in a logical way
5) clothes appear in some reasonable way in your inventory
6) clothes may cover up particular body parts, affecting actions involving those parts
7) anything else I think of in the next few hours

A good way to get started is to decide that clothes will have an area, which determines where on the body they are worn, and a number that is low for underwear and high for outerwear, to determine what clothes go on top and which go underneath.

```
    Class  Clothes
     with  clothing_area 0,
           clothing_layer 0
     has   clothing;
```

Now, what values are we going to use to specify the areas? That was rhetorical: I actually know the answer because I've done this before. We're going to use an exciting thing called a bit field.

```
    Constant    FACE_AREA = $01;
    Constant    ARMS_AREA = $02;
    Constant   CHEST_AREA = $04;
    Constant MIDRIFF_AREA = $08;
    Constant  CROTCH_AREA = $10;
    Constant     ASS_AREA = $20;
    Constant    LEGS_AREA = $40;
    Constant    FEET_AREA = $80;
```

These numbers are all powers of two (specified in hex). If you don't understand that, your homework is to look up "bit field", "hexadecimal" and "exponent" in Wikipedia; but the important information is that `clothing_area` can be a bunch of these `_AREA` constants added together, and we can check whether an item of clothing goes over a particular area with the test `(item.clothing_area & particular_area)`.

So how do we use this to prevent the wearing or removing of other clothes? Normally, we'd do this with a `before` routine on the class. So let's do that.

```
    Class  Clothes
     with  clothing_area 0,
           clothing_layer 0,
           before
           [;
            Wear: if (self in player && self hasnt worn)
               return self.blockwearing();
            Disrobe: if (self has worn)
               return self.blockdisrobing();
           ],
           blockwearing
           [ x p;
              if (self.clothing_area==0) rfalse;
              p = parent(self);
              objectloop (x in p) if (x has worn && x ofclass Clothes
                             && x.clothing_layer >= self.clothing_layer
                             && (x.clothing_area&self.clothing_area))
              {
                 print_ret (CTheOrYou) p,” can’t wear “, (the) self,
                    “ over “,(the) x,”.”;
              }
           ],
           blockdisrobing
           [ x p;
              if (self.clothing_area==0) rfalse;
              p = parent(self);
              objectloop (x in p) if (x has worn && x ofclass Clothes
                             && x.clothing_layer > self.clothing_layer
                             && (x.clothing_area&self.clothing_area))
              {
                 “You would have to remove “,(the) x,” first.”;
              }
           ],
     has   clothing;
```

blockdisrobing goes the wearer’s inventory. If it finds a worn item of clothing with a greater value for clothing_layer on any of the same clothing areas, it prints a suitable rejection message and returns true (which, in typical I6 style, means “stop”).

blockwearing goes the intended wearer’s inventory. If it finds a worn item of clothing with a greater or equal value for clothing_layer on any of the same clothing areas, it prints a suitable rejection and returns true.

It would be convenient to store what item (if any) is being worn over another, and then we can use that to control descriptions and whatever else it turns out we need. This only changes when clothes are put on and taken off (successfully), so let’s add an after routine to keep track of that.

```
     ! in Class Clothes
           cover 0,
           findcover
           [ x p;
              if (self hasnt worn || self.clothing_area==0)
              {
                 self.cover = 0;
                 return;
              }
              p = parent(self);
              self.cover = 0;
              objectloop (x in p) if (x has worn
                             && x.clothing_layer >self.clothing_layer
                             && (x.clothing_area&self.clothing_area)
                             && (self.cover==0 || x.clothing_layer >
     self.cover.clothing_layer))
              {
                 self.cover = x;
              }
           ],
           after
           [ x p;
            Wear: p = parent(self);
              objectloop (x in p) if (x has worn
                         && x ofclass Clothes)
              {
                 x.findcover();
              }
            Disrobe:
              objectloop (x ofclass Clothes)
                if (x.cover==self)
                   x.findcover();
           ],
```

This sets `obj.cover` to an item of clothing being worn on the same area as `obj` with a higher `clothing_layer`. If there are several such objects, it picks the one with the lowest `clothing_layer`. So if you're wearing a bra, a T-shirt and a coat, the bra's cover will be the T-shirt, and the T-shirt's cover will be the coat. It sets `cover` whenever an item is worn or removed, but to make sure they're correct at the start of the game you can do this:

```
     [ Initialise   x;
        objectloop (x ofclass Clothes)
           x.findcover();
        ! probably some other initialisation...
     ];
```

Now we probably want a convenient way to list the items of clothing someone is wearing. For descriptions, this shouldn't include information about items covered up, since they're not visible to casual examination. Probably. The following routine will go through the items in someone's inventory and pick out the ones that are worn and are not covered by anything.

```
     [ FlagWearing p   x n;
        objectloop (x in p)
        {
           if (x has worn &&
               ~~(x provides cover && x.cover))
           {
              give x workflag;
              ++n;
           }
           else
              give x ~workflag;
        }
        return n;
     ];
```

This marks anything that should be listed with the attribute `workflag`. After that you can use `WriteListFrom` to list those items. So then to list what the PC is wearing on examination, you might put something like this:

```
     [ pcdesc;
        print "You look much the same as ever. You are wearing ";
        if (FlagWearing(player))
        {
           WriteListFrom(child(player), WORKFLAG_BIT|ENGLISH_BIT);
        }
        else print "nothing";
        ".";
     ];
```

And then in Initialise, add the line:

```
  selfobj.description = pcdesc;
```

Here's some example clothes to try it out.

```
    Clothes panties "white panties"
      with name 'panties' 'knickers' 'white',
           clothing_area CROTCH_AREA + ASS_AREA,
           clothing_layer 2
      has  pluralname;

    Clothes trousers "black trousers"
      with name 'trousers',
           clothing_area CROTCH_AREA + ASS_AREA + LEGS_AREA,
           clothing_layer 5
      has  pluralname;
```

Note that the trousers go over the same area as the panties, but also go over the legs area. Also, the trousers have a higher `clothing_layer` than the panties, because trousers typically go over the top of underwear.
If you wanted several variations on a item of clothing, you'd probably want to inherit a class from `Clothes`, something like this:

```
    Class  Bra
     class Clothes
     with  name 'bra' 'bras//p',
           clothing_area CHEST_AREA,
           clothing_layer 2
    ;

    Bra   bluebra "blue bra"
     with name 'blue'
    ;

    Bra   yellowbra "yellow bra"
     with name 'yellow'
    ;
```

I'll mention this now because it's one of the things that people consistently forget: if you have several objects that share noun vocabulary, do make sure the plurals get properly parsed. That means if you have several bras, put `'bras//p'` in their `name` arrays. The same logic applies to tables, chairs, girls, **drawers in a desk**, and anything where a bunch of objects share a noun in their vocabulary.

Next in my list of things to do is inventory. The player-character is probably aware of all the things she is wearing, so it doesn't make sense just to list outerwear. So how do we want the list to appear?
Ideally (in my opinion) we want the wide inventory to look like this:

```
You are carrying a yellow bra.
You are wearing a blue bra and some black trousers (under which are some white
panties).
```

... and the tall inventory to look like this:

```
You are carrying:
  a yellow bra
You are wearing:
  a blue bra
  some black trousers, under which are:
    some white panties
```

That seems like a lot of work. Fortunately, I've already done it. Waaay back when we were doing "Programming Erin", I wrote a file called `GInv.h` to list inventory without bodyparts, but as is my wont I also made it list clothing the way I like it. This file is available at the Yahoo aifarchive. To include it in a game, put the following lines between `Include "Parser"` and `Include "VerbLib"`.

```
    Replace InvSub;
    Include "GInv";
```

This uses the `cover` property, which is the real reason it's there.

Now ... clothing covering body parts. For this, we will need a bodypart class. This is going to be fairly minimal, because it's not supposed to be the focus of this article. If you want to write a proper bodypart class (and handle genitives properly) go back to Programming Erin.

```
    Class  Bodypart
     with  body_area 0,
           topcover
           [ x p c;
              p = parent(self);
              if (p==0) return 0;
              objectloop (x in p)
                 if (x has worn && x ofclass Clothes
                     && (x.clothing_area & self.body_area)
                     && (c==0 ||
                         c.clothing_layer < x.clothing_layer))
                       c = x;
              return c;
           ],
     has   scenery;
```

When specifying a body part, set `body_area` to whatever seems appropriate from the `_AREA` constants.
For instance:

```
    Class Breasts
     class Bodypart
      with name 'breasts' 'breast' 'boobs' 'boob'
              'tits' 'tit' 'chest' 'chests//p',
           body_area CHEST_AREA
      has  pluralname;
```

The routine `mypart.topcover()` will return the outermost item of clothing being worn over `mypart`. So if you're writing some tastey breast-licking action (and, indeed, why wouldn't you be?):

```
           before
           [ x p;
              Lick: x = self.topcover(); p = parent(self);
                 if (x) "You can't lick ", (the) self, " while ",
                    (theIs) p," wearing ",(the) x,".";
           ],
           ! ...

    [ theIs x;
       if (x==player) print "you are";
       else
       {
         print (the) x;
```

```
        if (x has pluralname) print " are";
        else print " is";
      }
    ];
```

However, if you're writing some stimulating arse-fondling action (and, indeed, who isn't?), you might prefer that the action be allowed to go ahead:

```
          before
          [ x p;
             Touch: x = self.topcover();
                if (x) "You grope Buffy's bum through ", (the) x,". She
                   punches you in the head.";
                "You put your hand on Buffy's bare arse and enjoy yourself.
                She stabs your face off with a wooden stake.";
          ],
```

If you're writing an NPC, it's probably appropriate to check the most noticeable bodyparts for nudity and react accordingly.

```
    Object Katy "Katy"
      with description
          [;
             print "Katy is a tall, athletic brunette in her mid-twenties. ";
             if (FlagWearing(self))
             {
               print "She is wearing ";
               WriteListFrom(child(self), ENGLISH_BIT | WORKFLAG_BIT);
               print ".^";

             }
             else print "She is completely naked.^";

             if (katybreasts.topcover()==0)
                PrintOrRun(katybreasts, description);
             if (katypussy.topcover()==0)
                PrintOrRun(katypussy, description);
          ],
    ...
```

Assuming you write Katy some appropriately named body parts with descriptions, that will generate her a suitable description. Body parts themselves also probably want their descriptions to reflect their coverings. Take your `Bodypart` class and add something like this:

```
    ...
          description
          [ x;
             x = topcover();
             if (x) self.covered_desc(x);
             else PrintOrRun(self, bare_desc);
          ],
          covered_desc
          [ cov;
             print_ret (The) self," ",(isorare) self,
               " concealed by ",(the) cov,".";
          ],
          bare_desc
          [;
```

```
             "You see nothing unusual about ",
             (the) self,".";
          ],
```

This will print "Neil's balls are concealed by the stainless steel spacesuit." (if such circumstances arise). You would generally want to customise `bare_desc` for every body part (possibly skipping feet and ears, depending on how thorough you are with your anatomy classes). And you can customise `covered_desc` as well.

```
        ! in Katy's breasts
          covered_desc
          [ cov;
             if (cov==katybra)
                "Katy's breasts look particularly sexy, barely
                concealed by her black lace bra.";
             if (cov==spacesuit)
                "You can't see much of Katy's breasts. She's
                wearing a fucking spacesuit.";
             "The shape of Katy's breasts looks very tempting
             under ", (the) cov, ".";
          ],
          bare_desc "OMFG! I seen another girl's naked boobs! W00t!"
        .....
```

There's a couple of other things to fix. Most importantly, I6 doesn't support `worn` items for NPCs, mainly insofar as that the grammar for removing clothes is written in a way that assumes only the PC can ever wear anything.
To fix the grammar, you need to rewrite the grammar for every verb that can generate a `Disrobe` action, which means putting the following lines after `Include "Grammar"`.

```
    Extend 'disrobe' replace
       * noun                                  -> Disrobe;

    Extend 'remove' replace
       * noun                                  -> Disrobe
       * multiinside 'from' noun                -> Remove;

    Extend only 'take' replace
       * multi                                 -> Take
       * 'off' noun                            -> Disrobe
       * noun 'off'                            -> Disrobe
       * multiinside 'from' noun               -> Remove
       * multiinside 'off' noun                -> Remove
       * 'inventory'                           -> Inv;
```

You'd also need to replace `DisrobeSub` if you want that to allow the player to remove NPCs' clothes.
Put the line `Replace DisrobeSub;` before including VerbLib, and then put the following routine in your code:

```
    [ DisrobeSub p;
        if (noun hasnt worn) return L__M(##Disrobe, 1, noun);
        p = parent(noun);
        if (p ~= player && ObjectIsUntouchable(p)) return;
        if (p provides preventundressing
            && p.preventundressing(noun)) return;
        give noun ~worn;
        if (AfterRoutines() == 1) rtrue;
        if (keep_silent == 1) rtrue;
        L__M(##Disrobe,2,noun);
    ];
```

This also adds an entry point for the NPC refusing to let the player undress her. In your NPC you would then put:

```
    preventundressing
    [ item;
       if (item==hat || item==coat) rfalse ! allow that
       if (some condition that indicates she hasn't been seduced yet)
          print_ret (The) self, " refuses you let you undress her.";
       rfalse;
    ],
```

It's probably also a good idea to write a `ChooseObjects` routine that is biased against covered clothing, since, ideally, the player shouldn't even know NPCs' underwear is there until she's uncovered it.

That's it for now, disciples. In the unlikely event that someone is reading and trying to use this, you would do well to combine it with the bodypart code from Programming Erin, and the parsing in `GParse.h` (especially since player's are likely to try and distinguish clothes by genitive expressions).

Have fun with your layers.

Inform 7 Segment by Dudeman

This month we will be tackling a very important topic to AIF, and that is layered clothing. Today, a layered clothing system has become very common in AIF games, but it's not an easy thing to set up for people just learning to code games. Luckily, we at the Inside Erin newsletter are here to help you with that.

Now of course, a layered clothing system is one where clothing is broken down into "layers" which can be worn on top of or under other clothing like real life clothing does. It wouldn't make much sense for someone to be able to remove a girl's panties before taking off their jeans would it? This is just one of the problems that is solved with a layered clothing system.

Now, before I start with the coding let me start by saying that I will be going off the assumption that you already know the basics of Inform 7 coding and have read the topics already covered in previous installments of Coder's Corner and its predecessor Programming Erin. Things like sexual actions, creating body parts for people, and basic clothing have already been covered and I will be expanding on them to create a more complex layered system of clothing. If you have any questions about those I suggest going back and reading those to better understand this issue.

Ok, with that out of the way let's start with the coding. First let's just set up a simple scenario to play with our layered clothing system. Right now we just need a room and a person that we can play with later.

```
     The Changing Room is a room. "A private room used for models to dress into
     their clothes for photo shoots.".

     Erin is a woman in the changing room.
```

Now that we have a women (we will deal with the description of her later), we should probably give her some clothes, right? We wouldn't want women to just walk around naked would we? Of course not ;) However, before we create her clothing, let's start setting up the basics of a layered clothing system. First we just need to create a new kind of wearable thing for the clothing items we will create.

```
     A garment is a kind of thing. A garment is wearable.
```

To have a layered system, we will also need a way to define what "layer" a piece of clothing belongs too. I find this is easily done with numbers (layer 1 being underwear, 2 being clothing worn directly over underwear, and so on). Since every garment will have a layer, we can attach a number to all garments called its "layer".

```
     A garment has a number called layer. The layer of a garment is usually 2.
```

This works fine for keeping track of layers, but right now it can't differentiate between different kinds of clothing in the same layer. When we look at what kind of behavior we ultimately want, we should have girl wearing both panties and a bra on level one, and probably a shirt and pants on layer two. For this, we want clothing worn on the top part of the body to be treated separately from clothing worn on the bottom even when on the same layer. For this, we can separate garments by where they are worn.

```
A garment can be top covering. A garment is usually not top covering.
A garment can be bottom covering. A garment is usually not bottom covering.
```

Of course you could create more body areas if you wanted to (like the head for hats, feet for socks/shoes, legs for stockings, etc…), but for this simple example we will just deal with the upper and lower body.

With this, we now have the basics to create clothes for Erin being sure to define the layer and coverage area for each new garment we create.

```
Erin's t-shirt is a garment worn by erin. The description of erin's t-shirt is
"A small, stylish t-shirt.".
The layer of erin's t-shirt is 2.
Erin's t-shirt is top covering.

Erin's jeans are a garment worn by erin. The description of erin's jeans is "A
tight pair of blue jeans.".
The layer of Erin's jeans is 2.
Erin's jeans are bottom covering.

Erin's bra is a garment worn by erin. The description of erin's bra is "A plain
white bra.".
The layer of erin's bra is 1.
Erin's bra is top covering.

Erin's panties are a garment worn by erin. The description of erin's panties is
"Some white, thong style panties.".
The layer of erin's panties is 1.
Erin's panties are bottom covering.
```

We can also create something like a dress which covers both the top and bottom part of the body.

```
A silky red dress is a garment in the changing room. The description of the
silky red dress is "A long silky red dress designed for today's photo shoot.".
The layer of the silky red dress is 2.
The silky red dress is top covering. The silky red dress is bottom covering.
```

Now Erin has a full set of clothes along with a nice dress to change into to test the system. However, as of right now the layers we made have no meaning. Erin can still put on her bra over her shirt, wear her dress over her t-shirt and jeans, and take off her panties while still wearing her pants which are all things we don't want. To fix this, we need to create a set of rules to prevent this from happening. We could use Before or Instead of rules, but these kind of situations are what Check rules were made for. In this case, we want Inform to check what a person is already wearing before we let them wear or take off any garment. We can do this using a "repeat loop" (you can learn more about them in Chapter 11.10 of the documentation) which runs through all things meeting a certain criteria and runs a check on them. In this case, to make sure a person cannot take off his/her underwear over their outer clothing, we would want to write a rule like:

```
Check an actor taking off a top covering garment worn by a person (called the
wearer):
repeat with clothing running through garments that are top covering worn by the
wearer begin;
if the layer of the clothing is greater than the layer of the noun, say "That
would be hard to take off from under [the clothing]." instead;
end repeat.
```

What this rule does is run a check every time a person in the game tries to take off a garment with is top covering which runs through all other clothes worn by that person to check if they are wearing any other top covering garment with a higher layer number than it. If it finds such a garment, it will stop the action and print out a failure response that tells the player why the rule failed. We will also want to create a similar rule for bottom covering garments as well as rules that prevent a person from putting on underwear over outerwear, wearing two garments which are the same layer and cover the same body area, and examining underwear when they are covered by outerwear;

```
Check an actor taking off a bottom covering garment worn by a person (called
the wearer):
repeat with clothing running through garments that are bottom covering worn by
the wearer begin;
if the layer of the clothing is greater than the layer of the noun, say "That
would be hard to take off from under [the clothing]." instead;
end repeat.

Check an actor wearing a top covering garment:
repeat with clothing running through garments that are top covering worn by the
actor begin;
if the layer of the clothing is greater than the layer of the noun, say "That
would be hard to put on with [the clothing] in the way." instead;
end repeat.

Check an actor wearing a bottom covering garment:
repeat with clothing running through garments that are bottom covering worn by
the actor begin;
if the layer of the clothing is greater than the layer of the noun, say "That
would be hard to put on with [the clothing] in the way." instead;
end repeat.

Check an actor wearing a top covering garment:
repeat with clothing running through garments that are top covering worn by the
actor begin;
if the layer of the clothing is the layer of the noun, say "It probably wouldn't
look good to wear that over [the clothing]." instead;
end repeat.

Check an actor wearing a bottom covering garment:
repeat with clothing running through garments that are bottom covering worn by
the actor begin;
if the layer of the clothing is the layer of the noun, say "It probably wouldn't
look good to wear that over [the clothing]." instead;
end repeat.

Before examining a top covering garment worn by a person (called the wearer):
repeat with clothing running through garments that are top covering worn by the
wearer begin;
if the layer of the clothing is greater than the layer of the noun, say "You
can't see under [the clothing]." instead;
end repeat.

Before examining a bottom covering garment worn by a person (called the wearer):
repeat with clothing running through garments that are bottom covering worn by
the wearer begin;
if the layer of the clothing is greater than the layer of the noun, say "You
can't see under [the clothing]." instead;
end repeat.
```

```
Before taking off a garment worn by erin: say "Erin is quite able to dress/
undress herself and just needs your direction as to what to wear." instead.

A persuasion rule for asking erin to try taking off a garment:
Persuasion succeeds.
A persuasion rule for asking erin to try wearing a garment:
Persuasion succeeds.
A persuasion rule for asking erin to try taking a garment:
Persuasion succeeds.

Unsuccessful attempt by someone wearing: stop the action.
Unsuccessful attempt by someone taking off: stop the action.
```

The "unsuccessful attempt" rules are just something thrown in there to stop the "Erin is unable to do that" message from being printed when a rule fails. The persuasion rules have been covered before, but they just allow you to issue commands to Erin to wear/remove clothing.

Finally, we have the basis of a working layered clothing system. It is a bit simplistic, but for this example it should work just fine. The only thing we have left to do is to show how descriptions of people, body parts, and sexual actions can be altered based on the clothes a person is currently wearing. These can all be done with basic "[if][otherwise][end if]" phrases in your text responses like so;

```
The description of erin is "Erin is a young fashion model that you hired to
model some of clothing for you. She is tall, blond, curvy, and has nice large,
firm breasts. Currently she is [if erin is wearing the silky red dress]wearing
a long, silky red dress[otherwise if erin is wearing erin's t-shirt]wearing a
small, stylish t-shirt and[otherwise if erin is wearing erin's bra]wearing a
plain white bra and[otherwise]completely topless and wearing[end if][if erin is
wearing the silky red dress].[otherwise if erin is wearing erin's jeans] a tight
pair of blue jeans.[otherwise if erin is wearing erin's panties] some white,
thong style panties.[otherwise] nothing covering her lower body.[end if]".

The description of erin's breasts is "[if erin is wearing the silky red
dress]The red dress is designed to accent a women's chest and it does just that
on Erin.[otherwise if erin is wearing erin's t-shirt]Erin's t-shirt is obviously
a few sizes too small and shows off Erin's large breasts.[otherwise if erin is
wearing erin's bra]Erin's bra is faily plain, but is still very sexy over her
large, firm tits.[otherwise]Erin's tits are very nicely shaped and very large.
Even better then you thought they would look.[end if]".

Report rubbing erin's breasts: say "[if erin is wearing the silky red dress]You
rub Erin's large tits through the silky material of her red dress.[otherwise if
erin is wearing erin's t-shirt]You gently massage Erin's large breasts over her
tight fitting t-shirt.[otherwise if erin is wearing erin's bra]Over her thin bra
you get a nice feel of Erin's firm breasts.[otherwise]You eagerly palm Erin's
huge, naked tits.[end if]".
```

You can then similarly write the descriptions of all the other body parts and sexual actions in the same way based on what the person is wearing. It may seem like a lot of extra work (and in a way it is), but to many people the extra work is well work it to create a better sexual experience by including the effects of clothing into the scene. Over all the decision to include such a clothing system is up to you, but if you do want to I hope you found this article to be helpful. As always, if you have any questions about this week's topic or anything else, feel free to email me and I would be more than happy to help. Otherwise, I will be back next month for another issue of coder's corner.

ADRIFT Segment by BBBen

For reasons not worth exploring right now I don't have much time to write this article, so I'll have to keep this brief. Layered clothing in ADRIFT is something that the engine really isn't too well geared up for, hence my usual tactic to just have a "strip" command and a "dress" command to simplify the whole thing. This does not mean that you *can't* do more complex clothing systems, but it's debatable whether it's worth it. This is the reason you won't often see anything approaching layered clothing in ADRIFT.

Anyway, first of all, I'm going to recommend you use a workaround. ADRIFT's clothing system wasn't built with the idea in mind that those clothes would be seductively peeled off, and I think it doesn't work too well for that purpose. Instead we'll use the ALR file (again, if you haven't learned to use that yet, take half an hour to read a tutorial on it and test it out – it's not too complicated and extremely useful!).

First up, create several different variables – I'm going to keep this fairly simple, but you'll see that there's potential to make it more complex if you want. Let's say the girl is wearing a t-shirt, a bra, jeans and panties, and nothing else. That means we'll have two variables, called: *topclothes* and *bottomclothes*. Set both variables to start at value 2.

In the girl's description include the following:

```
This is a damn hot girl with big tits. maintopdescription%topclothes% mainbotto
mdescription%bottomclothes%
```

Then in the ALR file include the following lines:

```
maintopdescription2|She is wearing a white t-shirt
maintopdescription1|She is wearing a white bra
maintopdescription0|She is topless
mainbottomdescription2|and has on a pair of jeans.
mainbottomdescription1|and has on a pair of panties.
mainbottomdescription0|and is naked from the waist down.
```

Obviously you can modify this to suit your own descriptions, though you may have noticed a bit of a problem with this system – that your descriptions of top and bottom will have to be elegant and modular enough so that it doesn't sound silly when you combine two descriptions. "She is topless and is naked from the waist down" sounds a little silly, really, and I could probably do better but I'm in a rush! Anyway, play around and you might be able to find better ways to make that work. In the meantime, back to the mechanics!

You can just use tasks for the stripping tasks; for example, create a task called "remove t-shirt", make it require that the *topclothes* variable equal 2, and in the actions make it change the topclothes variable to 1. In the description make it describe the t-shirt being removed. Repeat the process for a task called "remove bra", but make it require topclothes to be 1, and set topclothes to 0. Repeat the process again for "remove jeans" and "remove panties", this time using the bottomclothes variable instead.

You should also extend this to body part descriptions; for example, if you have an "x tits" task (as you probably will) then in the description put in a line like "titsdescription%topclothes%", and then in the ALR file enter something like this:

```
titsdescription2|Her tits strain her t-shirt.
titsdescription1|Her lacy bra shows off her cleavage.
titsdescription0|Her naked tits are really big.
```

Repeat the process for pussy and ass, both using the "bottomclothes" variable instead. Note that in addition to what I've outlined here, you could also have one command, like "remove dress" that impacts both variables, lowering *topclothes* and *bottomclothes* each to 1.

Okay, so that's the basics. I hope that's clear enough to be workable for you if you're thinking about doing layered clothing in ADRIFT, but I'd recommend if you're new to the system to just go for one layer of clothing. This option is available if you'd rather ignore that advice, however. ◆

If you can write game reviews, articles, opinion pieces, humorous essays, or endless blather, we want you. Contact the Editor for suggested content or just write what you want and send it to us.

Submitting your work to Inside Erin:

Please direct all comments, articles, reviews, discussion and art to the Editor at aifsubmissions@gmail.com.

Editor:

Purple Dragon has written six AIF games including *Archie's Birthday - Chapter 1: Reggie's Gift, A Dream Come True*, and *Time in the Dark.* He has received one Erin award and been nominated for several others.

Staff:

A Bomire is the author of several TADS AIF games, including *Dexter Dixon: In Search of the Prussian Pussy*, *Tomorrow Never Comes* and *The Backlot*. His games have won numerous awards and Erin nominations. He was the co-recipient of the Badman Memorial Lifetime Achievement Award in 2006.

A Ninny is an AIF player, author of four AIF games and frequent beta-tester. His *Parlour* received an Erin for Best "One Night Stand" game in 2004 and his most recent game, *HORSE* walked away with three Erins at the 2007 awards show.

BBBen is an author of a number of Adrift AIF games. His games have received numerous Erin awards and nominations and first place in A. Bomire's 2004 mini-comp. He was also the recipient of the 2007 Badman Memorial Lifetime Achievement Award.

Bitterfrost is a longtime IF/AIF player working on his first (and last) game, *How I Got Syphilix*.

Dudeman has released one game and is working on a second. He has also released an impressive Inform 7 sex extension to help make it easier for others to write games of their own.

Knight Errant is an AIF player who has released two games and is currently working on a couple of others.

'trix has released one game, *Casting*, which was written in Inform 6, and is sporadically working on another in TADS 3.