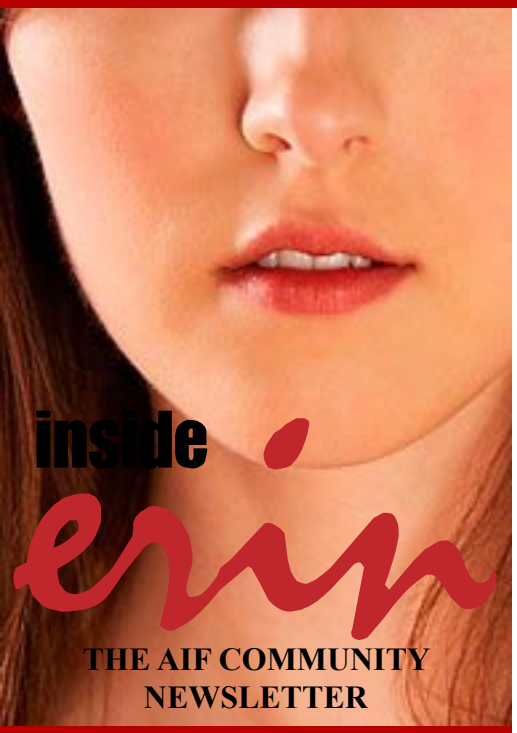

Contents

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.

Hello all you AIFers out there in AIFer land. Purple Dragon here with another issue of Inside Erin. This month we have another installment of Coder's Corner for you. I hope you're enjoying this feature as much as I am. I have to admit that I enjoy reading it quite a bit more when I'm not responsible for writing one on them. Sluffing stuff off – er, that is, delegating responsibility is one of the perks of being editor. We also have an interview this month with Wayne McWilliams, author of the game WUZ. With the exception of my "In the Hot Seat" pseudo interviews, we haven't had one in awhile, and I appreciate Mr. McWilliams' thoughtful answers to my questions.

Finally, you may have noticed that the annual Mini-comp has not been announced yet this year. I'm going to remedy that now, and you can read the details later in this issue. With the recent threesome comp, I had thought to push this back a bit to give people time to recharge. Of course, since we only received one entry for the comp (thanks rip_cpu) I suppose it's not as much of an issue. Still, I decided to give just a bit more time than normal. The due date will be June 8th, which is almost exactly three month from now. There usually tends to be about two and half months between announcement and due date, so hopefully the extra couple of weeks will help. Please do NOT wait until the last minute to start working on your game if you are planning to enter. Last year we extended the deadline two weeks. Consider this an upfront extension and a final (very firm) due date.

As always, thanks for reading, and if you have any comments, suggestions, criticisms, etc, feel free to drop us a line. See you next month. ◆

After the failure of the Threesome comp last month I decided not to release the sole entrant game myself, but rather to allow the author, rip_cpu, to do so at his leisure. At first he intended to hold back the game for further testing and polish, but then decided to simply release the game. However, it seems that nobody really noticed this game release and unfortunately it went for several days without anyone passing comment upon it. This was a rather unfortunate case, as rip_cpu had wanted some feedback on his game.

Which brings me to another topic – I'd really like it if there were a different culture around the AIF community, in which it is rather expected that you will give feedback on games you play, rather than just doing so if you happen to feel like it. It has been pointed out many times before that AIF authors don't get paid (in any way) for writing the games, but rarely is the next logical step proposed that the compensation should be to give some feedback to the author. Therefore, I encourage you to think of it this way – the games you play aren't exactly free, the price is that you make a comment about them after playing on one of the boards, or in an email to the author. And don't let yourself be deterred from commenting just because there are two or three comments on the board already! Two or three comments on a forum is not enough reward to justify writing an AIF game. Okay, enough of my pleas for attention, on to other matters…

Continued on page 3

There are many ways that mainstream AIF and its cousins TG/TF and Hypnotism differ from each other but the main one is probably the most important, losing. Now the main aim of AIF is the sex, so to lose would be, well, not to have sex, which makes no sense.

But, whilst some TG etc. games have sex in them, it's not the be all and end all. No, it's the acts that the game is written about that are the most important. In many cases people prefer the bad ending to the good one, which by default will be a return to the (almost) normal state at the start of the games.

It's true to say that for most of us, RAGS (and perhaps even ADRIFT) is for those who aren't programmers. But for Ehlanna this is untrue, because Ehlanna is one of those rarest of things a programmer who chooses to write in RAGS.

Now before I carry on I feel it's only fair to declare that I have a strong link to Ehlanna's games, the first one *State of Mind* (RAGS) from Hypnopic credits me (along with some other) whilst the second, *Debutante* has my Avatar (along with others) as a hidden Easter Egg in the game. I also hear a rumor that I'll feature as a character in her next as yet untitled next games.

So I now feel it's fair to say that as a programmer, Ehlanna tends to experiment more than some of us, and may at times add features that are not exactly essential to the flow of the game. But they're always well written, well structured, and written with a very wry sense of humor.

Finally, and perhaps most importantly, Ehlanna is a strong champion of sharing her knowledge around, always willing to give practical advice, with a number of practical demos on RAGShare and a help document in the works.

Continued on page 3

Firstly, I want to make note that this past February 23rd was the fourth birthday of the Collective! From humble beginnings we currently have over 361,000 posts on the Board and over 73,000 images on the Gallery!

Onto the happenings in the Games Forum...

Dragontrainer has updated *Jigsaw Town II*, the newest version can be found here:

http://rapidshare.com/files/181615670/Jigsaw_II_ver_1.02.rar

Xiriel released a game called *Uncle Henrey's Legacy* which can be found here:

http://www.4shared.com/account/file/85175702/75c43e49/uncle_Henreys_legacy.html

And discussions / bug reports can be placed here:

http://hypnopics-collective.net/viewtopic.php?f=11&t=14836

Darstan has noted that he is working on version 1.10 of *Bodywerks*, but the release date is unknown at this moment...

MadisonX noted that a full version of the game *Dominated* is also in the works. but no exact release time scale so far...

And that's all for this month from the Collective! ◆

AIF - Continued from page 1

A rare case of an actual discussion of writing technique and character took place on the AIF Archive, which is a nice change from intermittent hint requests punctuating long periods of silent inactivity. Also, I get the sense from the chatter that there are at least still people working on games out there – mind you this season always tends to be the busiest for game production, and we still haven't had all that much yet. No need to get despondent, however, as I think things will pick up eventually, and I've resolved to stop trying to force it. No more "feed the beast" attitudes!

Oh, and another thing that came up on the AIF Archive was a topic on an article about how perverse video games are a threat to blah blah blah, yadda yadda yadda. This incited a fair amount of derision and scorn, but also a little bit of derision and scorn directed *at* the derision and scorn. If that last sentence made any sense to you then you might check out the thread "Insidious attack on the morals of the nation" on the AIF Archive to see the details.

New Games

A Night With Dani and Liz by rip_cpu for Inform, released 3rd February 2009 (the only entry to the defunct threesome competition). Your old flame invites you to come to visit her in New York, only to reveal that she wants you to have sex with her lesbian girlfriend. ◆

TF Game Site - Continued from page 2

This month we have a mini ADRIFT revival with several writers choosing to release new games in that format. First we have Gunny1, who I've heard is a big shot in BE poser art who burst on the scene (if you'll pardon the pun) with a little demo of *Escaping Fate* (ADRIFT) a great little start which has real promise, if he finishes it. He also promised a new game for Valentine day, which never showed up, which proves my first rule of writing IF games: you don't talk about… oh sorry, I mean never give a release date. Second was Bimbo Alison with a small but perfectly formed little game, *The Note* (ADRIFT), again hopefully bigger and better things are on the way. And last, but no means least, was the start of, *The Sorority Life* (ADRIFT) which is building to an interesting little story.

Two old timers also crawled out of the woodwork to update their works in progress. Aisha Ryan finally, after many trials and tribulations, managed to finish the first part of *Molly's Mystical Musings* (RAGS), and promises a part 2 in the near future. And someone called Nandi Bear managed another chunk towards the completion of *Unfortunate Bimbo Gift*. All thanks to the praise from the many newcomers we've had added to our forum. See, paying writers complements can have good consequences; just keep on saying nice things.

On the past masters front Cleo Kraft posted a very early Beta (possibly Alpha) of *Hookey* (TADS), which I really enjoyed, but it must be said was met with mixed reviews.

Finally, in our crossover of the month, Xiriel kindly shared with us a tiny weekend written games, *Uncle Henry's Legacy* (RAGS) which he originally posted up on Hypnopic's.

Away from the actual game releases we have several discussions about the act of game creation. On whether it was better to release as one or as a serial, either depending on the person, and on whether rendered or pictures were better, either depending on the circumstances.

And a bunch of us shared some of our idea's to help start off (hopefully) a whole new slew of games for the future. Whilst Gunny posted up a whole load of rendered images with a challenge to see what we could do with them.

No promises, but this month is a special occasion with our humble little board being a whole year old. So I'm trying to arrange a special guest writer for next month's Inside Erin's so watch this space. ◆

2009 Mini-comp submission rules are as follows:

- Your game must have three or fewer rooms. Closets do not count as rooms so long as they're just places to store things. If your player is required to spend more than a couple of turns in a closet, it counts as a room.
- Your game may have no more than three characters, including the player-character(s). No more than two of those characters may participate interactively in sex scenes. This is a expansion on previous years' rules and allows the PC to be a non-participant or voyeur while the two non-player characters have sex, and also allows the game to switch the PC from one character to another.
- Multimedia (images and sounds) are permitted, but may not add more than 150KB to the native (unzipped) size of the game file.
- No part of your game can have been released to the public before the deadline.
- Your game must be winnable (or at least it must have an ending that the player can reach).

Mini-comp submission procedures are as follows:

- The submission deadline is 9:00 a.m. CST Monday, June 8, 2009.
- I will be available to help beta-test your game. Beta-testing is strongly encouraged but not required.
- I will collect the entries by e-mail and post the games on this web site. Send your entry to **purpledragon.aif AT gmail DOT com** (obscured to prevent spam).
- Authors should send a walkthrough with their entry. The walkthrough will be used by comp organizers to verify the game can be won and to provide hints for players.

Voting procedures are as follows:

- Everyone, including entrants, will be allowed to vote.
- Voters will have approximately two weeks to play all the games and vote. The voting deadline will be announced when the games are released.
- Voting will be conducted in a manner similar to that of the Erins: return a ballot marked in order of preference for each category. Votes will be counted using the instant-runoff method.
- Discussion of games (including requests for hints) will be forbidden during the voting period.
- Authors will not be permitted to post updates of their games during the voting period. They may post 'technical bulletins' with instructions as to how to work around bugs that are discovered. Technical bulletins must be approved by mini-comp organizers before being posted.

Voters will be asked to judge all the games in the following categories:

- **Concept.** Is it a good idea for a mini-comp game? Does it work well with the set limits? Does it feel complete or more like a game fragment?
- **Writing.** How well-written is it? Do the settings have the atmosphere that the author seems to be after?
- **Characters.** Do the characters 'come to life'? How sexy are they?
- **Sex.** How hot are the sex scenes?
- **Technical.** How many bugs are there? What neat tricks did the author invent?
- **Enjoyment.** How much did you like the game?

This month I'm very happy to be able to talk to the author of WUZ. If you haven't heard of it, I'm not terribly surprised since it was written in 1989, twenty years ago. Although not the first AIF game ever written, it is certainly one of the first. It is also the only one from that era that I know about where the author is still available. So on this, the twentieth anniversary of the game's release, I'm pleased to welcome Mr. Wayne McWilliams.

IE: Thank you for taking the time to talk to us a bit. Perhaps you could start by telling us just a bit about yourself.

WM: I've been 'into' computers, consoles and gaming since the early 80's. My first computer was a VIC20, then of course the C64 – which I still own, packed away with its color monitor. But my pride was the Tandy 1000SX. I've also been a writer – past tense because despite three novels none were published. On the computer, back in the early days, my favorite genres were wargames (SSI was the king of those games) and text adventures where Infocom ruled. I played an Infocom game on every computer I owned (except the VIC) thanks to Infocom's cross platforming all its games. I taught myself programming and wrote inventory programs and then jumped into games.

IE: I know that at the time you wrote WUZ, you had already written a couple of regular IF games. Of course, they weren't even called IF at that time, but you know what I mean. What prompted you to try your hand at something in the adult genre?

WM: Ok, this answer might open a can of worms. What started this whole thing was one of my younger friends wrote a Trojan – that's a malicious program that disguised itself as a legitimate program which the user would innocently download and then wreak havoc on his machine. Back then the dark side and the light side had blurred lines. He was a genius with machine language code but had no idea how to create an innocent cover and asked me for advice. I suggested any game that dealt with sex – because many wouldn't hesitate to download such a game. I boasted I could write a text adventure in a few days that he could easily slip his code into.

Well, this 'scheme' lost steam – he went on to something else, but I kept the idea of trying to write an AIF. I had just finished an IF adventure and decided to give it a try.

IE: The game is nearly unique in that you are not only allowed to choose your gender, but also your sexual preference. What led you to include those choices?

WM: Here's another can of worms. WUZ began and was conceived as a gay game. Back then, and as far as I know today, there were very few gay AIF. I only recall one I found, whose name I can't even recall. That was the framework WUZ began with. Then I realized there'd be no chance of wide distribution since gay was so taboo. So, after my friend suggested his Trojan scheme, I already had the game's basics down. So I went back and rewrote it to allow the user to choose their sex. Then I rewrote the responses depending on how the player might approach the sex situations – be it gay or straight.

As I did so I also realized it'd be great 'closet' game – meaning a person hiding their true sexual preference could be free to experiment in the game. In other words, the star quarterback with so many girlfriends could hit the dorm room and act out his true fantasies. They could download a 'sex game' without revealing they were playing a gay role.

IE: I know that you wrote it in BASIC. Although there weren't the same number of authoring systems available at that time, I do know that some were. Specifically, I know that AGT, which some of us are still familiar with, was released in 1985 (although I believe under a different name at that time). What made you decide to go with BASIC, rather than one of the other systems?

WM: AGT, at that time, was too bulky for me. It also required the end user to have the program. I wanted to keep the games short, simple, with the minimal space requirements. I was also concerned about who would actually own the copyright. I was comfortable with BASIC, and when I upgraded to Microsoft's QuickBasic 4.5 – which could compile the code into a stand alone executable file – it seemed the best solution. The parser was a challenge, and underwent revisions and improvements over months until I was quite pleased with it. Everything was written from scratch. I would never attempt that again. BASIC also allowed me to print out the code and send it to Washington for copyright registration.

IE: What would you say were some of the main problems with writing games in that era compared to today?

WM: Size, size, size. Size mattered – the smaller the better. We were dealing with limited storage, limited memory, limited processing power. Not only that, but to distribute your program required uploading to bulletin boards which, back then, had limited and very slow transmission speeds ranging from 300 up to 9600 baud – but very few had 9600 – the common would be 1200 and 2400 baud. And boards had limited storage as well. So writing a game – no matter how grand you hoped it'd be, or what innovative ideas you might have had – you were forced to make compromises.

IE: On a related topic, what are some of the general differences that you have noticed relating to AIF or games in general between 1989 and today?

WM: In 1989 – authors and their intended audience were younger than today. We're talking high school to college age. So you have to judge any game of that era – IF or AIF – with that in mind. Especially considering an older audience would consider a lot of that stuff, crude, simple, immature, even silly. Today, AIF has grown up and matured mostly from what it started as. Back then – plot and complex puzzles meant little. It was more shock value and quick porn than an attempt at writing Henry Miller or DH Lawrence.

I would also venture to say most of the players of AIF today are older because today the younger generation don't have the patience or the focus to play though a text adventure when they have such options as PS3 and 360 games. This was evident even back in 1989 when the graphic heavy RPG games began to squeeze out IF games. Infocom/Activision apparently realized this and basically gave up by 1989.

IE: Have you played any "modern" AIF?

WF: To be honest – no. My gaming is focused on the Xbox 360 and computer games and I simply don't have the time. I have browsed through a few. I am very impressed with TADS and Inform – the things you can do now opens up a lot of possibilities for a creative mind.

IE: Have you ever considered updating WUZ or writing a new game now that there are much better resources for game writers?

WM: I wouldn't update WUZ because it's a game totally reflective of its era – the late 1980's – and I prefer it to stay as a slice of that era. It was intended to be a short, simple, fun game to be played between classes and forgotten. That's why I was surprised to see it still floating around 20 years later.

However, I certainly will attempt a sequel some day. I would like to see how far I could take an AIF with the power Inform allows you. It would be exciting to be able to tackle a project with none of the limitations I had in 1989 and see what might result.

IE: We would like to thank Mr. McWilliams for taking the time to answer our questions, and I for one hope that the sequel sees the light of day. It would have a good "full circle" feeling to it, if nothing else, to revisit WUZ twenty years later. ◆

Dear Mortal Men and Women,

When I left my story last, I had just made a match between Eric and Sonia, who had commuted on the same train for months, with him lusting after her the whole time, and never meeting. In exchange for helping him meet Sonia, I had made Eric agree to share their first sexual encounter with me, either through an invitation to join in, or to videotape it, or to give me a description written by each of them.

For the next couple of weeks, we saw each other every morning on the train. Eric seemed increasingly at ease with Sonia, and increasingly more nervous around me. Their relationship was entering that giddy stage. But clearly, Eric felt he was getting closer to sleeping with Sonia, which created an awkward

problem for him. One Friday morning when Sonia was not there, I had a chance to discuss it with him. He admitted straight away that he hadn't been able to make himself talk to Sonia about my request. He also said that he felt like he'd made a deal with the devil, at which I laughed merrily.

"But I don't understand what you want with our sex life," he protested. "Collecting first encounters, as you call it, sounds really creepy."
"I can see how you'd feel a little creeped out, but you have nothing to worry about. You see, I'm trying to write a self-help book about sex. One chapter is designed to help people make the most of the first time they make love with a new partner. This is just research." Now that wasn't true, though it definitely struck me as a good idea - and is a chapter (or an entire book) that would be worth writing.
He looked at me askance, not sure whether to believe me – or maybe he was thinking about asking me for a draft of that chapter.

"I see you're having a hard time with this," I said, acting frustrated with him. "Do you want my help?"
He nodded. "Give me your phone."
I dialed Sonia's number. "Hello Sonia, it's Veronica… yeah, from the train… fine, how are you?... No, we weren't worried about why you're not here. I'm here talking to Eric…It's so cute, he just keeps talking about how great you are and how happy he is… No, I'm using his phone because he has something important he needs to ask you…. OK, here he is."
"That's help?" he whispered to me as he took the phone.
"You'll be fine," I whispered back. "Remember, it's my mishogas, not yours."
He nodded and spoke into the phone: "Hi sweetie…" [He called her 'sweetie.' How cute is that.] After a moment of preliminary chat, he took a deep breath, squeezed his eyes shut and dove in. "Veronica—and this is going to sound insane, she wants us to share the first time you and I—um—er—make love—with her ... I know it is… Well, she said she would either want to be there in person, or for us to take a video, or for us to each write a description of it and give it to her… No, she says she's collecting 'first encounters' for a self-help book… [here he raised one eyebrow and glanced at me over the top rim of his wire-framed glasses] No, I'm not sure I believe that, either… well, um, it was on the morning of the first day she introduced us… I'm sorry I didn't tell you before, but I was kind of floored by her request… What? You don't? You are?! [here he covered the phone and whispered to me, excited, "she's into it!"] … Tonight? …Well, okay… can I take you out for dinner first? … Seven o'clock.... OK. … I'll see you then… Me too. Bye."

"That sounded like it went well."
"Amazing. Again, I don't know how you do it. She wants to do it tonight and for us to write descriptions for you."
I just smiled. "Congratulations, Eric. I can't wait to read how it goes."
"Me neither. I mean, I can't wait to see how it goes and to write it."

* * *

The following Monday morning, the two of them got on at his station and then sat down beside me on the train. Blushing madly, and trying not to make eye contact with one another, they each handed me a sealed envelope. I thanked them briefly and tucked the envelopes into my purse.
"So you two are obviously still together after this?" I said, knowing the answer already.
"Better than ever," replied Sonia, grinning. "This is the least we could do for you after you helped us get together."

"Have you read each other's description?" I asked.
They both shook their heads.
"Really? Why not?" I asked.
"Well, you see, knowing that we were going to write descriptions already made us a little self-conscious, I guess it felt like we were being watched," Sonia replied. "I think that made us less confident. Plus, we were just feeling each other out. To put it simply, I'm not ready to know the truth about how he thought it was."
"Me too," Eric added simply.
"We're going to…uh…do it a couple more times before we read them," she said.
"Good idea," I agreed. "And I can't wait to read these."
"Can we see this collection of yours?" Sonia asked.
"I might be able to arrange that," I responded.

I'm waiting until next month to begin sharing what they wrote, but I have a little time left, so I thought I'd describe this collection of initial sexual experiences. I have several thousand letters, many of them many hundreds of years old; some are a few paragraphs long and others are novel length. Some of them were even published and became famous books. Nabokov was a contributor of mine, as were Anaïs Nin, Salman Rushdie and John Updike. In those cases where I was invited to participate I wrote the descriptions myself and added them to the collection. I have many dozens of these, as well as a quite a few video tapes. It's a collection of incalculable value, in terms of its folkloric worth, true, but it's also a great rejuvenator for me: whenever I doubt my value or power as Goddess of Love, I read these stories and they give me a new lease on my relationship with the world. But more than that I believe the collection has great transformative potential. Each story contains a basic, core truth that the Furies have no power to counter. I believe that it can help purify the sexual morality of the world – not to disinfect it, but rather to help us understand the essence, the wonderful exploratory, personal nature of sex and love. At least I hope it can.

Next month I will present one of the letters from Eric or Sonia.

Until then, I wish you all wonderful love,

Aphrodite

NPCs are some of the hardest things to program in a game, but they are also one of the most important aspects in AIF, and IF in general I would say. Because of this, you will probably notice a good many of these article having something to do with their creation and actions. This month we will take a look at how to get a character to join the player and move around with him. Here was the assignment for the month.

The player starts alone in a room. Soon he meets another character and asks him/her to join him on his fantastic quest. From that point on, the character should follow the player from room to room. There should be a way for the player to tell the new party member to stay put while he leaves the area, and then be able to pick him up again (not literally of course) later. The new party member should also respond to orders by the player to do things, but only once he has joined the party. As an example, put in handling for the new party member to pick something up and carry it, then to hand it to the player when asked.

TADS 2 Segment by A. Bomire

This month's "Coder's Corner" involves other characters, which is always the trickiest part of any A/IF game. For the most part, characters stand around like store mannequins unless you (the author) take care to have them actually do something. Make use of the timing features we discussed last month is a good start by having them do or say something now and again. This month, we will cover having characters move on their own. In this case, a character will join the player's "party", follow the player around and even take orders from the player. Most of the stuff we will go over here can be found in the TADS manual, Chapter 10 – Advance TADS Techniques. That section goes over this exact scenario – having an NPC respond to player commands and follow the player around.

We'll start by designing a scenario to illustrate this problem. In this example, the player is part of a military commando squad, dropped off in some remote jungle with the mission to infiltrate the enemy's position. During the drop, the squad and their supplies were separated. The mission: find your squad-mate, and the supplies (explosives) and then make your way to the rear entrance to the building. Once there, blow a hole in the side of the building with the explosives and enter the building.

To make it simple, there will only be 4 rooms in this example. The first is where the player starts, the second is where the player joins up with his squad-mate, the third is where the supplies landed, and the fourth is where the rear entrance is located. In an actual game, these rooms would be well defined, but for our purposes simple descriptions will suffice:

```
startroom: room
  sdesc = "Somewhere in the Jungle"
  ldesc = "You find yourself in a tangled jungle. Trees and
   undergrowth block your vision and movement in almost
   every direction, except to the north where you see a
   narrow game trail. "
  north = GameTrail1
;

GameTrail1: room
  sdesc = "Along the game trail"
  ldesc = "You are following a winding, narrow game trail through the thick
    jungle. It is hard to see much through the thick trees and undergrowth,
    but you can follow the trail north and south. "
  north = GameTrail2
  south = startroom
;

GameTrail2: room
  sdesc = "End of the game trail"
  ldesc = "The game trail you have followed seems to disappear at this
   point, although you can pick it up again to the south. Fortunately,
   however, the thick jungle thins enough here for you to make out the
   rear entrance to a building to the east. "
  south = GameTrail1
  east = RearEntrance
;

RearEntrance: room
  sdesc = "Rear Entrance"
  ldesc = "You are standing at the rear entrance to what your intelligence
     section has told you is the location of the enemy. To the west, you
     can see the thick jungle, while a metal door blocks your entrance
     to the east. "
  west = GameTrail2
  east = RearDoor
;
```

Okay, so that is our four rooms. Again, in an actual game you would want to include descriptions of the tress, undergrowth, maybe even describe the path along which you pass. But for our simple purposes this will do. Now, our entrance is blocked by a metal door, which we will have to bypass. So, let's create one. The door will be closed and locked, with no possibility of being unlocked by the player. He will have to blow it up to pass. There are quite a few ways of doing this, but my preferred method is to create a door which requires a key, and then to put the key into the *nil* location (nil = no where). You may be tempted to simply set the *mykey* property to *nil*, but as this tells TADS that there is no key for the door. However, this simply allows the player to unlock the door at will. By creating a key and placing it out of the player's reach, then the player cannot unlock the door:

```
RearDoor: doorway, keyedLockable
   location = RearEntrance
   sdesc = "metal door"
   ldesc = "This solid metal door may be rusted and unused, but
    it is still solid enough to prevent you from entering the building. "
   noun = 'door'
   adjective = 'solid' 'metal' 'rusty' 'unused'
   islocked = true
   isopen = nil
   mykey = RDKey
;

RDKey: keyItem
   sdesc = "rear door key"
   location = nil
;
```

An alternative to doing it this way would be to override the **verDoUnlock** method for the door. Either would work; this is simply my preferred method for a door which cannot be unlocked. For more information on using *keyedLockable* and *mykey*, please consult the in-code documentation within ADV.T, or the TADS manual Appendix A.

Okay, that's the setting, now let's create our character. Any character will do, but because this is AIF (and we live in an enlightened age) our character is going to be female: Private Emily Parts. To keep this section brief and to the point, I'm going to skip defining and describing all of her interesting bits, but please consult this newsletter's series on "Programming Erin" for more information along those lines if you are interested. And now, a definition of Private Parts:

```
Emily: Actor
  location = GameTrail1
  sdesc = "Private Parts"
  thedesc = self.sdesc
  adesc = self.sdesc
  ldesc = "Private Emily Parts is new to your squad, having just
    finished her training. From what you've seen so far, she seems
    a competent, and pretty, addition to your squad. Her unflattering
    uniform does its best to hide her charms, but you can still
    see hints of her well-shaped form. "
  noun = 'parts'
  adjective = 'private' 'emily'
  actorDesc = "Private Emily Parts looks nervous, but awaits your orders. "
  isHer = true
;
```

Now, Emily is a raw addition to your squad – she won't do anything on her own. You will need to tell her to follow you around, stay put when you want her to, pick up and carry things, give things to you, etc. TADS has already created a verb that allows the player to follow other characters, *followVerb*, which we will use to have Emily follow the player. When Emily is following the player, a "flag" will be set that tells us that she is in follow-mode. "Flag" in this case is a simple boolean (true/false) property that we'll create for her, which I'll call *isFollowing*. To have her stay, we'll create another verb called *stayVerb*.

```
stayVerb: deepverb
  sdesc = "stay"
  verb = 'stay'
  action(actor) =
  {
    if (actor = Me)
     "You don't need to tell yourself to 'stay'. If you don't
     want to move, then you don't have to! ";
    else if (actor = Emily)
    {
      if (Emily.isFollowing)
      {
       "\"Private! Hold this position!\" you command Emily.
       \b\"Yes sir!\" she responds. ";
       Emily.isFollowing := nil;
      }
      else
       "Private Parts is not following you! ";
      }
   }
;
```

In a real game, we would also include else-conditions for other actors, but with just Emily and the player the above would work fine. As you can see, all this really does is set the *isFollowing* flag for Emily to *nil* so that she stops following the player – if she is following him. Otherwise, we alert the player to the fact that she isn't following him.

When the player tells Emily to "follow me", "Me" becomes the direct object of the *followVerb*, with Emily as the actor. So, we will need to make sure to define **verDoFollow** and **doFollow** for the player character and set them up to correctly alter Emily's *isFollowing* property:

```
modify Me
  verDoFollow(actor) =
  {
     if (actor = Me)
       "You can't really follow yourself around. ";
     else if (actor = Emily)
     {
        if (Emily.isFollowing)
         "Private Parts is already following you! ";
      }
   }
   doFollow(actor) =
   {
        "\"Private, come with me!\" you command Emily.
        \b\"Yes sir!\" she responds. ";
        Emily.isFollowing := true;
   }
;
```

We're getting there! Now, we need to alter Emily's definition. When the player gives a command to an actor, TADS checks the *actorAction* method defined for that actor to verify that the actor will actually follow commands. By default, all actors are set to ignore player commands. We will need to modify Emily's *actorAction* to have her listen to the player – at least for the "follow" and "stay" commands. To be thorough, we should also put in a verification method for *followVerb* in case the player asks Emily to follow herself:

```
Emily: Actor
  ...insert here her previous definition - I am only going
```

```
    to include the additions
    isFollowing = nil
    actorAction(v, d, p, i) =
    {
      if (v = followVerb or v = stayVerb)
        return true;
      else pass actorAction;
    }
    verDoFollow(actor) =
    {
      if (actor = Me)
        "You're in charge! She's supposed to follow you! ";
      else if (actor = Emily)
        "You have a brief vision of Emily spinning in a circle as she
         attempts to follow herself. You chuckle to yourself, but don't follow
         through on the command. ";
    }
;
```

To have Emily follow the player around, we'll use a technique from last month's "Coder's Corner" - we'll set up a daemon for Emily. Every turn, we will check to see if Emily is in the same room as the player (checking Me.location). If she isn't, and she is supposed to be following the player, we will move her to the player's location.

```
Emily: Actor
  ...again this is just the new stuff.
  actorDaemon =
  {
    if (Emily.isFollowing and Emily.location <> Me.location)
    {
      "\bPrivate Parts follows you. ";
      Emily.moveInto(Me.location);
    }
  }
;
```

Now, this is a relatively simple implementation. If our game was full of chairs, beds, etc. then we would need to modify the above to make sure that she only followed the player from room to room, and not in-and-out of chairs, beds, etc. This could be done by verifying that the player's location is not a *nestedroom* first, or any number of other techniques. But, this will work for us.

We need to initialize this daemon. We could initialize it at the beginning of the game and have it constantly running in the background – after all, Emily won't follow the player until he meets her and gives her the "follow me" command, so this is a valid technique. However, I'm going to use a TADS room property called *firstseen*, which is called when a player first enters a room – and only when he first enters the room. We'll use it to describe meeting up with Private Parts and to kick off her daemon:

```
GameTrail1:
  ...add the following to our current room definition
 firstseen =
 {
   "\bAs you creep along the game trail, you are greeted by a familiar voice.
   \"Oh sir! Thank goodness you've come along! I think I'm lost!\" From the
   edge of the jungle you see Private Emily Parts, a new recruit who has
   only recently joined your squad, emerging from the undergrowth. She looks
   relieved to see you, and a bit nervous about being in the jungle alone. She
   hastily composes herself, squaring her shoulders and facing you. \"Awaiting
   your orders, sir!\" she says. ";
   notify (Emily, &actorDaemon, 0);
 }
;
```

For more information on using *firstseen*, see the in-code documentation within ADV.T, or the TADS manual, Appendix A. Upon testing, you'll see that the above is displayed after the regular room description, and after listing Private Parts as an object/character within the room. You can alter this so that it is displayed before listing Emily as being in the room by altering the *ldesc* property of the room into a method, like so:

```
ldesc =
{
   insert here the regular room description

   if (not self.isseen)
   {
     insert here the contents of firstseen from above
   }
}
```

Either technique will work just fine.

Now that Private Emily is following the player's command, let's have her do something useful. Using the techniques we learned above, we'll alter her *actorAction* method to have her respond to "take/pick up" things and to also "give" them to the player. We'll place some explosives into the room just before the two commandos come upon the building, and allow the player to "set" the explosives (by creating a *setVerb*), and we'll disallow having Emily set the explosives (she's too new, she doesn't know how).

First, the *setVerb*:

```
setVerb: deepverb
  sdesc = "set"
  verb = 'set'
  doAction = 'Set'
;
```

Next, the explosives:

```
explosives: item
  location = GameTrail2
  sdesc = "explosives"
  adesc = "some explosives"
  ldesc = "These explosives were part of the supplies that were dropped into
     the jungle with your squad. "
  noun = 'explosive' 'explosives'
  verDoSet(actor) =
  {
   if (actor.location <> RearEntrance)
     "Setting the explosives here would not do you any good. ";
   else if (actor = Emily)
    "Private Parts looks at the explosives, her eyes widening.
    \"I..I can't sir! I don't know how!\" she informs you.
    What are they teaching recruits these days! ";
   else if (actor = Me and self.location <> Me.contents)
     "You don't currently have the explosives. ";
  }
  doSet(actor) =
  {
    "You set the explosives carefully around the locking mechanism of
    the door - using enough to damage the lock, but not enough to alert
    any guards inside (hopefully). You set the charge, and back away,
    motioning Emily away from the door as well. With a satisfying
```

```
        WHUMP, the charges blow. When the smoke clears, you can see that
        the door is now open! ";
        self.moveInto(nil);
        RearDoor.isopen := true;
        RearDoor.islocked := nil;
      }
;
```

You'll note that I am using the TADS built-in function *isCarrying* to determine whether or not the player is holding the explosives.

Next, we'll alter Emily's *actorAction* method to allow her to take and give objects:

```
Emily: Actor
  ...I'll just describe the modifications to  actorAction
  actorAction(v, d, p, i) =
  {
     if (v = followVerb or v = stayVerb or v = takeVerb
       or v = giveVerb or v = setVerb)
       return true;
     else pass actorAction;
  }
;
```

Last, but certainly not least, we need to alter the definition of *RearDoor* so that it actually leads somewhere – now that we can open it:

```
RearDoor: doorway, keyedLockable
   ...this is in addition to the previous definition
 doordest =
 {
   "Together, you and Emily enter the building - sewing desctruction and
    mayhem along the way. You have a wonderful adventure which, alas,
    is not detailed here. When finished, you exit the building. ";
    return nil;
  }
;
```

And that should do it! I hope that this was helpful to any potential authors out there!

TADS 3 Segment by Knight Errant

You're in luck this month, because the topic is NPCs, which is one area that TADS 3 makes quite easy. We've gone over defining maps and NPCs before, so I'll skip right into NPC activities this time. First, we need to set up a way to get the NPC, in this case, Erin, to join the character's group. The base libraries provide actorStates for this purpose. Depending on the exact behavior we want, we have an AccompanyingState (where the NPC follows the player around) and a GuidedTourState (where the player follows the NPC around). In this case we want an AccompanyingState.

```
+ womanFollowing : AccompanyingState
  specialDesc = "Erin is standing beside you. "
  stateDesc = "She's standing beside you. "
  accompanyTravel(leadActor, conn)
    { return leadActor == gPlayerChar; }
;
```

This goes after the definition of Erin. When Erin is following the player, the specialDesc is appended to her in-room listing

and stateDesc is appended to the end of her description. Now, once she's in the AccompanyingState, she will follow the player wherever he may go (although you'll need to do some special coding if you're going to use nested rooms such as vehicles or beds). Now, in order to get her into the AccompanyingState, we need to call setCurState(womanFollowing). Let's put it in a AskTopic.

```
+ AskTopic @ woman
    topicResponse
    {
        "<q>You're cute, want to come along with me?</q> you ask.<.p>
        <q>Sure!</q> she replies.";
        woman.setCurState(womanFollowing);
    }
;
```

Now when you ask her about herself, she begins to follow you. If you want her to stop following you, we need to setCurState to something other than womanFollowing. If we have other special handling for the NPC, we'd probably use another ActorState. However, since this is a simple demo game, we'll just setCurState to nil (in TADS 3, nil is a key word meaning "nothing" or "undefined"). Since the player is telling her to stop, we'll put it in a TellTopic.

```
+ TellTopic @ woman
      topicResponse
      {
            "<q>Wait here until I come back.</q> you say.<.p>
            <q>Ok.</q>";
            woman.setCurState(nil);
      }
;
```

Of course, in an actual game you'd have to check to make sure that Erin is currently following you before you tell her to wait, and make sure she's not following you before you ask her to come with you. That's easily enough accomplished with AltTopics, look here for more details: http://www.tads.org/t3doc/doc/tourguide/alttopic.htm

Right now Erin just follows us around like a lost puppy, it's time to put her to use. Since the AccompanyingState is a type of ActorState, all the usual features apply. That means that whichever conversation topics exist in the AccompanyingState are active when the state is active. This can also include orders you give to the NPC. The standard adventure library has CommandTopics which are intended to be used for controlling NPCs. However, they don't have all the features we would need for controlling NPCs. To fill this void, Eric Eve has developed an extension called TCommandTopic, which is found in the Libraries/Extensions folder of the TADS 3 install. Here's an example of how to use it:

```
+ TCommandTopic @TakeAction
  matchDobj = dildo
  topicResponse
  {
     "<q>Erin, would you <<cmdPhrase>>?</q> you ask.\b
      <q>Sure!</q> she replies.";
      nestedActorAction(getActor, Take, cmdDobj);
  }
;
```

In this case, the TakeAction and matchDobj parameters means this topic will only match a Take action directed against the dildo object. matchDobj can be either one object or a list of objects. cmdPhrase is a property that takes the phrase of your command (">erin, take dildo") and turns it into something that the PC might actually say ("take the dildo"). Finally, the nestedActorAction is the part that actually has the NPC take the object. We can also use a similar TcommandTopic to have her give the dildo to the PC.

```
+TCommandTopic @GiveAction
   matchDobj = dildo
   topicResponse
```

```
        {
          "<q>Hand that dildo to me.</q> you insist.\b
           With a wicked smile, she hands the dildo to you.";
            nestedActorAction(getActor, Give, cmdDobj, gPlayerChar);
        }
    ;
```

The only real difference here is that nestedActorAction also has an indirect object, the PC.

Of course, these are just the basics of NPC interactions in TADS 3. TADS 3 also allows NPCs to initiate actions or conversations with Agendas and InitiateConversationTopics. I highly recommend the TADS 3 Tour Guide to look into these in more detail.

Inform 6 Segment by 'trix

Hello Ratfans.

This week we're implementing party members, for which we'll be writing new actions, defining new verb grammar, and implementing NPC orders.

For a start, here's a couple of rooms and an NPC.

```
Constant STORY "Party robot";
Constant HEADLINE "^Coder's Corner, March 2009^";

Include "Parser";
Include "VerbLib";

! There was a slightly different TwoDoor implementation in
!  Programming Erin, if anyone remembers
Class   TwoDoor
  with  locations, directions,
        found_in
        [;
           return (location==self.&locations-->0
                   || location==self.&locations-->1);
        ],
        door_dir
        [;
           if (parent(self)==self.&locations-->0)
              return self.&directions-->0;
           return self.&directions-->1;
        ],
        door_to
        [;
           if (parent(self)==self.&locations-->0)
              return self.&locations-->1;
           return self.&locations-->0;
        ],
  has   static door;

Object  lab "Laboratory"
 with   description "An electronics lab. Junk is strewn over
           various desks, and a small closet lies to the
           south.",
        s_to closetdoor
 has    light;
```

```
Object  -> labdesks "desks"
 with   name 'desk' 'desks' 'table' 'tables' 'wood' 'wooden',
        description "Plain wooden desks, strewn with wires,
           circuit boards and other electronic junk.",
 has    scenery supporter pluralname;

Object  -> -> labjunk "junk"
 with   name 'junk' 'electronic' 'wires' 'circuit' 'boards',
        article "some",
        description "Wires, circuit boards, some other kind
           of stuff, all strewn haphazardly around the lab.",
        before
        [;
           Take, Remove:
              "None of the electronic junk looks useful.";
        ],
  has   scenery;

TwoDoor -> closetdoor "closet door"
  with  name 'closet' 'small' 'door',
        locations lab closet,
        directions s_obj n_obj,
        when_open "The closet door is open.",
        when_closed "The closet is closed."
 has    openable;

Object  closet "Closet"
  with  description "A tiny space, with barely room to stand, and a
           door to the north.",
        n_to closetdoor,
  has   light;

Object  -> frybot "Frybot"
  with  name 'frybot' 'fry' 'robot',
        description "Frybot is a (very) simple robot.",
        initial "Frybot stands here, watching you expectantly.",
        each_turn
        [;
           if (self has concealed)
           {
              give self ~concealed;
              "^In the dim light of the closet, a robot turns towards
              you and says, ~Greetings, human. I am Frybot. Beep.~";
           }
        ],
  has   animate proper concealed;

[ Initialise;
   location = lab;
];

Include "Grammar";
```

Now if we want to get the NPC to follow the PC, a "follow" action might be a good idea. Stick this at the end of your code:

```
[ FollowSub;
    if (noun==player)
       "You make a mental note to stay with yourself at all times.";
    "There's no need for you to follow ", (thatorthose) noun,".";
];

Verb 'follow'
   * creature                          -> Follow;
```

This defines an action Follow, matching commands of the form "Follow <someone>", and executing the FollowSub routine when performed. Note that FollowSub doesn't do anything interesting: that's what happens when the player tries to follow someone. The point of the action is that NPCs can be ordered to follow someone.

So how do we make Frybot obey a command to follow? This is done by giving him an orders property. This is executed whenever a command has been parsed as an order issued to the NPC.

```
! In the Frybot object
        orders
        [;
          Follow:
             if (noun~=player)
               "Frybot looks confused. ~My mighty logic circuits cannot
               comprehend your inferior human reasoning. Beep.~";
             "~Affirmative, Earthling. I am a followmotron. Beep.~";
        ],
```

Now Frybot will give a reasonably sensible response to an order to follow, but he won't actually follow. To make him follow, we need something that will be called every turn, particularly when Frybot is not in scope. What did we learn last month? That's right, nothing. But the solution is to use a daemon.

```
        orders
        [;
          Follow:
             if (noun~=player)
               "Frybot looks confused. ~My mighty logic circuits cannot
               comprehend your inferior human reasoning. Beep.~";
             StartDaemon(self);
             "~Affirmative, Earthling. I am a followmotron. Beep.";
        ],
        daemon
        [;
           if (self notin location)
           {
              move self to location;
              "^Frybot follows you, occasionally beeping.";
           }
        ],
```

Now whenever the player moves to another room, Frybot's daemon (which is run shortly afterwards) will notice he's not there and move him into the new location.
Now to get him to stop. I'll be subverting the existing action Wait for this purpose, and adding some more grammar to it.

```
Extend only 'wait'
     * 'here'                          -> Wait;
Verb 'stay' = 'wait';
```

Now the commands "Wait", "Wait here", "Stay" and "Stay here" will all match the Wait action.

Add this to Frybot's orders property:

```
        Wait:
          StopDaemon(self);
          "~Affirmative, flesh creature,~ says Frybot. ~I will
          remain here and wait for ye. Beep.~";
```

Now it's all very well having Frybot following the PC around, but can he do anything useful? Can he, for instance, pick up a giant gold statue of a duck? It seems unlikely.

```
! Put this in the lab
Object  -> statue "giant gold statue of a duck"
  with  name 'giant' 'gold' 'statue' 'of' 'a//' 'duck',
        description "Once you've seen one giant gold statue of a duck,
           you've pretty much seen as much as you need to.",
        parse_name
        [ i j w;
           for (:: ++i)
           {
             w = NextWord();
             if (~~WordInProperty(w, self, name)) break;
             if (w~='of' or 'a//') j = i + 1;
           }
           return j;
        ],
        before
        [;
          Take: "Perhaps I didn't mention how giant and gold this
            particular statue of a duck is, but you certainly can't
            pick it up.";
        ]
;
```

The parse_name routine is not essential nor relevant, so feel free to miss it out.

To have Frybot follow an order to take the statue, we intercept the Take command in his orders property. We should check what he's being ordered to take, to make sure he responds appropriately.

```
                Take:
          if (noun==self)
             "~Error: stack overflow,~ says Frybot. ~Beep.~";
          if (noun==player)
             "~I am not a transportobot,~ says Frybot.";
          if (noun~=statue)
             "~Negative, bossy meat creature,~ says Frybot.
              ~I have no need of your primitive ", (name) noun,".~";
          if (noun in self)
             "~Error: I am not a juggleotron,~ says Frybot.";
          move noun to self;
          "~Affirmative, mammal,~ says Frybot. ~I am a nature-bot. I
          will collect a specimen of type Giant Yellow Metal Bird.~
          ^Frybot picks up the giant gold statue of a duck.
          ^~Beep.~";
```

This should illustrate a style of programming that's a good way to write in Inform 6, which is to go through and deal with every case that isn't what we want, until all that's left is what we want. If the noun is something other than the statue, give an appropriate message and reject it. If the statue is already carried, give an appropriate message and reject it. The remaining possibility is that

the noun is the statue and that Frybot is able to take it. (If there was a locked glass case that it could be inside, we would have had to deal with that as well, but I do my best to keep these things simple.)

When Frybot is carrying a giant gold statue of a duck, it seems only right that his description should reflect this. So lets amend his description property thus:

```
description
[;
   print "Frybot is a (very) simple robot.";
   if (statue in self)
     print " He's carrying a giant gold statue of a duck,
        and making it look easy.";
   new_line;
],
```

If he were able to carry a variety of things, you would have to write something that could list all the things he was holding, and you'd probably want to use the library routine WriteListFrom, something like this:

```
  if (child(self))
  {
     print " He is carrying ";
     WriteListFrom(child(self), ENGLISH_BIT);
     print ", and making it look easy.";
  }
```

The last thing I'm supposed to include this month is getting the NPC to give an item to the player. Consistently enough, we do this by adding a Give interception into Frybot's orders routine, so the whole orders routine should now look something like this:

```
orders
[;
  Follow:
     if (noun~=player)
       "Frybot looks confused. ~My mighty logic circuits cannot
       comprehend your inferior human reasoning. Beep.~";
     StartDaemon(self);
     "~Affirmative, Earthling. I am a followmotron. Beep.";
  Wait:
    StopDaemon(self);
    "~Affirmative, flesh creature,~ says Frybot. ~I will
    remain here. Beep.~";
  Take:
    if (noun==self)
       "~Error: stack overflow,~ says Frybot. ~Beep.~";
    if (noun==player)
       "~I am not a transportobot,~ says Frybot.";
    if (noun~=statue)
       "~Negative, bossy meat creature,~ says Frybot.
        ~I have no need of your primitive ", (name) noun,".~";
    if (noun in self)
       "~Error: I am not a juggleotron,~ says Frybot.";
    move noun to self;
    "~Affirmative, mammal,~ says Frybot. ~I am a nature-bot. I
    will collect a specimen of type Giant Yellow Metal Bird.~
    ^Frybot picks up the giant gold statue of a duck.
    ^~Beep.~";
  Give:
    if (second == self)
```

```
            "Frybot beeps disconsolately.";
        if (noun notin self)
            "Empty-handed, Frybot mimes.";
        if (second ~= player)
          "Frybot tries to give ",(the) noun," to ",(the) second,
          ", but ", (the) second," doesn't appear interested.";
        move noun to player;
        "~Affirmative, animal. I will transfer the requested item(s). Beep.~
        ^Frybot thrusts ", (the) noun, " into your expectant hands.";
    ],
```

The Give action takes two objects, which both need to be checked: noun is the object being given, and second is the recipient. Finally, because the PC has just been handed a giant gold statue of a duck that has already been established as uncarryable, there ought to be some repercussions for him/her. I put this each_turn property on the statue.

```
    each_turn
    [;
      if (self in player)
      {
        deadflag = 1;
        "^You are crushed under the weight of a giant gold statue
        of a duck. You always suspected it would end like this.";
      }
    ],
```

That's it from me. Happy writing, mammals.
Beep.

Inform 7 Segment by Dudeman

This week's topic of choice is to code followers who will follow the player around when asked and stay put when told to do so. This is a very useful thing in IF and luckily is not that complex to do using Inform 7. The first thing we have to do is establish a way to define a way to tell the computer that a person is supposed to follow another person. This is the perfect task for a "relation" in I7 (see chapter 13 in I7 documentation for more info) which deals with different relationships between two things (in this case the relationship between follower and followed). We can define this relationship with a few simple lines of code;

```
Following relates various people to one person.

The verb to follow (he follows, they follow, he followed, he is followed, he is
following) implies the following relation.
```

The first line of this code tells inform 7 that the following relationship can only apply to people and more than one person can follow another person. The second line gives inform the syntax we wish to use when we define the relationship in future code. With this, we have the first tool needed to create follows, but we also need a way for this relationship to be applied in game. This can be done with a simple action covered numerous times in past issues:

```
Following is an action applying to one thing.

Understand "follow [something]" as following.

Persuasion rule for asking someone to try following: persuasion succeeds.

Carry out someone following the player:
now the actor is following the player.

Report someone following the player:
say "[The actor] is now following you.".
```

Note: For the purposes of this simple demonstration, I am assuming that only the PC can be followed so that an NPC can't follow another NPC or the player follow an NPC. Though this can easily be done using the same type of code used here and just changing the rules accordingly.

Now moving on, the persuasion rule used above just says that the player can tell other people to follow and they will follow. Next, the line "now the actor is following the player" above is the line which tells I7 that when the following action succeeds, we want to person performing the action to follow the player. However, we have yet to define what following means and what we want done with it. To do this, we want a rule that will always check to see if the player has moved into another room and have any people following them to move as well. This can be done with an "every turn" rule such as;

```
Every turn while a person is following the player:
repeat with follower running through people who are following the player begin;
if the location of the follower is not the location of the player begin;
move the follower to the location of the player;
say "[The follower] follows behind you.";
end if;
end repeat.
```

This repeat phrase (Chapter 11.10 of the documentation) will run through all the people currently following the player and more them into the same room as the player if they are not already which is exactly what behavior we are looking for.

Next, we have to write a few simple "check" rules to stop the player from doing things you don't want them to do like trying to follow things that aren't people, telling someone to follow when they are already following, and stop the player from following someone else.

```
Check someone following the player:
if the actor is following the noun, say "[The actor] is already following you."
instead.

Check following someone:
say "You don't feel any need to follow anyone right now." instead.

Check someone trying following something:
if the noun is not a person, say "Inanimate objects can't be followed." instead.
```

That should pretty much cover the code for getting NPC's to follow the player around when he moves, but we also want to create a similar action and set of rules to tell the NPC to stop following the player.

```
staying is an action applying to nothing.
Understand "stay" or "wait" as staying.

Persuasion rule for asking someone to try staying: persuasion succeeds.

Carry out an actor staying:
now the actor is not following the player.

Report an actor staying:
say "[The actor] stops following you.".

Check an actor staying:
if the actor is the player, say "You aren't following anyone." instead;
if the actor is not following the player, say "[The actor] isn't following you
right now." instead.
```

and with this we pretty much have the basis of a system to have NPC's follow and stop following the player. One last thing we want to cover is allowing the player to issue out commands to an NPC and have them follow them when they are following the

player when they otherwise wouldn't. This can be done with a simple blanket persuasion rule such as;

```
Persuasion rule for asking someone to try doing something:
if the actor is following the player, persuasion succeeds.
```

which tells I7 that all persuasion succeeds if the person asked is following the player. There we have it, we got all the code we need to cover this week's objective. Now all that's left is to create a little scenario to test out the features we just coded. For this week, let's try a good base for a typical AIF scene, picking up a hitchhiker. The player starts out in his house, but upon exiting finds a young girl trying to hitchhike in the street outside his house. Of course, what kind of AIF PC would he be if he didn't offer to let the young, free spirited girl stay with him for the night and guide her back to his room?

```
Your bedroom is a room. "Just your ordinary bedroom, a place to sleep and
occasionally fuck. Not much else. The rest of your house is to the west.".
Your house is west of your bedroom. "Your house isn't that big, but since it is
just you living here it suits your needs fine. Your bedroom is to the east and
the door leading to your front yard is to the north.".
Your front yard is north of your house. "A small grassy front yard, nothing
special. Your house is to the south and the street in front of your house is to
the north.".
Your street is north of your front yard. "An ordinary suburban street. Your
front yard is to the south".

Julie is a woman in your street. "Across the street you can't help but notice a
young girl carrying a large backpack and appears to be trying to hitch a ride."

Before going to your street for the first time:
move the player to your street;
say "'Hey,' you call out to the young girl. 'What are you doing?'

'Just trying to catch a ride out of here. I just turned 18 and I couldn't wait
to get out of my parents house. Even if I don't have another place to stay, I
couldn't stand my prude of parents any longer. They never let me have any fun.'

'Well, if you really need a place to stay. You are more then welcome to crash
with me for awhile,' you offer.

'Really? That would be great. I don't have any money to offer you rent, but I'm
sure we can work [italic type]something[roman type] out that we both can agree
too,' she says suggestively while looking up and down your body.

'Oh, I'm sure we can,' you say calmly while screaming JACKPOT in your head.

'Cool, if you could just show me to your house I can put down my things and we
can talk things over.'" instead.

After going to your house while julie is following the player:
say "After you lead Julie into your house, she sets down her things and drags
you into your bedroom. The two of you make hot, wild love for hours after that
and have repeated the act every night for the last few weeks that Julie has
crashed at your place. You aren't sure when she is planning to leave, but you
are perfectly fine with the arrangement you currently have and plan to enjoy
every moment of it.";
end the game in victory.
```

and there we have an extremely small example of the type of scenario that you can use the following action for, though I will leave it up to all you aspiring authors out there to come up with some more creative, interesting uses for it and I just hope this article will help you do just that.

ADRIFT Segment by BBBen

Okay, this month's task is quite complex, but I think we've reached the point in these coding articles where I don't really need to tell you every little thing about ADRIFT. Therefore, rather than giving you all the details, I'm going to address the more important questions about this topic (in fact, the frequently asked questions) and I'll leave the details up to you.

Let's take a look at the brief for this month by breaking it up into pieces.

First up, we want the character Betty to follow the PC as he walks around.

- One way that you can make characters follow you is by overriding the standard movement commands with tasks. For example, creating a task called " * north * " will override the normal movement north (make sure to include those asterisks, and also add the alternative command " * n * " so that all the different movement commands are covered). Then you go into the actions tab and have the task move your player and anyone you want to follow your player into the room to the north. Make sure the task is repeatable, can only be completed in the right room, and in the description box type "You move north. Betty follows." or whatever (something that reads the same as normal movement commands). After that is the option "Then show description for room", and you should select the room into which they are moving, to complete the effect.

  This technique works well (I used it heaps in Crossworlds Part 2, so if you want an example go and check that out – the password to the game file is "rabbit"), but the main problem with it is it's a little crude, and requires you to do this for every room in which the player will be followed. That's not quite as bad as it sounds as you'll be knocking these tasks together pretty fast once you understand them, but it would be nice to have something more elegant.

- The second technique we could try is to use the ADRIFT generator's in-built character "walks" system. I must admit that I don't really like this system and haven't used it much, as it didn't work in ADRIFT 3.9. If you're using 3.9 DON'T BOTHER trying to use the walks system to make a character follow the player. It just doesn't work. It does work in 4.0, but it's not really necessary, as the next technique I describe to you is, I personally think, better.

  Still, I guess I have to explain it a little. In the character menu go to the "movement" tab and "Add walk". Start movement when task "Betty follow me" is complete; walk can be terminated when task "Betty stop following me" is complete. Add the walk, then set "Where should character move to" to "Follow player". Select "Loop walk when finished".

  I really can't stand by this method too well, as I haven't extensively tested it (and I don't 100% trust it) but there do appear to be some cool options in version 4.0 for making characters behave more autonomously than normal, without adding heaps and heaps of tasks and events to do everything manually. I'd encourage you to try and fooling around with it yourself, and drop me an email if you come up with anything really cool.

- Here's a third option, and this is the technique that I'm going to assume you use, as I'm going to use it for the next task, too. Also, I think this is the best one. Create an integer variable called "bettyfollow" to start at value 0. Then create a task called "betty follow me" or something like that. In the restrictions tab create a restriction that Betty must be in the same room, and in the description tab put in a little description. This task should be repeatable. Finally, in the actions tab, make the task change the variable "bettyfollow" to 1.

  Okay, now create another task called something like "bettyfollowstheplayer", completable in all rooms, and don't bother to give it a description (it won't work for some reason). Make sure it's repeatable, then put in a restriction that Betty must NOT be in the same room as the player. Then you will need another restriction: that the variable "bettyfollow" is equal to 1. Then put in an action that moves Betty to SAME ROOM AS player.

  The next step is to create an event (call it whatever). The event should start immediately, should last between 1 and 1 turns, and should restart as soon as it finishes. Then in the advanced tab, have it execute the task "bettyfollowstheplayer". Now we're done, and you've probably noticed how much simpler that is than the first method.

  To get her to stop following you it's as simple as creating another task called "Betty stop following me" that puts the

"bettyfollow" variable back to 0.

Second we want the follower to respond to commands, for example picking things up and carrying them, as long as the character is following the player.

- This isn't too hard. If you know anything about ADRIFT you'll know getting the character to do things is as simple as putting in a task titled "Betty pick up banana" or whatever. Making sure that they will only respond when they are following you is equally easy if you've used the second method I described above, as you can just make the task require the "bettyfollow" variable to equal 1.

  To have the character pick something up and carry it, just give the task an action that moves the object into the character's inventory. I'm afraid you'll have to do that manually for every object, however, and the same goes for getting characters to hand items over to you, taking items when you give them and putting items back down (if you want to do those things).

Okay, well that wasn't too painful, was it? If you have any further questions you can email me (try to be specific with what you want me to explain) and I'll be happy to help. ◆

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 one game 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.