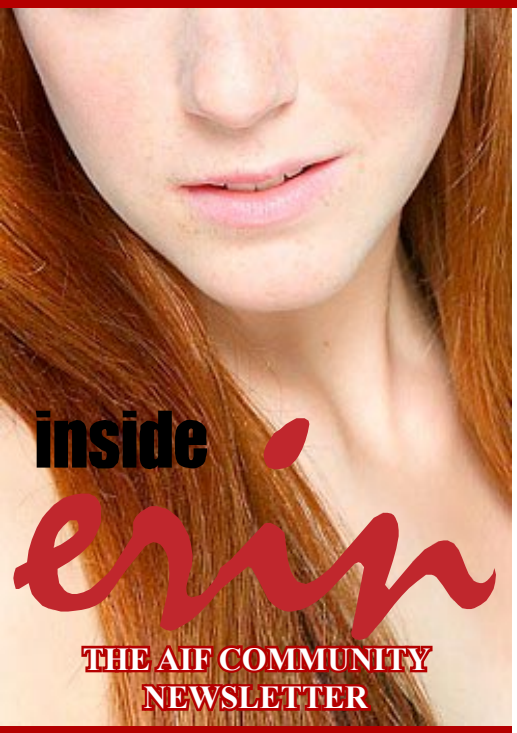

Contents

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.

Something that we have been trying to do for quite awhile now is to come up with a way to get more people involved in the community. What I primarily mean by that is to have more people writing the games that we all enjoy so much. However, even if that is something that you really don't feel you can do, there are other areas where you can make a contribution.

One of the original purposes for the Mini-comp was to try to get new authors involved. The main reason that we had short story comps was to try to get new authors involved, thinking that maybe if someone "got their feet wet" with something like a story, they would then go on to try their hand at a game. Well, that didn't work out all that great, but at least we got some good stories out of it.

This newsletter was downloaded over 500 times last month, and I know that that number is nothing compared to the number of people who are lurking on AIF Archives and AIFGames (not to mention TF Games Site and Hypnopics Collective). Out of all those people, and twenty or so years of history, there have been less than 150 AIF authors – Ever!

So what's my point? Do I have a point? Am I just ranting here? That's partially it, no doubt, but I do have somewhere I going with this, because something interesting (or at least potentially interesting) happened this month.

Continued on page 3

The first thing to report this month would have to be the release of *School Dreams 3* by Goblinboy, as that probably occupied about 90% of the conversation this month. Needless to say, it caused quite a splash. This isn't really the place for game analysis and discussion, so I'll simply report that there was a frenzy of SD3 related activity on the AIF Archive as the community leaped on the much-anticipated game.

Surprisingly, perhaps, there are actually some other things to report on this month. The start of a community-modable game was released with a few pictures built in, the same author/artist created a selection of very well done illustrations for Chris Cole's game, *Camp Windy Lake* (causing a re-release of the game), and another game, *Scandal on the Seven Seas* by Faraday was also re-released. Over on AIFGames.com, Holet wrapped up his Live AIF *The Test* after a long run of posts.

One more topic of discussion has been the immanent deadline for the 2009 AIF mini-comp, which comes up on June 8. It's probably a bit late to be starting a game now if you haven't already, but if you have got a game mostly done then get it finished and tested. Oh, and if you're working on any games that aren't for the mini-comp, *please* don't try to top SD3. Just write a medium-sized game! Seriously, we'll like it! Okay, I'm done, see you later.

Continued on page 3

The one advantage of writing a column like this is it's a free form of therapy. I mean I can babble or bitch about anything I want and unless the editor thinks I've gone too far it will show up in print where no one can answer me back. Luckily I realise the true extent of my power and only use it for good.

Case in point is a chance to inform you about how my month has gone so far.

You see, the first signs of summer have poked their head over in Bearville and a combination of sunny days and pollen have made Nandi very tired and grumpy. For Nandi really is a creature of the night. This has two main effects, first it makes Nandi very grumpy and likely to get upset at posts in forums, and second, it gives me writers block.

Now normally I wouldn't bother you with such trivial matters but, (and here's the crux of the matter) I have other commitments. First I've been working with other people on games, and no I'm not going say who, and whilst they've never given me any pressure to complete the game I just can't help but feel compelled to try and do as much for them as I can.

And then my own games, it seems recently that I've actually become quite popular with a vast array of people wanting me to finish one or more of my games in progress. And because I've foolishly gone and started a lot of games, with more ideas coming at quite a rate, I have more that need finishing than I have hours in the day.

Continued on page 4

May was a rather slow month game wise, though one of the Collective's artists is working on a game!

Benmbedlam has reappeared to let followers of him know that his game called *Rough Landing* is still being worked on, but there will be some delay as he is rewriting a lot of the game...

MadisonX posted a new version of the game Dominated at:
http://www.4shared.com/file/104289287/6258fec5/Dominated_02.html

The Slavemaker game has been updated to version 2.9
http://rapidshare.com/files/231977936/Update10.rar

> "*Lots of bug fixes in this update:*
> *- lesbian training fixed - only happens for lesbian trainers*
> *- a bug where you could use the mansion and not choose wealthy*
> *- typos*
> *- fixes for Ayane for end game and some test and question issues*
> *- fixes for Riesz where demon girl recruiting happened in the wrong place, end game issues, updates for question system, some small graphics tweaks, some text fixes*
> *- Lady Farun's 'Nymph's Tears' quest can also happen for trainers with ruins access now*
> *- fixed meeting Count in the Lakes always had him ignore you*
> *- could get the cock dildo but you were refused to do Dildo actions*
>
> *Changes/New*
> *- new tentacle event - beware of milking on days when the monsters are about*
> *- tentacle events are a little more common for higher difficulty levels*
> *- Lord meetings now use large transparent graphics in main window*
> *- Some meetings with the bounty hunter the same (when you chat in bar)"*

Further discussion on the game is at:
http://hypnopics-collective.net/viewtopic.php?f=11&t=12603&st=0&sk=t&sd=a

Continued on page 4

LFTE, Continued from page 1

It started with a discussion on the pros and cons of modding AIF games. The person spearheading the idea is The High King, and he went so far and to release a "modded" version of *Camp Windy Lake*, the classic by Christopher Cole, and a favorite of many. Personally, I don't think a game should be modded without the authors permission, so it's lucky for us all that Chris chimed in and gave just that, along with some extras that he wanted to put in himself. If you haven't checked out the game, then do. The illustrations added by The High King are wonderful.

That game was good news, but the interesting thing that I'm talking about is a game called *The Shifting City*. The High King wrote this game and released it along with the full code of the game, offering to let anyone who wanted to, mod till their heart's content. The game is written in Inform 7, and as it stands now, it's not much more than bare bones. A few rooms, a few objects, and one character with minimal interaction. Because of this I have a hard time thinking of it as a modding project. To my own thinking, modding happens when you take a (at least mostly) complete game, and add to or change things in the original. Now, it might just be a matter of perspective, or a difference of categorization, but because of the obviously unfinished state of the game, it feels much more like a community project to me, which is not a bad thing at all, quite the contrary.

Community projects have been discussed more than once on the various boards. What normally happens is that someone suggests one, there is some discussion, no one can agree on exactly how it should look, and then the discussion stops and we don't hear anything else about it. Then wait a few months, and someone will suggest one, there is some discussion… you get the point. The main difference here is that instead of going through all that, The High King has started a game and put it out there to be added to by anyone who wants to. In other words, something has actually come of it this time.

Will anyone add to it, or will the game as is stands now, stand like that forever? I don't know. There has been some discussion and arguments about the idea with some people saying that a project like this doesn't stand a chance, because with so many different authors, the end result will be a mess. A hodge podge of conflicting ideas and scenes with nothing holding them together. Do they have a point? You're damn right they do. There is a good chance that that is exactly what will happen, but you know what? I don't see a problem with that. Why? Because at least then it would prove to everyone that this community could get together and produce something as a whole. It doesn't have to be great to prove that, it only has to exist. Once everyone sees that it's possible, who knows what could happen? And even for this game, even if the scenes don't flow together just right, even if there are inconsistencies in plot, characters, what have you, that doesn't mean that the end result (if there ever is an end result) won't be something damn fun to play.

So if you've been thinking about writing a game, this is your chance, one of the best ones you're going to get. This is an opportunity to write something where you don't have to come up with every room, object, character, and command all by yourself. Where you can build on other peoples ideas and code. Where you have a whole community of beta testers to help you out if you get stuck or need help, and where you don't have to come right out and say, "this is the game that I wrote," because if there are mistakes, the other guy did it. Right? :)

So check out the game and see what you think. If you need help with the coding part of it, there are always people around (myself included) who are willing to help. I think that The High King has done a good job of commenting on the code in the game, and I think you'll be surprised at just how much of it you'll understand, even if you've never looked at I7 before.

In other news, the deadline for this year's Mini-comp is just a couple of days away now. Hopefully we will be able to get the games out, and get the voting wrapped up in time to let you all know the results in next month's issue.

As always, thanks for visiting, and until we meet again I wish you all dirty thoughts and sexy dreams. ◆

AIF, Continued from page 1

New games

School Dreams 3: School Dreams Forever by GoblinBoy. Released May 10th 2009 for TADS 3. Continuing the story of the campers from *The Camping Trip*, the PC now has to choose between Mike's sister Molly and his own girlfriend Becky. In a series of dates and other situations, the player will navigate a series of possible events, with many possible outcomes. (All right, you try summarising it!)

Camp Windy Lake v1.4 by Christopher Cole, with pictures by The High King. Released May 24th 2009 for ADRIFT 3.9. This classic game about a bunch of over-sexed teens at "Camp Windy Lake" has been remade with some expanded responses, and all new portrait pictures illustrated by The High King.

The Shifting City by The High King. Released 29th May 2009 for Inform 7. You are a teen boy looking for sex, but you have a strange sense that your world is constantly changing around you. This is a project intended for community development and modding.

Scandal on the Seven Seas v2.0 by Faraday. Released May 31st 2009, for ADRIFT 4.0. This entry into the 2007 AIF mini-comp about piracy and debauchery has been rewritten and re-released according to feedback the author received on the game. ◆

TF Games, Continued from page 2

Basically it's all come to head and I've had to say no, and stop and take time off to recharge my batteries so I can build up the energy to carry on writing and creating games.

If there's a moral to this story, and if there really needs to be one, it's this. When a game creator has gone quiet, there's probably more going on than just not wanting to write anymore for their game (though sometimes it's just that). I've actually been quite lucky on this count with other creators having much more serious real life problems to deal with, and I should hope you all realise that real life should always come before this little virtual one we all inhabit.

Before a brief summary of this month in the TFGamesSite forum, (you knew there was a reason for all this didn't you) I'd like to take this opportunity to thank my ~~therapist~~ editor, Purple Dragon for not only putting up with my erratic writing and posting schedule but actually allowing me to write and then printing this random babble on a monthly basis.

And now back to your regularly scheduled programming.

With another move finally settling in, thoughts have turned to the future of the group. TinaB has been canvassing us on ways to improve the group, over whether we can encourage increased participation, and whether another contest is a good idea. In short, no we can't, and yes we should, respectively. We also had another discussion about the merits of RAGS and the style of the games. Whilst it started to get a little bitchy, it's all settled down and I think we've all come out of it a little better and more respectful for it.

On the game front we had a couple of little dozies in a variety of formats. First in Adrift and then in RAGS we have Fuearye's *Payback*, a nice little linear story told in game format with words and picture. And in a similar vein is Birion's *Daughter of Summer*, a nice story in a new story telling system, Ren'Py. And in a more tradition form is Skunk_Witch's RPGMaker game the *Maid of Wynn*.

And finally in the traditional TGGamesSite / Hypnopics crossovers, Sally has introduced the forum to the pleasure and frustration of *Blue Devils Blood War* (RAGS).

Well I'm off to have a good rest now, feel free to contact me if you want to offer support or even take issue with anything I've said so far. It'll be nice to know that people are actually taking the time to read these random thoughts. ◆

Collective, Continued from page 2

Further discussion on the game is at:
http://hypnopics-collective.net/viewtopic.php?f=11&t=12603&st=0&sk=t&sd=a

Blue Devil appeared and left a note on *Bloods War* for those interested in the game:

> *"Hi everyone! It's been a long time since I last posted here...*
>
> *First of all, I want to thank you all for your comments. As Tina mentioned, I have been working in an upgrade of this game. I started it a long time ago (more than a year) but I have been facing several issues that had delayed it. I had a lot of personal stuff going on (some of it was good, and some... not quite so). And I faced some mayor bugs in the rags system. I came to a point where I couldn't save the modifications I made to the game because a memory issue in rags just crashed the program every time.*

But fortunately Tina got the problem solved and got me a beta version of rags that can actually load the file in the designer and save the changes, so I'm planning to develop the game some further.
As for what I've got so far, it's not playable yet. I haven't even completed a boss plot, the texts are incomplete, it's bugged... there's a lot to do with it before I release it."

And that little tease at the beginning I should tell you all about...

One of the most talented (and I'm not just saying that because she's my Sis) artists on the Collective, VVrayven, is going to try her hand at making a game that has a more female slant to it.

You can follow that discussion in this thread:
http://hypnopics-collective.net/viewtopic.php?f=11&t=14957&st=0&sk=t&sd=a

So not a lot happening this month gamewise at the Collective, but lots of promise for the summer! ◆

School Dreams 3: School Dreams Forever

Review by ExLibris

Game Info:	School Dreams 3: School Dreams Forever
Author:	Goblinboy
Release Date:	May 9, 2009
Platform:	TADS 2
Size:	50.6 MB (with pictures); 2.56 MB (without)
Content:	m/f, m/f/f, m/m/f, m/m/f/f, m/m/m/f, voyeurism, incest, nc (based on the limits you select there can also be bestiality & underage, and the incest can go further) (the daydream sequences also include f/f, mc, bdsm and a mega-orgy!)
Type:	T&AIF, though that doesn't really do it justice.
Length:	Long, and it lends itself to multiple playthroughs
Reviewed:	June, 2009
Extras:	A whole boatload of pictures and multiple endings

Basic Plot/Story

Molly wants to lose her virginity on her birthday. Her brother Mike wants you to be her first (and himself her second). Your girlfriend Becky wants monogamy. What do you want?

SD3 begins a few weeks after the events of SD1. No matter who won or lost the bets in that game, the PC has somehow been saddled with the task of deflowering Molly (Mike's sister) while Mike watches. The game covers four days, climaxing with the night of Molly's birthday party. There are a number of different possible endings, depending on the choices that the player makes.

Overall Thoughts

There is no doubt that this game is going to be the standard by which all other AIF is measured for a long time to come. Thanks to the hundreds of images (a mixture of real pictures and 3D graphics) it is a 50 MB monstrosity. But even without the pictures, it is still the largest TADS AIF ever. You can and will play it for hours, trying to find every last thing. There is just that much content.

Opinion will be divided on the 3D graphics. Personally, I felt that they were generally very good, although there were a few that didn't work (e.g. the cinema ticket girl, who looks like a refugee from IMVU). The pictures do definitely add something to the game, especially as there is less unique text in some of the sex scenes than you might expect.

While the plot is a rehash of SD2, the thing that makes SD3 special is the unparalleled number of options that the player can choose from. Although there is an ostensible goal, the PC is under no real obligation to pursue it and can instead follow any path the player wants. The game doesn't make any path more compelling than another (i.e. there is no 'true' path), although some paths are supported by more content than others. The sacrifice inherent in having multiple paths is that none is going to be as well developed as a single-storyline game would be, but that's a small price to pay for the gameplay experience that SD3 offers.

Puzzles/Gameplay

SD3 doesn't have many puzzles in the traditional sense. What it does have are choices, some more significant that others. These can affect the PC's relationships with the other characters and can sometimes lead to bonus content further into the game. Working out what choices to make to achieve the results you want is the real puzzle of SD3.

Sex

SD3 has a large number of sex scenes, some of them very hot indeed. Each of the three main female characters has three to six minor scenes that the PC can enjoy with them before the grand finale. These scenes are quite small, but each is distinct and they serve to heighten the player's anticipation for what is to come.

Of the major scenes, two (Molly and Becky/Molly) are among the best ever written, and definitely the best illustrated. Each is almost a mini-storyline in itself, which I think makes them a lot more memorable and interesting. The other major scenes (Becky, and especially Alison) are a little disappointing by comparison. While they are still enjoyable, they are noticeably smaller and lack the x-factor that makes the other two scenes so special.

Apart from the three main female characters, the PC has five other potential playmates. However, only two have interactive scenes. The PC's encounters with the other three are simply narrated once they have been triggered. There are also a number of opportunities for voyeurism to be found. However, some of the best sex that the PC can have occurs in his own head. There are nearly a hundred scripted daydreams, a majority of which have pictures, plus an infinite number of randomly generated fantasies. On the minus side, finding all of these daydreams requires repeating the same command over and over, a tedious process that sucks some of the enjoyment out of them.

Technical

For a Goblinboy game, SD3 is comparatively buggy. I noticed a half-dozen or so bugs and glitches during my playthroughs, although only one was really serious. It would be unrealistic to expect no bugs or glitches in a game of this size and complexity, but I think SD3 would have benefited from even more playtesting than it received.

Less easy to be forgiving about are the large number of spelling mistakes and typos, but at least they are easier to fix (and might well have been by the time you read this).

Intangibles

SD3 was a game that I found easy to admire, but slightly more difficult to like. Although the characters were adequate, compared to the epic scale of the rest of the game they definitely felt like the weak point. The minor characters were flat, which isn't necessarily a bad thing, but some bordered on caricature and none were particularly likeable. The main characters were more rounded, but failed to be very compelling.

The descriptions of the game world were also very sparse, the four houses that the PC visits being the worst offenders in this regard. There were one or two details that added some depth to the characters, but overall I would consider it an opportunity lost.

More subjectively, the world of SD3 is basically a dystopia. The guys are misogynistic jerks and the girls are sluts (with a couple of exceptions). Love is the exception rather than the rule, absentee parenting is the norm, and there is no place for empathy or altruism. The PC has the freedom to rape an unconscious girl, but not to stop someone else from raping her. The character of Alison shows that this kind of world has a human cost, but that didn't make it any more palatable to me.

Final thoughts

Objectively, this is the greatest AIF ever made. There are other games that are better in individual areas, but overall SD3 is the king of the mountain. Goblinboy deserves to be applauded for having completed such a massive undertaking.

For me it lacked some of the sense of fun of comparable games, and I would also have preferred stronger and more likeable characters. Despite those gripes, I have to admit that I have spent an enormous amount of time playing SD3, trying to discover every secret. It's a game that sucks you in and doesn't let go.

Is it the greatest AIF that will ever be made? I don't think so. After all, we still have GoP3 to look forward to ;)

Rating: A

The bugs and spelling mistakes, as well as all my other subjective quibbles, push SD3 very close to an A-. However, given that there is likely to be a bugfix release before too long, I think an A is appropriate. ◆

This month we're talking about names. You all have played enough of these games to know that sometimes the PC already has a name, and sometimes you are allowed to enter your own, but how exactly do you do that? Well, as it turns out, it's pretty easy to do no matter what language you choose, but I didn't want to torture our staff members every month (I leave that for special occasions). I did throw a couple of extras into the assignment to keep it interesting, see what you think.

This should be an easy one. Just show how to set it up so that the player can type in a name for the PC at the beginning of the game. In fact, that's a bit too easy isn't it? Also allow the player to name one of the NPCs in the game. Still too easy? Ok, how about letting the PC change his name mid-game?

TADS 2 Segment by A. Bomire

This month's "Coder's Corner" is a little less complicated than some of the previous ones. In fact, we are going to be making use of a lot of the stuff we've already covered – just in a new way. This month, we will be exploring names – the player's and also that of an NPC. The player will be allowed to input his (or her) own name, and even name an NPC. To add a little spice to it, we'll also allow the player to change either of those names at any point in the game.

Neither of those tasks is very complicated, and actually involves techniques we've discussed before. The main thing we will be interested in doing is getting the player's input, which was covered in "Coder's Corner #3" when we created the watch with an alarm that can be set. The new twist that we are adding this month is changing the nouns and adjectives of objects mid-game (as opposed to at the design stage). Again, it isn't very complicated, but there are some things to keep track of and remember, which I will go over as they arrive.

Enough discussion, let's get coding! To set the stage, let's assume for this game that the player is a mad scientist working hard to create a fem-bot in his laboratory. For what purposes? Well, this ***is*** AIF! Once the fem-bot is active, the player can give "her" a name. Let's create the lab first:

```
startroom: room
    sdesc = "Laboratory"
    ldesc = "This is your secret laboratory, where you do all of your work.
        Today, you are working on your 'FEMBOT' project. "
;
```

In an actual game, I would of course create work benches, equipment, and other objects appropriate to a mad scientists' laboratory, but a simple room is all we need here.

Okay, that sets the stage – let's actually do something. To start the game, we will want to get the player's name. Well, let's pause here a minute. Do we actually want the player's name? After all, our mad scientist is going to be either male or female, and our player is not necessarily going to be the same sex. What I like to do is let the player know what sex the player character is going to be and allow him or her to choose an appropriate name. In our game, the player character is going to be male. I usually set up a small function to get the player's name, and store it in an appropriate property of the player character object. Let's call our player's name....*name*! So, here is a quick function to acquire the player character's name, using TADS *input* function:

```
modify Me:
     name = 'Irving'
;

GetName: function
{
    local x := '';

    "In this game you will be playing the part of a male scientist.
     \nPlease enter the name you wish to use: ";

    x := input();

    if (x = '')
       "\bNo input detected, so the default name of <<Me.name>> will be used. ";
    else
    {
       Me.name := x;
       "\bWelcome, <<Me.name>>! ";
    }

    "\bPress any key to continue..... ";
    x := inputkey();
}
```

And there you have it – a simple routine to get the player's input, assign it to a *name* property, and even redisplay it to the player using TADS embedded expressions. In fact, anytime during the game that you wish to display the player's name, you simply need to include this code: <<Me.name>>. The *inputkey* function that you see at the end waits for the player to press a key before continuing. (For more information on using *input* and *inputkey*, and on embedded expressions, see the TADS manual, Chapter 8 – Language Reference).

I usually place a routine such as the *GetName* function defined above someplace early in the game, usually in the initialization area. A great place to put it is in *commonInit*:

```
replace commonInit: function
{
      GetName();
}
```

But, you can of course place it anywhere you feel is appropriate. Now, I'm not going to bore you by going over the exact same thing for naming our Fem-bot – it is done in the exact same way! However, there are a few additional things that need to be done

when naming an object other than the player character. The player will want to refer to the object by name. Usually, this is done by setting the *noun* and *adjective* properties when creating the object. However, in our case we want these properties to be dynamic. Well, TAD won't let you just change them like we changed the *name* property above. Instead, we have to use two TADS functions: *addword* and *delword*. Much as their names imply, *addword* allows us to add a word to the *noun* or *adjective* property of an object, and *delword* allows us to delete a word from the same properties. For both of them, the format is basically the same:

addword(*object*, &*property*, *word*);

Where *object* is the object whose *noun* or *adjective* list you wish to alter, &*property* is &noun to alter the *noun* property and &adjective to alter the *adjective* property, and *word* is the word you wish to add (or delete in the case of *delword*). (For more information about either of these functions, see the TADS manual, Chapter 8 – Language Reference.)

To see how this works, let's create our Fem-bot. She will be an Actor object who is currently "inactive". She is completed, but just awaiting our player to press the "start" button and get her up and online. Once this is done, the player will be prompted to give her a name. Because she is a female robot, she will also have female body parts. I'm not going to go extensively into this (for more information, see this newsletter's series on "Programming Erin"). I'll just create a simple pair of breasts for her to use as an example.

```
Fembot: Actor
     location = startroom
     sdesc = "<<self.name>>"
     ldesc =  {
         "This is the fembot you have been working upon. ";
         if (not self.isActive)
            "She is lying here on the workbench, waiting for you to activate her
            by pushing the start button. ";
     }
     noun = 'fembot'
     name = 'fembot'
     isActive = nil
;

FemBreasts: fixeditem
     location = Fembot
     sdesc = "<<Fembot.name>>'s breasts"
     ldesc = "Your fembot has a lovely set of breasts. Well, they would
        be, wouldn't they? You designed them after all! "
     noun = 'breasts' 'breast' 'tit' 'tits'
     adjective = 'fembot\'s'
;

startButton: fixeditem
     location = Fembot
     sdesc = "start button"
     ldesc = "This button is designed to activate and deactivate your fembot. "
     noun = 'button'
     adjective = 'start'
     verDoPush(actor) = {
          if (Fembot.isActive)
              "After all this work to get her online, you aren't
               willing to shut her off again. ";
     }
     doPush(actor) =
     {
          local x := '';

          "You reach towards the button which will activate your fembot, your
```

```
finger shaking with excitement. After all these months, you work is finally
complete. With a quick stab, you depress the button, standing back and watching
in anticipation as your 'bot hums to life. Her eyes open, shining with an inner
light. She sits up, and looks at you.

         \b\"Running internal tests....all components functioning within normal
parameters,\" she says. She closes her eyes, and opens them again. \"What is my
designation?\"

         \bWonderful! Fantastic! You are almost dancing with joy! Then you
realize she is waiting for your answer. Hmm..you never really thought of a name.
What do you wish to call your creation? ";

         x := input();
         if (x = '')
         {
            "You just say the first thing that comes to you: Dot. ";
            x := 'Dot';
         }

         Fembot.name := x;

        "\bYour robotic companion nods. \"Yes, Master,\" she intones, as she
        rises from the workbench to stand before you.
        \"I am '<<Fembot.name>>'.\" ";

        addword(Fembot, &noun, x);
        addword(FemBreasts, &adjective, x + '\'s');
        delword(FemBreasts, &adjective, 'fembot\'s');
        Fembot.isActive := true;
   }
;
```

There is a lot going on there, and I'm not going to go over all of it. One of the important things to notice is that I set the *sdesc* property of our Fembot and her breasts to change dynamically as we rename our Fembot. If I had created other body parts and clothing, I would do the same for them. Also, you'll notice that the adjective I added for our Fembot's breasts is the possessive form of her name (with a 's on the end of it). This may not always be accurate, for example a name like "Chris" uses a possessive form of "Chris'" not "Chris's". However, it will be fine most of the time. If you wish, you can check the final character of the entered name and alter the possessive form based upon whether the name ends in "s" or not. This is a little beyond the scope of this article.

The last thing we want to add is the ability to change the name of the player or the Fembot. This is simply a re-application of the techniques we've already reviewed, so it isn't anything major. To do this, I will create a new verb, *renameVerb*, which the player can use to rename either the player character or the Fembot:

```
renameVerb: deepverb
     sdesc = "rename"
     verb = 'rename'
     doAction = 'Rename'
;
```

Simple enough. Now, let's add some code to rename the player:

```
modify Me
     name = 'Irving'
     verDoRename(actor) = {}
     doRename(actor) =
```

```
        {
            local x := '';

            "\bOkay, what name do you wish to be known by now? ";

            x := input();
            if (x = '')
              "No input detected. Your name will remain the same. ";
            else
            {
                self.name := x;
                 if (Fembot.isActive)
                    "\bYou get <<Fembot.name>>'s attention. \"From now on,\" you
                    say. \"You will refer to me as <<Me.name>>.\" ";
                else
                    "\nOkay, from now on you will be known as <<Me.name>>. ";
            }
        }
    ;
```

This is really just a variation of our already created *GetName* function. In fact, you could use the exact same function here if you wanted. I simply added a bit to have the player tell his robotic creation what his new name will be.

Changing the Fembot's name is much the same, with the small addition that we need to alter the appropriate *noun* and *adjective* properties using *addword* and *delword*:

```
    Fembot: Actor
    ...previously defined code is cut for clarity
        verDoRename(actor) =
        {
            if (not self.isActive)
               "Your fembot is not yet activated. ";
        }
        doRename(actor) =
        {
            local x := '';

            "Okay, what name do you wish to call your fembot? ";

           x := input();
           if (x = '')
              "No input detected. Your fembot will remain <<Fembot.name>>. ";
           else
           {
               delword(Fembot, &noun, Fembot.name);
               delword(Fembot, &adjective, Fembot.name + '\'s');
               Fembot.name := x;
               addword(Fembot, &noun, Fembot.name);
               addword(Fembot, &adjective, Fembot.name + '\'s');
               "You face your creation. \"From now on your name is
               <<Fembot.name>>,\" you tell her. She nods in assent. ";
            }
        }
    ;
```

That about wraps it up. There is one last thing to remember when using *addword* and *delword*: they are a bit unpredictable, especially when performing UNDO and RESTORE. What I mean is that sometimes your changes to the noun and adjective properties aren't 'UNDO'ne, or remembered when you do a restore. This is a rare occurrence, but it can occur.

TADS 3 Segment by Knight Errant

This month's task is a little different than others, we'll be allowing the player to specify the name of his character. It's a bit different because TADS 3 normally loads the intro text and immediately dumps the player into the starting room. Here, we'll have to interrupt that normal process to get data from the prompt before loading the intro text. We'll still utilize the showIntro() method of the gameMain object, like so:

```
gameMain: gameMainDef
  initialPlayerChar = me
  showIntro()
    {
      local inputName; // "name" is a predefined property, so we use
inputName to avoid confusion.
      Do
      {
        "<.inputline>"; // inputline is a method that reads a line of text
from the prompt and stores it as a string.
        "What would you like your character to be named?";
       "<./inputline>";
       inputName = inputManager.getInputLine(nil,nil);
      }
      While ( inputName != nil ); // This is just a conditional check
                          // to verify that the name is acceptable.
                          // Replace as appropriate
       me.pcName = inputName;
    // Put the rest of your intro here.
    }
```

We use a Do-While loop so that we can run a check on the name before accepting it. T3 will first run the Do section, then perform the check in the While section. If While returns true, it'll repeat the Do section and check While again. After While returns false, then we make a property of our "me" object called pcName and set it to inputName. This means that when our PC introduces himself, we can use the me.pcName property to repeat what the player input, like so: "Hi, my name is <<me.pcName>>, what's yours?" NPCs can refer to the pcName in a similar fashion. If you like, you can even use the same method to allow player customization of whatever else you like, by taking player input and assigning it to a property of whatever object the property applies to. Feel free to try out some variations on your own. Incidentally, if you'd like to put some restrictions on what the player can input, in the While list you can compare the input to a list, or you can run it against a Regular Expression, which is a pattern-matching system common to many programming languages. Read more about that here:

http://www.tads.org/t3doc/doc/sysman/regex.htm

Good luck!

Inform 6 Segment by 'trix

Hello, minions and minionettes.

This month we're doing more letter-by-letter parsing (hooray for us), so here's some preliminary stuff about characters and arrays.

Inform has (essentially) two types of arrays: byte arrays and word arrays. A word, in this context, just means a block of bytes, not a sequence of letters. Byte arrays are accessed by the operator `->` and contain numbers between `0` and `255`. Word arrays are accessed by the operator `-->` and hold values from `-32768` to `+32767` (on the Z-machine). If you're compiling to Glulx then words are 4 bytes (as opposed to 2), and hold a much bigger range of values.

Inform uses 1 byte characters, so letter-by-letter parsing generally involves using byte arrays. Here's some functions for dealing with byte arrays.

```
! Checks if two byte arrays hold the same values
!  (case insensitive)
[ MatchCharArray arr0 arr1 len   i;
   for (i = 0 : i < len : ++i)
     if (LowerCase(arr0->i) ~= LowerCase(arr1->i))
       rfalse;
   rtrue;
];

! Prints the characters of a byte array
[ PrintCharArray arr len   i;
   for (i = 0 : i < len : ++i)
     print (char) arr->i;
];

! Copies the contents of one byte array into another
[ CopyByteArray src dest len   i;
   for (i = 0 : i < len : ++i)
     dest->i = src->i;
];
```

Functions like these do the work required for this month's task: letting the player name something, being able to print that name back to the player, and being able to match that name next time the player types it.

To start with we'll get the player to enter her name.

```
Constant STORY = "Names";
Constant HEADLINE = "^Putting stupid things for names may
  seem funny but it's just annoying after the first ten seconds.^";

Include "Parser";
Include "VerbLib";

Constant MAX_NAME_LEN = 16;

Array player_name -> MAX_NAME_LEN;
Global player_name_length = 0;

Object cave "Cave"
  with description "A big boring cave of emptiness."
  has  light;

[ Initialise  l;
   do
   {
      print "^Enter your character's name:^>>";
      KeyboardPrimitive(buffer, parse);
      l = WordLength(1);
   }
   until (l > 0); ! Yes that's right: I use repeat-until loops. So what?
   if (l > MAX_NAME) l = MAX_NAME;
   CopyByteArray(WordAddress(1), player_name, l);
   player_name_length = l;
   print "Your chosen name is: ";
   PrintCharArray(player_name, player_name_length);
   new_line;
```

```
    location = cave;
];

! Prints the characters of a byte array
[ PrintCharArray arr len   i;
   for (i = 0 : i < len : ++i)
     print (char) arr->i;
];

! Copies the contents of one byte array into another
[ CopyByteArray src dest len   i;
   for (i = 0 : i < len : ++i)
     dest->i = src->i;
];

Include "Grammar";
```

Notice that we have an upper limit on the length of the name (specified by the constant `MAX_NAME`), so a longer name will be truncated.

Unfortunately, the Z-machine is case insensitive: the player's input is always in lower-case by the time we get it. Glulx preserves the case, so we can use whatever capitalisation for the name that the player used when she typed it in. But in Z-code, it's probably best just to guess at a suitable capitalisation. This version changes the first character in the name to a capital (but only if we're compiling to Z-code):

```
[ Initialise  l;
   do
   {
      print "^Enter your character's name:^>>";
      KeyboardPrimitive(buffer, parse);
      l = WordLength(1);
   }
   until (l > 0);
   if (l > MAX_NAME) l = MAX_NAME;
   CopyByteArray(WordAddress(1), player_name, l);
   player_name_length = l;
#ifdef TARGET_ZCODE;
   player_name->0 = UpperCase(player_name->0);
#endif;
   print "Your chosen name is: ";
   PrintCharArray(player_name, player_name_length);
   new_line;
   location = cave;
];
```

Now the next thing we're going to need is to be able to spot when the player has typed the given name in her command. For this, I'm going to assume you're a Programming Erin devotee, and you have all the arcane parsing stuff I posted all those many whatever ago. If not, you can get GParse.h from the Yahoo aifarchive, and include it like this:

```
Constant STORY = "Names";
Constant HEADLINE = "^Putting stupid things for names may
  seem funny but it's just annoying after the first ten seconds.^";
Replace Identical;
Replace PrefaceByArticle;
Include "Parser";
Include "GParse";
Include "VerbLib";
```

Now a player-character object which parses the given name:

```
Object playchar "yourself" cave
  with words
       [ w l arr;
          l = WordLength(wn-1);
          arr = WordAddress(wn-1);
          if (l == player_name_length
              && MatchCharArray(arr,player_name,l))
            return name;
       ],
       number 0,
 has  proper animate concealed;

! Checks if two byte arrays hold the same values
!  (case insensitive)
[ MatchCharArray arr0 arr1 len   i;
   for (i = 0 : i < len : ++i)
     if (LowerCase(arr0->i) ~= LowerCase(arr1->i))
       rfalse;
   rtrue;
];
```

Since this is AIF, we generally need to parse genitives as well as nouns. For our purposes, a genitive is the given name with an apostrophe afterwards (optionally followed by an s). So it should be this:

```
  with words
       [ w l arr;
          l = WordLength(wn-1);
          arr = WordAddress(wn-1);
          if (l == player_name_length
              && MatchCharArray(arr,player_name,l))
            return name;
          if (l - player_name_length == 1 or 2 &&
              arr->player_name_length=='')
          {
             if ((l - player_name_length == 1
                  || arr->(player_name_length+1)=='s')
                 && MatchCharArray(arr,player_name,player_name_length))
               return genitive;
          }
       ],
```

That's pretty much all there is to it. But for a bit more depth, let's see how we'd go about naming an NPC. This time, instead of doing it from an "Enter name" prompt, we'll do it from an ordinary player command. We're going to name an Elf.

```
Object elf "Elf" cave
  has  animate;
```

Now this Elf needs somewhere to hold its name (when we give it one). This one's going to use a property array, so we can have a look at property arrays.

```
Object elf "Elf" cave
 with  namedata 0 0 0 0 0 0 0 0,
       namelength 0,
  has  animate;
```

`namedata` is the property array. Each of those zeroes is a word in the property array, so the `namedata` array as declared is 8 words long (16 bytes in Z-code; 32 bytes in Glulx).
The array can be accessed by `elf.&namedata-->i` (as words) or as `elf.&namedata->i` (as bytes). The length of the array is given by `elf.#namedata`, which is the number of bytes in the array.

```
Object elf "Elf" cave
 with  namedata 0 0 0 0 0 0 0 0,
       namelength 0,
       words
       [w l arr;
         if (w == 'elf') return name;
         if (w == 'elf^s') return genitive;
         l = WordLength(wn-1);
         arr = WordAddress(wn-1);
         if (self.namelength > 0
             && l == self.namelength
             && MatchCharArray(arr,self.&namedata,l))
           return name;
          if (self.namelength > 0
             && l - self.namelength == 1 or 2 &&
              arr->(self.namelength)=='')
          {
             if ((l - self.namelength == 1
                  || arr->(self.namelength+1)=='s')
                 && MatchCharArray(arr,self.&namedata,self.namelength))
               return genitive;
          }
       ],
       short_name
       [;
         if (self.namelength==0) rfalse;
         PrintCharArray(self.&namedata, self.namelength);
         rtrue;
       ],
  has  animate;
```

Now we let the player name the elf in the easiest possible way: e.g. "Name the elf Idiothole".
Stick this new action at the end of your code.

```
[ BaptiseSub  arr len;
   if (~~noun provides namedata)
      "You can't name ",(the) noun,".";
   arr = WordAddress(consult_from);
   len = WordLength(consult_from);
   if (len > noun.#namedata) len = noun.#namedata;
   CopyByteArray(arr, noun.&namedata, len);
   noun.namelength = len;
#ifdef TARGET_ZCODE;
   noun.&namedata->0 = UpperCase(noun.&namedata->0);
#endif;
   give noun proper;
   "You name ",(itorthem) noun," ",(name) noun,".";
];

Verb 'name' 'baptise'
   * noun topic              -> Baptise;
```

The action will work for any object with `namedata` and `namelength` properties.

Inform 7 Segment by Dudeman

Another month gone by, another coder's corner for you all to enjoy. This month, we are dealing with the topic of naming players in our games. Many AIF/IF games will present you with a prompt in the beginning of the game asking you to name the Player Character (PC). This is presumably to enhance role-playing and making it easier to insert yourself or whoever you want into the role of the PC. In fact, this is so common that the Inform 7 "recipe book" in the documentation comes out of the box with an example on how to do this using Inform 7 code. The example "Identity Theft" in Chapter 5.2 "Traits Determined By the Player" shows a way to do this. This is a perfectly fine way to do it and I only changed a few minor things that are mostly cosmetic. The only major problem that I find with it is that you are prompt for the name after the first room's description is printed which I don't really like so I added the rule "Instead of looking for the first time: do nothing." Which keeps the first room description from being printed.

```
player's name is an indexed text that varies.

After printing the banner text:
      change the command prompt to "Please enter your name: ".

Instead of looking for the first time: do nothing.

To decide whether collecting names:
      if the command prompt is "Please enter your name: ", yes;
      no.

After reading a command when collecting names:
      if the number of words in the player's command is greater than 5 begin;
            say "[paragraph break]Who are you, a member of the British royal
family? No one has that many names. Let's try this again.";
            reject the player's command; end if;
      change player's name to the player's command in title case;
      change the command prompt to ">";
      say "[line break][intro text]";
      move the player to your office;
      reject the player's command.

Rule for constructing the status line when collecting names: do nothing.
```

Now some of this code is new stuff that hasn't been covered in previous issues so let me go into a little depth on some of them and define a few things that authors might find useful to use in the future.

1. An indexed text is a text variable that holds a text value and can be changed during play. This is just what we want for the name of the player since it can store a name and print it back out when we need to refer to the player (can learn more in Chapter 19.1 of the documentation).
2. The banner text is the text that is printed at the beginning of the game that displays the title of the game along with a little information about the game. In this case we want the player to be prompted after this text is printed (can learn more in Chapter 17.35 of the documentation).
3. An "after reading a command" rule is a rule which is checked after the player enters a command and hits "enter", but before that text is processed by Inform to perform an action. Normally if the player just typed a random word like "James", Inform would try to process that into a command, not find anything and return something like "That's not a verb I recognize.". Using this rule, we can avoid that and grab that text to be used in our indexed text value (can learn more in Chapter 17.31 of the documentation).

There we go, the player will now be prompted at the beginning of the game before the intro text is printed to give the PC a name of their choosing. The name is then stored in a text token named "player's name" which can be used at any time during the game to print the PC's name as it was typed by the player.

Now let's proceed to set up a small game world and the basic plot of our game:

```
To say intro text: say "You are [player's name], or perhaps you should say Dr.
[player's name] since you just recently finished Medical school and got a nice
job at your uncles private practice. You're a little nervous since it is your
first day, but you don't expect anything too out of the ordinary so you're sure
you can handle it.".

The description of the player is "Your name is [player's name], a young doctor
fresh out of med school.".

Your office is a room. "This is your new office with just about everything you
need in it. To the east is the waiting room.".

The waiting room is east of your office. "The waiting room is the place where
patients can wait for a doctor to see them. Right now the room has no patents
waiting since you haven't officially opened for the day yet."

The secretary is a woman in the waiting room.
The initial appearance of the secretary is "Set up in a large desk at the front
of the waiting room is your office's secretary."
```

Now we have a very basic game world full with one NPC, the office secretary. Now, while descriptive of her position, "the secretary" is not a very good name for a character is it? Why don't we give the player the ability to name the secretary whatever they like. This can be done in a very similar way that we allowed the player to name the PC.

```
The secretary has an indexed text that varies called the name.
The name of the secretary is usually "the secretary".

Understand the name property as describing the secretary.

Before examining the secretary for the first time:
say "You've met your uncles secretary once before. You remember her introducing
herself, but can't quite remember what her name was...";
change the command prompt to "What was her name again? " instead.

To decide whether collecting the secretary's name:
      if the command prompt is "What was her name again? ", yes;
      no.

After reading a command when collecting the secretary's name:
      if the number of words in the player's command is greater than 5 begin;
            say "[paragraph break]Try a shorter name.";
            reject the player's command; end if;
      change the name of the secretary to the player's command in title case;
      change the command prompt to ">";
      say "[line break]Ah yes, [the name of the secretary] was her
name![paragraph break]Anyways, [run paragraph on]";
      try examining the secretary;
      reject the player's command.

The description of the secretary is "[the name of the secretary] is a very young
woman, you would guess she can't be much more then 20. She is also unbelievable
hot. With her huge breasts, apple shaped ass, and tight curves, she looks more
like the kind of girl you would see the lead role in a high quality porno then
as a secretary in a small doctor's office."
```

Just like before, the player is prompted to type in a name for the NPC (in this case when the secretary is examined for the first time) and that text is then stored in a text token. The only major difference is that this time we have made the indexed text a part of the

secretary. This is so that Inform 7 will be able to recognize the secretary's name when used in a command since it is a property of the secretary and not an abstract text token not attached to any in-game object.

One final thing we might want to do with the PC's name is give the player the ability to change it during the game. We can do this by creating an action just like we have done before, but instead of that action applying to a thing, we want it to apply to one "topic" which means any text not recognized as an item by the game.

```
name changing is an action applying to one topic.
Understand "change name to [text]" as name changing.

Carry out name changing:
now player's name is the topic understood in title case;
say "Your name is now [player's name].".
```

That's all there is too it. This simple action is all you need to give the player the ability to change the PC's name on the fly by changing the indexed text "player's name" to the "topic understood" which is the text that the player has typed in the text portion of the last command.

And with that we are finished. Well not finished with the game of course, but we have gotten to a nice start which shows off a way to name players in your AIF using Inform 7. It's not that difficult to do and can give a game a nice customized feel that some gamers like. Anyways, whether you choose to go this route in your games or not, I hope you learned something about how Inform 7 works from this example and have found it to be a helpful read.

ADRIFT Segment by BBBen

This month's challenge is a particularly funny one to deal with in ADRIFT. The first part is easy: in order to have the player be able to enter the name of the PC, just go into the "Adventure" menu, select "Player", and tick the little box that says "Prompt for name." In order to bring up that text at any time in-game, all you have to do is type:

%player%

And you can put it in the middle of text, like:

"Hey, %player%!" said Erin, "It's good to see you!"

Nice and simple. Now, the tricky part comes with the remaining dimensions of the task – they cannot be done.

To break it down more specifically, the first thing that can't be done is renaming a PC. You can indeed have the player define a piece of text as a variable (in ADRIFT 4.0, not in 3.9) so you could have the player define a girl's hair colour or something, but you couldn't have them define the girl's name, as the parser would not be able to understand any commands given that refer to her. It's been tried, and to my knowledge it can't be made to work, but I'm not a definitive source on this, and I would be happy to be proven wrong by someone who's spent more time fiddling around with the inner workings of 4.0 than I have.

On to the second part: again, it isn't possible, for the same reasons. The closest you can really come is a kind of illusionary work-around that I did in *Crossworlds Part 0*. In that game the PC temporarily becomes different characters, and to accomplish that I had to override a bunch of the default engine responses and do some other fancy stuff that I can't remember. The point I'm getting at, however, is that it's still kind of a cheat. You just can't change the name of a character after that initial "Prompt for name" opportunity, and then only for the PC. At least, that's as far as the parser is concerned. I suppose if you were just constructing some kind of a short story in which you make a few limited choices along the way then you actually could change the names of characters, as they wouldn't be interactive "characters" as far as the engine is concerned.

Anyway, that's about all I have to say about that one. Sorry if this disappoints anyone who was keen to have custom names in an ADRIFT game… hey, maybe it'll be in version 5? We can only hope, right? ◆

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.