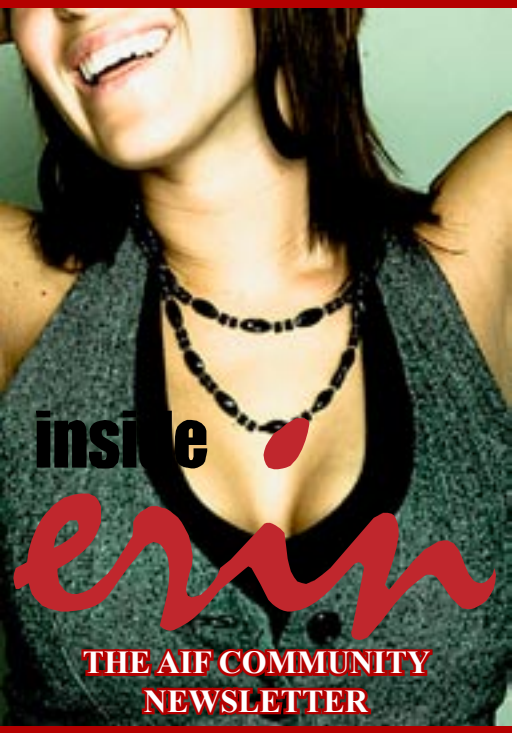

Contents

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.

It seems like it's been awfully quiet this month. So much so that at one point the AIF Archive went for a whole week without a single post until someone commented on it and things started back up a bit.

Personally, things have been a bit quieter too, which gave me the chance to actually play a few games for a change. I went back and started playing though the games from the 2004 Mini-comp. Of course, then the newsletter editor thing in me said, "Since you're playing them again, you should do some reviews." (damn voices). So I wrote up a couple of reviews. If things stay slow, I might try to do some more for next month, we'll have to wait and see.

Speaking of Mini-comps, you have about a month left before the June 8th due date for the current comp, still plenty of time to put something together, even if you haven't started yet. Remember that the point of the comp is to produce fairly small, simple games, not really to write the most complicated game possible within the bounds of the rules. I know that it seems to have turned into that sometimes, but this has always, primarily, been a format to allow new authors to get their feet wet, without feeling like they have to write a novel sized game. Keep that in mind.

As always, we try to keep you updated on what's going on in, not only our neck of the woods, but over at TFGames and The Collective. Sounds like some interesting stuff actually, and I encourage you to check it out if you have the time. For those of you following along with Aphrodite's story, we have the second letter this month, and of course, Coder's Corner brings you another look into the world of cross-platform coding (I just invented that term. Like it?). Oops, strike that. I just googled it and got like 3000 hits. Oh well, better luck next time. ◆

This was a mostly quiet month, with the main topic of conversation being the beta test of GoblinBoy's next game. He briefly (accidentally) released the link to the beta to the whole group, but managed to retract it before long. It sounds like the game has been a little delayed by the addition of more content, so it's hard to say when it will be out.

AIFGames.com finally came back online this month, though things are still a bit quiet over there, either because it'll take a little while for people to get back into using it, or just because there really wasn't all that much going on around the place.

And since it was so quiet, that's about all I have to say, other than to remind you that the 2009 AIF mini-comp deadline is June 8, and I'm betting hardly anyone's gotten started on their entries yet, so get to work over May and remember to leave some time for beta testing! ◆

As soon as the first piece of ancient cave art was done one cave man (or woman) must have turned to another and asked "But is it art?"

Now AIF of all color goes through this shift from a desire for anything related to the subject to the desire to find the perfect examples of the art form. As a game creator myself I'm not in a good position to make any statement or judgment on the best of the form, though I'm humble enough to know mine aren't the finest examples.

No, I thought I'd point out a peculiar feature of the internet and maybe point out something that all game creators should be aware of, and again it's something that I've already committed the sin off.

It's relates to Sturgeons Law, that 90% of everything is crap. With most other mediums things disappear off the cultural radar, honing them down so we remember only the very best or the very worst. Thanks to the internet we don't have that. Thanks to the power of modern technology we can play every single AIF game since before the internet began. But whilst this allows us to enjoy blasts from the past, it also means every single bad game is there in all in inelegant glory.

In other words, it probably not a good idea to post any game, or fragment thereof, up to the internet, because people are going to be looking at it for years to come.

In the actual board itself things have been relatively quiet. In the March contest Kimberley Rex won the popular vote, and the organizer kindly decided to give their choice award to my good self. I'd like to thank both Jaimehlers and Maelyn for their time effort and money on this contest. There been a little discussion on the future of contests so we'll see what happens in the future.

On the game front TinaB continues to push out the latest Beta version of RAGS with *House of the Future*, and Fipse provides and update on *Magic Ring*, whilst on the RPGMaker front Alphamage gave us the *Warlock of Druwald*. On the Hypnopic crossover front Lydia (or SImgirl) has kindly decided to share her non-transformation game of *University Sim* (RAGS) and stop by for a chat. ◆

April was a busy month for the Collective Gamers...

Lydia updated University Sim 2

Game: http://rapidshare.com/files/225788281/dating_sim_demo_posted_005.rag

Discussion Thread: http://hypnopics-collective.net/viewtopic.php?f=11&t=14221&st=0&sk=t&sd=a

She notes that she has another good sized update for everyone.

"It's still a demo as I focused what game time I had on adding content. Thus this may be somewhat buggy. Please excuse as my little game is still under construction. Until I've gotten Charlie's path completed, everything is still so subject to change that alot of testing would not be very productive. If I waited until I had a more finished product, it would be too late to implement suggestions, so hopefully everyone finds this a suitable compromise. So please, no bug reporting. I'm sure they are there and I will address them at the allocated time. Please do provide me with your overall feedback on the game. Hope you enjoy."

The Sane Scientist released The Interface

Game: http://hypnopics-collective.net/cpg132/displayimage.php?pos=-90015

Discussion Thread: http://hypnopics-collective.net/viewtopic.php?f=11&t=15430&start=0&st=0&sk=t&sd=a

It is described as, currently a simple game that he really wants to expand, but he would much prefer some community feedback in the process. Please provide your thoughts and feedback.

Dr753 released Special Delivery

Game: http://rapidshare.com/files/219089174/Special_Delivery.rag.bak

Discussion Thread: http://hypnopics-collective.net/viewtopic.php?f=11&t=15258

He says:

"I've gotten into a groove when working on games with RAGS, Step 1 - Concept and working on game. Step 2 - Get off track when working on game and don't ever come back to it. Fun, huh? Anyway I've found a way around that. Game start 11:28 AM EST 4-7-9, Game Finish 5:19 PM EST 4-8-9. That will also answer any questions about the quality of the game. I hope you enjoy the game, if you don't, well it's not very long."

Lydia also started a discussion thread on the lack of comments on games when they are posted.

http://hypnopics-collective.net/viewtopic.php?f=11&t=15462

Some things to ponder in her thoughts and those that answered her thoughts there...

And that's this month at the Collective! ◆

Dear Mortal Men and Women,

Last month I brought you the first letter from Eric, part of a couple for whom I'd arranged their meeting in exchange for letters from each of them describing their first sexual experience together. Today, I present the letter from Sonia. Her presentation is more formal and considered than Eric's which I expected - knowing how focused and intense she is. I hope you all enjoy it.

***

Dear Veronica,

I have to thank you for bringing Eric to my attention. Left to my own devices, I might have missed him completely. That probably comes off cold. I need to clarify. When I'm on the train, I tend to draw in on myself, unaware of anyone or anything outside my own thoughts. I'm rarely where my body is, in the present. I'm either puzzling through things I need to take care of, or I'm savoring some dirty, little memory or fantasy.

I can't imagine what I look like when I'm withdrawn. Probably pretty aloof, unreachable. There's comfort in that, too. It's like having a secret. I'm sitting there worrying some delicious memory like a piece of hard candy, and no one around me would guess it. In fact, they might assume I'm incapable of any kind of passion. There's some gorgeous irony.

I'm a creature of extremes, I suppose. I'm either wandering in thought, completely disconnected from the now, or I'm one hundred percent in the moment, acutely aware of every sensation. There's no middle ground for me. Switching me on is the challenge. If you hadn't pushed, I might've missed out on Eric.

I made a lot of interesting observations throughout our evening, but I won't waste your time with them now. I know what you're after. I know what you want to hear about.

The more we talked, the more Eric intrigued me. I could tell that we were going to be good together and that sex with him was going to be great. At that point, I put thought aside and let need take over. I think that frightened him a little. It takes a lot to get my attention, but once you have it...

Eric has a fine body (although I doubt he'd admit it), but I have to confess I spent most of my time enjoying his eyes. They gave away so much, even more than all of his whispers and moans combined. Powerful eyes. I watched them roam my body, trying to wrestle off my clothes, and it was the most arousing sight. I like my body, but I don't think it's anything special. But the way he was looking it over... It was very empowering.

I really played into it, working the tease and baring my body a bit at a time. The more appreciative his look, the more aroused I became. I could feel his stare exploring me. It was all I could do to stay focused. I kept fighting the urge to masturbate there in front of him while he watched.

At one point, I said something silly like, "I can tell you like what you see." It was a little embarrassing and jarring. I wish I'd stayed quiet. I covered that up by bending down to kiss him. He has a fantastic mouth. And instead of trying too hard and attacking me with clumsy kisses and ramming tongue, he let me lead. It was a warm, penetrating kiss that left my legs trembling. I'd suspected we'd get along famously, but that kiss...

Eric was so into it that he didn't notice my hands fumbling with his pants until he was completely exposed as well. He seemed so overwhelmed, as though he was at my mercy. I have to admit I loved that. I became addicted to his reactions, to his shaky breaths and disbelieving looks. It's the sexiest thing in the world when someone looks at you with wonder.

While he sat on the edge of the bed, I entwined myself around him until his breath steamed over my breasts and his cock tickled my bush. He kissed the center of my chest, his five o'clock shadow rasping against the insides of my breasts. He absolutely radiated need: deep, desperate, infectious need. I was caught between the intense, physical sensations and his every reaction.

An impatient pang rang out between my legs, and my hips began to rock of their own will, rubbing my pussy against his cock. It was only then that I noticed how wet I was. I'd been so distracted, so caught up in observing. A delicious, simmering heat spread across the insides of my thighs. Need was boiling to the surface.

I lifted his face from my breasts. His eyes were wide with anticipation. As I cocked my head to kiss him, my hair spilled over our faces, making a private meeting place for our mouths. Another incredible kiss. I fished between us for his cock, drawing out a few more of those disbelieving moans I love. He was hot and hard in my hand--not too big, just right. I ran my thumb over his head and found the eager drop of pre-come I'd hoped to find. He was so ready. I knew I couldn't draw this out very long.

In an effort to catch up, I held his cock deep in my bush and writhed against him, my clit sliding up and down his shaft, throwing sparks that made me wince. I suddenly felt very empty, very much in need of filling. I locked eyes with him as I rose up and sank back into his lap, guiding his cock inside me. I was so wet that I just seemed to absorb him. It felt fantastic, but it was his awed expression that really made the moment memorable. His look was so intense that I could almost hear "oh thank you thank you thank you."

To show my appreciation, I focused, remaining completely still, and squeezed my pussy around his cock. Each little pinch made him blink and grunt, as though someone kept poking him in the side. I felt wonderfully greedy, every squeeze punctuated with the thought "mine." Need was getting the best of him. He tried to shift, tried to push deeper, but I held him still and kept sucking on him with my pussy.

I pushed my hair out of my face and smiled to myself. I couldn't get over how right it all felt. I knew that a few good bounces at that point would probably bring us both off, but I didn't want it to end that quickly. I disentangled myself from him--probably a little too brusquely. (To my complete surprise, that last little pinch had nearly made me come.)

I sat next to him on the bed and started up some slow, deep kissing while I regrouped from the sudden rush to the brink. As we kissed, I held his face and let his hands wander my body. I felt cherished, and his touch made me ache with need again. I brushed my lips against his ear and asked him if he knew how to give head. He made an anxious sound that I took as a yes. I led him over to a chair and sat on the edge of its cushion.

Eric scrambled out of the ruin of his clothes and knelt down in front of me. Again, his eyes were enrapt, and I felt thoroughly worshipped as he looked me up and down. That was intensely arousing. As he leaned in to get a taste, I was boiling with anticipation. I bit back a sigh as his nose tickled into my curls and his nervous breath steamed between my legs. His tongue made one tentative swipe just below my clit, and he recoiled with a gasp. For one tense moment, I worried that I tasted terrible to him and that our liaison was about to collapse.

Then he let out a long, shivering hum of approval and went at me hungrily. What he lacked in precise skill he more than made up in enthusiasm. Every other lick hit just the right spot. I think that was actually better than constant precision. It gave me a second to recover from each perfect swipe, only to be blindsided by the next. That respite narrowed as he learned from my moans and sighs and found my most sensitive spots.

I loved the sounds. There's nothing more sordid and indulgent than the clicks and pops of an eager tongue against a wet pussy. His desperate breath and little growls were wonderful, too. He was making a glorious mess of things in my lap. The wild wiggling of his tongue just below my clit drove me mad. Once he learned to alternate that with long, slow licks from below my pussy up into my bush, I was slipping and sliding closer and closer to orgasm. I can't recall how much noise I made, but I'm pretty sure I held nothing back and even cursed a few times.

I didn't want to come just then, worried that we might fall out of sync. I had to force his face from my pussy to get him to stop. He looked up at me with this wonderful, eager-to-please look, his face all spicy and shiny. I knew I needed to get him off before he burst. It wouldn't hurt me much, either.

I coaxed him onto his back on the bed and climbed up over him. He looked up anxiously as I loomed over him, and I'm sure I was smirking at his predicament. It was time to fully indulge my need and to put him out of his misery. Eyes locked with his, I held his cock as I sat back. Between my own excitement and his frantic licking, my pussy was incredibly slippery. The head of his cock slid inside with another one of those deliciously dirty clicks.

Mouth open in wonder, I drank in his look of amazement as I slowly engulfed him. His eyebrows would have leapt off his head if they had risen any further. His hands latched onto my thighs, as he desperately prepared himself for the ride ahead. I ground myself down until my bush, wet from his mouth, tangled with his.

I moaned, feeling completely filled. Then the need took over. No more subtlety. No more drinking in his reactions. I completely switched off. The only thing on my mind was getting off spectacularly. It was time to fuck. My hips rocked violently, levering my body against his. The wet sounds grew louder and more frequent as I fell into a pounding rhythm. Outside of my own intense sensations, I was only aware of his cock and his hands holding onto my thighs for dear life.

We were a perfect fit. Completely connected. It felt like my pussy was sucking his cock. I could feel the heat pouring back and forth between our bodies. I kept driving us toward the cliff. Even so, I was still surprised when orgasm crashed into me. All of my muscles locked. I felt Eric's cock wring up inside me, but I was too wet to feel his release.

I was loud. I'm sure of it. I drowned out any noise that Eric might have made. That's a shame, because I really wanted to hear him as he came. But I was too busy being tumbled by the sharp, intense waves of my own orgasm.

I stayed atop him for the longest time, catching my breath and getting my wits back. We just watched each other for a while. Every now and then my pussy would involuntarily squeeze his slowly softening cock as though saying "thank you." We talked for a bit, but I was surprisingly drained and kept my replies to a minimum as my thoughts whirled around drunkenly. However, Eric was eager to write to you, so I obliged him, pouring the last of my energy into my laptop and this letter.

Eric intrigues me and arouses me in just the right way. I think we'll get along famously (especially if he lets me get some rest after my hard work). I can't thank you enough, Veronica, for pressing us together. There's so much I could've missed.

Gratefully,

Sonia

***

Sonia made it sound so good I wish I had been there. She really has a great touch with detail. I also love how different their accounts are, despite the fact that both were there experiencing the same things. And while I'm sorely tempted to interject myself and have a fling with the two of them, I think it would be better to leave them to themselves and move on to another story. So that's what I'll do - beginning next month.

Until then, I wish you all wonderful love,

Aphrodite

Breakout

Review by Purple Dragon

Game Info:	Breakout
Author:	Grimm Sharlak
Release Date:	April 1, 2004
Platform:	ADRIFT 4.0
Size:	24KB
Content:	mf(alien)
Type:	ANW/PF
Length:	Short
Reviewed:	May 2009
Extras:	None

Basic Story

You are a janitor (first class) in a government building housing an agency responsible for policing the increasing wave of extraterrestrials visiting Earth. When an exceptionally beautiful one is arrested and brought in you decide to help break her out, and hopefully get yourself some in the process.

Overall Thoughts

I liked the basic premise of the game, and the story and characters (particularly the PC) were fairly well fleshed out. The writing in general was also very high quality, which just makes it more fun to play a game, even if you end up having problems with parts of it. It was an entry in the 2004 mini-comp and fit well into the rules, with good use made of the three room limit, and decent reasons give why you can't wander around the rest of this supposedly huge building.

Puzzles/Game Play

This game is very puzzle intensive, which is why I added the "Puzzle Fest (PF)" classification, even though it is a mini-comp game, and would normally fall into the "A Night With…" category. I won't say that I love puzzle intensive games, but I certainly don't hate them either, and most of the ones in this game are fairly logical and well done. There are, however, a couple of exceptions to this, which I'll discuss below.

Sex

The sex scene is pretty well done. Even though it is technically with an alien, and not a human girl, she is described as sufficiently hot that I doubt many people would object on that account. The downside of the scene is the length. Even though this was a mini-comp entry the scene seemed a bit short, and some people (particularly those who don't like a lot of puzzles in their games) might say that the reward wasn't worth the effort it took to get there.

Technical

Technically speaking, there were a lot of good things about this game. There are a lot of objects, a lot going

on in a relatively small space, and for the most part, a lot of attention to detail. Considering the fact that there are only three rooms, I still felt that it was a very rich environment. Unfortunately, there are a couple areas where things just didn't work out the way that (I assume) the author intended them to. Specifically, one of the puzzles had (at least in my opinion) a big guess-the-verb problem, and there is another area where it is possible to get to a place where the game becomes unwinnable.

Final Thoughts

The game had a good plot, good writing, and a good sex scene, although a little shorter and more prosaic than it could have been considering the fact that you are having sex with an alien that is supposed to be "born to fuck." If you don't like puzzles, this might not be your cup of tea, and a couple of guess-the-verb problems kept it from being as smooth as it could have been. Still, it's well worth a play.

Rating: C+

Freshmen Orientation

Review by Purple Dragon

Game Info:	Freshmen Orientation
Author:	Villainy
Release Date:	April 1, 2004
Platform:	ADRIFT 4.0
Size:	22KB
Content:	mf
Type:	ANW
Length:	Short
Reviewed:	May 2009
Extras:	None

Basic Story

You play a college senior who has arrived at school early for the express purpose of scoring with a few of the hot new freshmen during their orientation.

Overall Thoughts

The high school or college fuck fest has become something of a cliché setting in the AIF world. Of course, it's a cliché because it's a damn good setting. Being a mini-comp entry the author didn't really have space to flesh out the campus, but it still works as a way to get us from point A to point B.

Puzzles/Game Play

There are a few fairly simple puzzles leading up to the main event. They are logical for the most part. In fact, a couple of things are maybe a bit too realistic, in that if you get caught carrying any incriminating evidence

it could mean an abrupt halt to the game. I actually like this as it shows that actions have consequences in the game environment and it works well.

Sex

I really liked the descriptions of the body parts, and considering how short of a game this was I was pleased to see that even though she wears three different sets of clothes throughout, the descriptions change accordingly. The sex itself was pretty good, but I found many of the segments to be shorter than I would have liked. I had just typed in what I thought to be the final command, fully expecting the game to end, when I realized that I had one more opportunity to enter a command. There are five options for that last command, and you should save right before so you can go back and read them all like I did. Here is the length that I wanted earlier and those final five options are the best and hottest descriptions in the game. It was a nice touch and a great ending.

Technical

There were a few bugs in the game, although nothing that would keep you from finishing it. I found the command to actually get Becky into your room to be a little less than intuitive. I was actually hitting all around it, but it took awhile to hit it on the head. Most of the bugs that I found were more humorous than annoying, and while it still knocks a few points off, most of them were forgivable, especially for a first time author.

Final Thoughts

Although a bit bland and lacking much of anything in character development, this is still a quick, fun little game with some hot sex writing, and I think you'll enjoy it.

Rating: C

To Score or Not To Score

Review by Purple Dragon

Game Info:	To Score or Not To Score
Author:	David Whyld
Release Date:	April 1, 2004
Platform:	ADRIFT 4
Size:	24KB
Content:	mf
Type:	PF
Length:	Short
Reviewed:	May 2009
Extras:	None

Basic Story

The time has come to finally go all the way with your girlfriend Jennifer. She's not actually aware of that fact yet, but that's a small detail.

Overall Thoughts

This is a very clean, well implemented game. Although there isn't much in the way of plot, there is a fair amount of character development, and the ending varies depending on choices and interactions during the game, which is a nice touch.

Puzzles/Game Play

The main puzzle in this game is centered on the cauldron in your room. The cauldron is a form of hyper-dimensional transmogrification device using – well, I'll spare you the scientific mumbo jumbo. Basically, what you do it put two items into the cauldron, hit it, and if they are the right items something else is created in their place. I found some of the pairs to make at least a little bit of sense, but there were some that made no sense at all, so it's basically just up to trial and error. If you don't like puzzles (and this one is a bit tedious even if you do) then you can just put all the items you find inside and periodically hit the cauldron. Correct pairs will be combined, while ones that don't pair will remain in the cauldron. There are, however, a couple of items that you use elsewhere, so not every item will pair with another.

The items that are created all go toward helping you get into your girlfriend's pants. They are either gifts for her or other items that will help you on your way.

Sex

This is certainly the weak point of the game, or rather the weak point if you're expecting an AIF game to have a lot of sex in it. There really is no sex scene as such. Although sex is constantly on the PC's mind, and there are a few teasers in the form of his plans and dreams along the way, when the actual sex scene comes it is limited to one or two short paragraphs before the game ends.

Technical

Mostly, this is an incredibly tight game technically speaking. The environment is full and interactive, and the different days and conversations with Jennifer work well. The conversations actually affect your relationship and, along with the gifts you give her, determine which of the five endings you end up with. If you get the best ending (which I finally did, although I had to cheat) the author gives the password for the game so that you can go in and take a look. When I did that, I was even more impressed with how detailed the system is and realized that I didn't make full use of it.

Ok, that's the good. The bad is that there are actually a couple of annoying bugs in then game. The first has to do with the bookcase in your room. If you examine it, the only book that stands out from your large collection of hard core porn is a "horror novel." Seeing as how it is mentioned specifically, I naturally tried to examine it, but was told that I "see no such thing." Undaunted, I typed "take horror novel," and received the authors rather amusing comments about typing things not planned for. I went on about the game, but couldn't get the best ending, so I finally consulted a walkthough and found that the command is "get horror novel." Normally "get" and "take" are synonymous. However, in this case the command is actually a task written by the author, and since he didn't manually provide for this synonym, "take" won't work. It should be noted that some people would never even notice this, but for whatever reason I have always used the word "take" so it caused me some problems.

The second one wasn't quite as big a deal for me, but I thought I'd still mention it. There is a sign in the store that changes every day to reflect new items on sale. On day three, examining the sign just tells you there is nothing special about it. The fact that I got the default response instead of one saying there was nothing for sale alerted me to the possibility of a problem, so I simply tried each denomination from 1-9 until I had found everything. For your reference, the prices of items on day three are $2, 4, and 6.

Final Thoughts

Although the main puzzle was a bit annoying, the rest of the system to go through the days and conversations leading up to the finale worked beautifully. However, it would have been nice if it had led up to a more fully implemented sex scene since this was an entry in an AIF competition.

Rating: C+

This month we're going to be discussing transportation a bit. Obviously, the usual form of transportation in games is walking. Although you probably don't consciously think about it most of the time, that is usually the case. But what if you want to get to someplace a long way off or lug something around that is really too heavy or awkward to carry? Well, you could just allow the player to pick up that treasure chest and carry it from the enchanted cave, across the steeps of doom, and through the tunnels of turmoil back to his home on the plains of perpetual pleasure, but let's just say that we want to add a touch of realism to our game. You're going to need a vehicle of some sort, and that is what this month's article is all about.

Create a vehicle for the player. You also need to create several rooms (very minimal descriptions are fine) so that we have a way to test it out. Here are the tasks we want to accomplish.

1. *The player can walk between rooms like normal, but when in his vehicle, there should be a "drive" command that will take him to a specific room instantly (or rather in one turn as opposed to however many it would take to walk there).*
2. *The player should also be able to "drive [direction]" to move one room at a time like he was walking. Be sure to include at least one indoor location so that you can show how to disallow him driving into his living room (or wherever).*
3. *Make one room that the player cannot walk to (because it's too far away or whatever) and allow him to reach it by driving.*
4. *One task involves picking something up from one location and taking it to another. The object in question is too awkward or heavy to be picked up and carried around, so the player will need to drive to that location, load it into the vehicle, and then drive it to the new location.*

TADS 2 Segment by A. Bomire

In this month's "Coder's Corner", we will be talking about vehicles. Be it a plane, train, or automobile - something that the player can get inside and move about. Let's set up the situation. It is summer, and the player's girlfriend's window air conditioner has gone out. She purchased a new one at the local appliance store, and she needs him to pick it up for her. If he does – she'll be VERY appreciative. To do so, the player will need to drive his car to the appliance store (it's too far to walk), load the air conditioner into the car and drive it back.

Before we begin discussing the mechanics of creating a vehicle, let's set up our environment. We'll need an apartment for the player's girlfriend (which I'll call Erin), a street environment for his car, another street area outside of the appliance store, and

just for good measure we'll throw in a couple more street scenes to drive around within. Part of the journey will take place on the highway, which is unsafe for walking so that the player has to drive. We'll start out in Erin's apartment:

```
startroom: room
  sdesc = "Erin's Apartment"
  ldesc = "You are in the apartment of your girlfriend, Erin. It
   is a nice apartment - much nicer than yours, to tell the truth.
   The only downfall to it is that her window air conditioner is
   broken, and it is starting to become unbearably hot in here.
   You can exit her apartment to the south. "
  south = ElmStreet
;

Erin: Actor
  location = startroom
  sdesc = "Erin"
  adesc = self.sdesc
  thedesc = self.sdesc
  ldesc = "Erin has been your girlfriend for over a year, and the two of you get
along well. She is a very pretty girl, with a great body. Right now, she is a
little sweaty. While this causes her thin t-shirt to cling to her body in a way
that makes you a little sweaty as well, she definitely isn't in a good mood. She
would very much appreciate it if you could pick up her air conditioner. "
  noun = 'Erin' 'girlfriend'
  actorDesc = "Erin is here, sweating slightly in the heat. "
;

ElmStreet: room
  sdesc = "Elm Street"
  ldesc = "This is the street where your girlfriend lives. In fact, you can enter
her apartment to the north. Her apartment building is at the end of a cul-de-sac,
with the street leading to the west. "
  west = Intersection
  north = startroom
;

Intersection: room
   sdesc = "Intersection with Main Street"
  ldesc = "Here is where Elm Street and Main Street intersect. Elm Street continues
on to the west into another subdivision, but you really don't know anyone who
lives there. Going east on Elm will take you to your girlfriend's apartment. Main
Street runs to the north and south. South leads downtown, and north leads to the
main highway - which will take you to the appliance store. "
   south = "There's nothing of interest to you downtown. "
   north = Highway
   west = "There's nothing of interest to you there. "
   east = ElmStreet
;

Highway: room
  sdesc = "Highway"
  ldesc = "This highway parallels Main Street, and provides a bypass to the city
traffic. It is the quickest way to the mall area to the north and back again to
your girlfriend's apartment to the south. "
  south = Intersection
  north = Mall
;
```

```
Mall: room
  sdesc = "Mall"
  ldesc =
  {
    "You hate the Mall, but your girlfriend loves it. Actually, she loves shopping,
but the two go hand in hand. She purchased her air conditioner from one of the
stores here. You can return to the highway to the south. ";

     if (not self.isseen)
       "\bYou arrive at the Mall and go into the appliance store.
        You spend 15 minutes waiting for a clerk to finish complaining to another
clerk about the hours he has to work this weekend before he finally comes to talk
to you. After another 15 minutes of explaining that you aren't here to buy an
air conditioner, you just want to pick one up, the clerk disappears in the back to
bring out the appliance. Another 30 minutes later, and another clerk appears to
let you know that the clerk you spoke to is on break and what did you need again?
You go through the whole  thing again, and another 20 minutes later you finally
have  your air conditioner. By the way, you HATE the Mall! ";
  }
  south = Highway
;

airConditioner: item
  location = Mall
  sdesc = "air conditioner"
  ldesc = "This box contains your girlfriend's air conditioner. It is fairly heavy
- not too heavy to lift, but definitely too heavy to carry any real distance. "
  noun = 'conditioner' 'box'
  adjective = 'air' 'heavy'
;
```

Okay, that seems like enough for right now. As I say every month, in a real game I would expand upon all of that with additional rooms and objects to fill out the environment, but for our purposes that will do nicely.

Next, we'll need a vehicle. In this case, a car. TADS is nice in that it has anticipated the author's need for such conveyances, and has covered within its documentation not one but two types of vehicles. (See Chapter 10 of the TADS manual, Advanced TADS Techniques.) The first type of vehicle is a completely enclosed vehicle – something that the player enters and then conveys him to some other location. This type of vehicle could be an elevator, or a plane, or a train/subway or something where the player doesn't really control the vehicle - other than selecting a destination through purchasing a ticket or pushing a button. This type of vehicle is nothing more than a normal *room*-object, where the exit is variable depending upon choices made by the player. The other type of vehicle is something where the player has complete control over it, like a bicycle, raft, or other "small" vehicle. In fact, this type of vehicle is something that the player can pick up and carry about with him. TADS has given this type of vehicle its own class, *vehicle*, but it shares its properties with other objects such as chairs, beds, couches, etc. These are all considered to be small rooms within another room, or a *nestedroom*. The difference is that the *vehicle*-class objects can be picked up and carried by the player, while chairs, beds, etc. cannot.

Of these two types of rooms, our car is somewhat of a mixture between the two. It isn't entirely enclosed, as the player should be able to see the surrounding area. In other words, the car is of the room-within-a-room variety, which places it in the *vehicle* class. However, the car isn't something that the player can pick and carry with him – unless, of course, you are writing your game in some futuristic Jetson's-like setting where the car folds up into a briefcase. This sample game doesn't take place in that era, so our car is like other cars – big and heavy. We could design our car from scratch, starting with the *nestedroom* class, but TADS' *vehicle*-class is close enough to what we want that we should be able to simply modify it to disallow the player from moving it. First of all, we'll make it a multi-class object (an object which has more than one class), and make it a *fixeditem*. This will make it immovable. But, *fixeditem*-class objects aren't listed in the room description. No problem – this is controlled by a property of the *fixeditem* class called *isListed*. If this property is nil (which it is by default), the object won't be listed in the room description. If we set it to true, then it will be listed. For more information on this, read the TADS manual, Appendix A, and look for the *fixeditem* class.

```
yourcar: fixeditem, vehicle
  location = ElmStreet
  sdesc = "your car"
  adesc = self.sdesc
  thedesc = self.sdesc
  ldesc = "This is your car. It isn't the best car, but it gets you from here to
there, which is all you really need. "
  noun = 'car'
  adjective = 'your'
  isListed = true
;
```

One last thing we need to concern ourselves with (for now, there will be others later) is dropping items. By default, when you drop something in a *nestedroom*, TADS moves it to the enclosing room. For example, if you drop something while sitting in a chair, you don't expect to see that object in the chair, you expect it to be lying on the floor of the room. This will be important for us, because eventually we want the player to be able to transport the air conditioner within the car without having to carry it (which would be awkward, if not impossible, in real life). Not to worry, however, as TADS provides a simple solution. By setting a seldom used property called *isdroploc* to true, our vehicle becomes a location wherein a player may drop things without them passing to the enclosing room. So, we need only make this tiny addition to our car:

```
yourcar: fixeditem, vehicle
   ...This is in addition to the previous definition
   isdroploc = true
;
```

Now that we have our car, we can test it out. Try getting into and out of the car, and looking around. TADS will report you as being "in your car" when you are in it, but otherwise you can look around just as if you were standing on Elm Street as usual. Go pick up the air conditioner and carry it back to the car. Get in and drop it, then exit the car – it should stay "inside". When you attempt to move while inside of the car, however, TADS tells you that you will have to exit the car first. Now what do we do?

There are many different ways of handling this. How do you want your car to act? Do you want the player to simply be able to move in the necessary direction (i.e., when the player types "north", the car and player together move into the location that is to the north)? Or, do you want the player to use special driving commands (i.e., "drive north" will take the player and car together to the location that is to the north)? Maybe the locations accessible through the car are inaccessible for the player otherwise (i.e., the player cannot move to the locations using normal movement commands). All of these present different challenges, with different solutions. For us, we are going to have the player use a special command to operate the car. The command will be to "drive". If the player types "drive", with no objects, then he will be presented with a menu of locations (for us, it will just be two: the mall and Elm Street). If he types "drive [direction]", then the car will move in that direction (if possible). For example, "drive north" while in the Intersection will take the player (and his car) to the Highway.

Let's set up the first scenario: entering "drive" by itself. This will involve creating a new verb, which I will call *driveVerb*. With no direct object, the action performed by the verb will be handled by the *action(actor)* method/property of the verb. Since there is no verification process for this, we will have to perform all of our verification within the method. If the player isn't currently in the car, then we will display a message to the effect of directing him to enter the car first. Instead of displaying a menu of options, if the car is currently at Elm Street or the Mall we will take the player directly to the other location (the Mall or Elm Street, respectively). Otherwise, we will present the player with a menu prompting him to choose one of those two destinations. This latter option will involve prompting the player and getting his input. We covered this in our last "Coder's Corner" when discussing setting the alarm for the watch, so we won't go into terrific detail here. For more information, consult the TADS manual, Chapter 8 – Language Reference on the use of the *input()* function. Lastly, once the car has been moved, we will need to display the room just as if the player had moved there normally.

This last bit is not something we have covered, so I'll go over it. When the player enters a room, TADS has a special bit of code for each room called *enterRoom*. It handles such things as displaying the room description (in either verbose or terse mode), and setting the flag that tells TADS that the player has seen the room. We could duplicate this code for our own movement, but why bother? Why not simply invoke the normal *enterRoom* code when we move into the new room? For the same reason, we'll also invoke the *leaveRoom* code when we leave the room the car is currently within.

Using this, let's write our *driveVerb* code.

```
driveVerb: deepverb
    sdesc = "drive"
   verb = 'drive'
   action(actor) =
   {
      local x, moveLoc := nil;

      if (actor.location <> yourcar)
        "You really can't drive anywhere unless you are in your car! ";
      else
      {
          if (yourcar.location = ElmStreet)
            moveLoc := Mall;
          else if (yourcar.location = Mall)
            moveLoc := ElmStreet;
          else
          {
             "To where do you want to drive:
             \n\t1) The Mall
             \n\t 2) Elm Street
             \n\t ? ";
             x := input();
             if (x = '1')
               moveLoc := Mall;
             else if (x = '2')
               moveLoc := ElmStreet;
             else
               "That wasn't really one of the valid options, was it?
                  Instead of going somewhere, you just idle. \b ";
          }
          if (moveLoc <> nil)
          {
            "You drive your car to <<moveLoc = Mall ? "the Mall" :
            "Elm Street">>. \b";
            yourcar.location.leaveRoom(Me);
            yourcar.moveInto(moveLoc);
            yourcar.location.enterRoom(Me);
          }
       }
    }
;
```

This should take care of driving directly to the Mall or Elm Street, but what about driving through town? For this, we will need to set up additional coding to handle "drive north", "drive south", etc.

Because "north", "south", and other directions aren't actual objects but are instead verbs, you cannot simply add a *doAction* property to the *driveVerb* above, and then modify the travel verbs to handle the player asking to "drive" them. (Okay, EVERYTHING in TADS is an object – it is an object-oriented language. However, verbs aren't objects that the player can manipulate, which is how I meant that statement.) So, instead we are going to have to explicitly build verbs that handle "drive north", "drive south", etc. And, these verbs will need to move the car (along with the player and anything else within the car) into the room/location that corresponds to the surrounding room's *north*, *south*, etc. If you read over TADS' example in Chapter 10, you'll find that TADS creates a raft which once launched automatically floats downriver until the player decides to get off. In other words, the player controls entering and exiting the raft, but not the motion of the raft. This isn't very helpful.

Our little scenario is small enough that we could simply custom write some code such that if the player/car is in *this* room and commands "drive north", then he is move to *that* room. But, I'm going to construct a vehicle that could be dropped into any game, which could have varied and complex directions such that custom writing an exit possibility for each room would be impractical.

When we define an exit for a room, we can make it lots of things. It could be nothing (i.e., not defined) in which case TADS automatically substitutes the *noexit* property of that room. It could be a location, such as Highway or Mall. It could be a message, such as we defined for *Intersection.south*. It could even be a combination of these as we conditionally supply either a message or a location. For example (and we will be doing this later), the player may not be able to drive to certain locations (like into Erin's Apartment) although that exit may be elsewhere. So, the value of a room direction property can be one of three things: undefined, a message, or a location. Hmm...sounds complicated. What makes it even more complicated is that testing a property will evaluate it. How does that complicate it? Well, consider this piece of code:

```
if (isclass(yourcar.location.south, room) )
     "It is a room. ";
else
     "It is not a room. ";
```

First of all, that will get you a nice fat TADS error, because you are testing a property to see if it is a type of object – a mismatch in type. But, TADS will evaluate *yourcar.location.south* before attempting to test its class – which means that if the car is located in Intersection, it will display the message "There's nothing of interest to you downtown.", and then give you an error. Hmm...sounds very complicated. Perhaps we'll just move on to "Coder's Corner 5"? No? *sigh*

In reality, this seems more complicated than it is, and here's why. A message has no value. Well, it has value in that it displays information to the player, but it has no assigned value. To TADS it is functionally equivalent to *nil*. The same is true for an undefined property. Not being defined, it has no value either. It, too, is *nil*. However, an exit that has a location assigned to it has a value. That value is not some number or true/false value, but it isn't *nil*. We can use this to our advantage by doing something like this:

```
if (yourcar.location.north <> nil)
{
    yourcar.location.leaveRoom(Me);
   yourcar.moveInto(yourcar.location.north);
   yourcar.location.enterRoom(Me);
}
```

For more information about nil/true, see the TADS manual, Chapter 8 – Language Reference.

When the player moves about, TADS invokes a method defined for *Actor*-class objects called *travelTo*. This method is slightly more involved than what we need, but the basics are what we've outlined above. The additional coding has to do with passing through doors, into and out of lit rooms, etc. To be thorough, we really should incorporate these additions into our game, but since all of our rooms are going to be lit and we are going to handle passing through doorways a different way, then we can get away with a simpler definition. So, let's modify our definition of *yourcar* by adding this additional method. While we are at it, we'll also take advantage of the fact that a *nestedroom* is just a different type of *room*, i.e., it, too, can have directions (north, south, east, west, etc.). By default, if no direction is defined, then TADS invokes the generic *noexit* – and we will use this to instruct the player to use the "drive" commands (which we will define in a moment) to move about while in the car. Lastly, we will preempt three directions that the player may try to use while in the car: in, up and down. ("Out" will already have the player exit the vehicle.)

```
yourcar: fixeditem, vehicle
 ...This is in addition to other code we've written
   in = "You are about as far into the car as you are going to get. "
 up = "Unfortunately, this is not a flying car. "
 down = "This car is not capable of tunneling under the street. "
   noexit = "If you wish to leave the car, then \"stand\" or \"get out\".
  If you want to drive the car somewhere, then \"drive\", or preface your direction
with \"drive\". Example, \"drive north\" to go north in the car. "
 travelTo(room) =
```

```
    {
      if (room <> nil)
      {
        yourcar.location.leaveRoom(Me);
        yourcar.moveInto(room);
        room.enterRoom(Me);
      }
    }
  ;
```

Next, we will need to write a series of verbs to handle driving. There are eight compass directions we can "drive" to – north, south, east, west, northeast, southeast, northwest, southwest. There is no simple way around it, we will need to write eight separate verbs to handle this. To save time and space, I will only write one of them. The rest are identical, with the exception of changing the direction:

```
  dnVerb: deepverb
    sdesc = "drive north"
    verb = 'drive north' 'drive n'
    action(actor) =
    {
      if (actor.location <> yourcar)
        "You can only \"drive\" while in the car. ";
      else
        yourcar.travelTo(yourcar.location.north);
    }
  ;
```

And the above would be repeated for "drive south", "drive east", "drive west", "drive northeast", "drive southeast", "drive northwest" and "drive southwest". In our game, you would actually only need four directions: north, south, east and west. However, it would be good practice to include the other four directions as well as you never know just how your game may change as you write it.

Okay, we're pretty much close to finished. There are just a few things to finish up. The first thing is that we don't want to allow the player to drive the car into Erin's apartment. There are a few ways of preventing this, but the simplest is to modify the *north* property of ElmStreet to disallow the player going north while in the car. This will translate over to when the player attempts to "drive north" while in the car, as it ultimately references the *north* property. So, let's make that change now:

```
  ElmStreet: room
    ..alter the north property of this room to reflect this change
    north =
    {
      if (Me.location = yourcar)
      {
        "You can't drive the car into Erin's apartment! ";
        return nil;
      }
      else
        return startroom;
    }
  ;
```

Next, we want to make the Mall too far for the player to walk there. So, we'll modify the Intersection so that the player has to be in the car to travel north:

```
  Intersection: room
      ...just modify the north property to reflect this change
      north =
```

```
        {
            if (Me.location = yourcar)
               return Highway;
            else
            {
               "The highway isn't a safe place to walk. Besides, it is a long way
                from here to the Mall. ";
                return nil;
            }
        }
    ;
```

Since the player cannot walk to the Mall, he certainly cannot walk back, so we should make a similar change to the Mall location. While we are at it, we will clean up the description of the player arriving at the Mall. Right now, it describes the player talking to clerks and getting the air conditioner – all things that would take place outside of the car. To make the description consistent with the player's actual location when he arrives at the Mall the first time, we should move the player out of the car and into the Mall room:

```
    Mall: room
        sdesc = "Mall"
        ldesc =
       {
            "You hate the Mall, but your girlfriend loves it. Actually, she loves
    shopping, but the two go hand in hand. She purchased her air conditioner from
    one of the stores here. You can return to the highway to the south. ";

             if (not self.isseen)
             {
                 "\bYou arrive at the Mall and go into the appliance store. You
    spend 15 minutes waiting for a clerk to finish complaining to another clerk
    about the hours he has to work this weekend before he finally comes to talk to
    you. After another 15 minutes of explaining that you aren't here to buy an air
    conditioner, you just want to pick one up, the clerk disappears in the back to
    bring out the appliance. Another 30 minutes later, and another clerk appears
    to let you know that the clerk you spoke to is on break and what did you need
    again? You go through the whole thing again, and another 20 minutes later you
    finally have your air conditioner. By the way, you HATE the Mall! ";
                 Me.moveInto(Mall);
             }
        }
        south =
        {
            if (Me.location = yourcar)
               return Highway;
            else
            {
               "The highway isn't a safe place to walk. Besides,
               it is a long way to Erin's neighborhood. ";
               return nil;
            }
        }
    ;
```

Last, but certainly not least, the player should be prevented from driving anywhere while holding the air conditioner. (Have you ever tried driving with a big, heavy box in your lap?) This will involve making modifications to our "drive" command, as well as our eight "drive *direction*" commands. I won't repeat all of the code for these commands, but you should add something similar to this to all of them:

```
if (Me.location <> yourcar)
  "You can only \"drive\" while in the car! ";
else if (airConditioner.location = Me)
  "You can't drive while holding the air conditioner! ";
else
    ...proceed with normal handling as previously defined
```

And, that should do it! In an actual game, we would proceed from here to coding up Erin's reaction to you arriving with her air conditioner – to which she will be VERY grateful! Or, perhaps this would lead to another puzzle in which the player would need to figure out how to install the air conditioner! Those options I leave up to you.

TADS 3 Segment by Knight Errant

This month in Coder's Corner, we'll be tackling vehicles. There really haven't been many vehicles in AIF games over the years, but if you were thinking of trying one out, here's how.

Before we get to the vehicles themselves, I'm going to introduce a new concept, the NestedRoom. Now, normally characters are simply in a location, defined as a room. Normally, there's no real concept of "space" in the room, everything's just there. NestedRooms provide a way to be more specific with a character's location. It gives them a more specific "place" in the room to be. This can be something as simple as a chair. Anything that the player can be in or on while in a room is a type of NestedRoom. One type of predefined NestedRoom is the Vehicle class. A vehicle overrides the normal movement controls, so while the PC is inside the vehicle and types a movement direction (such as north), it is the vehicle itself that will move and all of it's contents (including the PC) are carried along with it. We can define one now:

```
carpet: Vehicle, Bed, Immovable 'magic persian carpet/rug' 'magic carpet'
    "You imagine this Persian rug was quite beautiful when it was new, but
now it's faded and threadbare.  "
    specialDesc = "A Persian rug lies here. "
    useSpecialDesc = (!gPlayerChar.isIn(self))
;
```

In addition to being a Vehicle, the carpet is also of class Bed, so that the PC can stand or lie on it. It's an immovable because we don't want the PC to be carrying it around. However, one of the consequences of Immovable is that Immovables are not included in room descriptions by default. To fix that, we have a specialDesc. We also set useSpecialDesc so that the carpet is only listed in the room when the player is not in or on it.

Now that we have our vehicle, we'll want to have somewhere to go with it. Simply by defining additional rooms we'd have somewhere to go, but then we'd still have to move from room to room manually like chumps. We've got a magic carpet, we should be able to fly where we want, right? There are two ways of doing this. We could use the TADS 3 pathfinding extension, pathfind.t Look for it in the extensions folder, and add it to your game. To utilize it, we need to add roomPathFinder to the Class List for the PC. Or, we could take the lazy way out and simply use moveIntoForTravel. I'm feeling lazy at the moment, but if you're actually interested in automated movement I highly recommend looking into pathfind.t. Note: a consequence of using moveIntoForTravel is that in a real game it could result in allowing the player to get to locations behind locked doors or similar places he shouldn't be allowed to go.

Now, the fly verb. Defining the verb itself is pretty much the same as any verb we've defined so far. The only difference is in the action method of Room (the only valid places to fly to). I've based this code on the NC Debug Actions code by Nikos Chantziaras. Here we go!

```
modify Room {
   dobjFor(Fly)
   {
         verify()
                {
```

```
                        if(gActor.location!=carpet)
                        illogical('You can\'t fly under your own power!');
                  }
            check() {}
            action()
            {
                  // Destination.
                  local dest = gDobj.ofKind(BasicLocation) ? gDobj :
                  gDobj.location;

                  while (dest) {
                        if (dest == carpet.location) {
                              // The location we found is the same as the PC's
                              // current location.  No need to fly.
                              "You are already there. ";
                              return;
                        }
                        if (dest.ofKind(BasicLocation)) {
                              // We found a BasicLocation.  Let's fly there.
                              "Grabbing hold of the carpet, you fly off to your
                              destination! ";

                              // Remember my original location - this is the
                              // location where the PC was before he flew.
                              local origin = carpet.location;

                              // Tell the old room we're leaving.
                              if (origin) origin.travelerLeaving(carpet, dest,
                              nil);

                              // Move to the destination.
                              carpet.moveIntoForTravel(dest);

                              // Tell the new room we're arriving, if we changed
                              // locations.
                              if (carpet.location)
                                    carpet.location.travelerArriving(carpet,
origin, nil, nil);
   gActor.lookAround(true);

                              // We're done.
                              return;
                        }
                        // One level up.
                        dest = dest.location;
                  }
            }
      }
}
```

In the Verify method, we simply check if the player is on the carpet. If not, he can't fly. All the meat of this is in the Action. First, we get the destination of the Fly command. If the destination is the carpet's current location, we stop here. If not, we can actually do something. We send a message to the current room to let it know we're trying to leave. This is so anything in the room that might try and prevent travel has a chance to interfere. Then we move to the new location, and notify the new location that we've arrived. This is so anything in the new room has a chance to react to our arrival. Finally, we have the player look around, to give the room description.

Right! Now we can move from room to room on the flying carpet, or we can instantly fly to any location on the map, so long as we've defined vocabWords for the destination. Now, just for fun, we can define a roomConnector with a TravelBarrier so that the player can only go across a particular junction if he's on the carpet.

```
roomTwo: OutdoorRoom 'Large field' 'large field'
    name='large field'
    vocabWords='large field'
    south=roomOne
    up=sky
;

sky : RoomConnector
    room1 = roomTwo
    room2 = topOfTower
    canTravelerPass (traveler) { return traveler==carpet;}
    explainTravelBarrier(traveler)
    {
        "The tower is far too tall to climb.";
    }
;

topOfTower: OutdoorRoom 'Top of the tower' 'top of the tower'
    name='top of the tower'
    vocabWords='top tower'
    down=sky
;
```

Now the player can only get to the top of the tower if he flies there. If we wanted, we could define an object that's too heavy to be moved by hand and has to be moved by carpet. We can make it a Heavy class, and define our custom handling for PutIn.

```
PileOfGold: Heavy 'pile gold coins' 'pile of gold'
   "It's a pile of gold coins. "
   dobjFor(PutIn)
       {
   verify(){if(gIobj==carpet) logical;}
       }
   dobjFor(Take)
       {
   verify(){}
   action(){inherited;"You pick up the pile of coins.  It's too heavy to
carry, so you set it down."; nestedActorAction(gActor,Drop,PileOfGold);}
;
```

We don't need to define the action for it because it will simply inherit from Thing.

There's a lot more that can be done with vehicles, of course. TADS 3 also defines a VehicleBarrier which can block all vehicles from crossing a passageway. You can play around with the various options to adapt this scenario to however it may fit your plot. See you next month.

Inform 6 Segment by 'trix

Hello again, minions.

This month: vehicles. A vehicle is an object that the player can get into, and then can move from room to room in it. We'll start by defining a basic game with a few locations and a vehicular-looking object.

```
Constant STORY = "My Tank";
Constant HEADLINE = "^An interactive vehicle of some kind^";

Include "Parser";
Include "VerbLib";

Object garage "Garage"
 with  name 'garage',
       description "This is a completely ordinary garage, containing
          nothing remotely out of the ordinary. The way out is
          north.",
       n_to yard,
 has   light;

Object -> tank "tank"
 with  name 'tank',
        description "A tank much like every other tank you've ever stolen from
 the army.",
       describe "^Your tank is here, ready for action.",
 has enterable open container;

Object yard "Yard"
 with  name 'yard',
       description "This is the yard where you usually drive your
          tank. Your house is westwards; south leads towards the
          garage; and to the north is the supermarket.",
       s_to garage,
       w_to house,
 has   light;

Object house "House"
 with  description "This is your charming house where you live. The way out is
east.",
       e_to yard,
 has   light;

[ Initialise; location = house; ];

Include "Grammar";
```

The tank (which is going to be our vehicle) has the attributes `enterable`, `open` and `container`. Being an open container means that things can go inside it without having to mess about with opening and closing it. The attribute `enterable` means that the player can get inside it. One thing worth remembering is that I6 by default does not distinguish entering something from sitting on it or standing on it, nor exiting something from standing up. This is something writers generally want to fix if they're writing something posture-sensitive, but in this case it doesn't really matter. The tank's `describe` property is the message that is printed in room descriptions announcing the tank's presence. It is not printed when the player is inside it: for that you would need the property `inside_description`, which I am not using in this case.

Now for the moment, if the player is in the tank and types "`Go north`", she will be told to get out of the tank first. The way to fix this is using the tank's `before` property. There is a special rule for enterable objects which says that if they intercept a `##Go` action and return 1 then the object is a vehicle and will go in that direction. If they return 2 then the action is aborted.

```
Object -> tank "tank"
 with  name 'tank',
       description "A tank much like every other tank you've ever stolen from the
army.",
```

```
        describe "^Your tank is here, ready for action.",
        after
        [;
         Enter: "You sit in the tank.";
        ],
        before
        [;
         Take: "The tank is very heavy. You can only lift it
           a couple of inches before your arms start to hurt.";
         Go:
           if (self in yard && noun==w_obj)
           {
              print "The tank won't fit in your house.^";
              return 2;
           }
           print "You drive ", (LanguageDirection) noun,".^";
           return 1;
        ],
    has   enterable open container;
```

I've added a `Go` interception here to allow travelling in the tank but disallow going west from the yard (into the house). The `Take` interception is to provide a suitable rejection for the playing trying to take the tank, and the `after Enter` rule is there to give a suitable message when the player gets into the tank. Notice that in a `Go` action, `noun` holds the direction of travel. `LanguageDirection` is a printing rule to print the name of a direction. The tank is now a vehicle, but there's a few more things we can still do.

We're going to define a new verb, 'Drive'. The player can either drive in a direction (e.g. "drive north"), or drive to a location (e.g. "Drive to Russia"). Both require new bits of Inform learning.

```
[ DriveToSub;
   if (player notin tank)
      "You're not in a vehicle.";
   if (real_location==noun)
      "You're already here.";
   move tank to noun;
   PlayerTo(tank, 2);
];

[ DriveSub;
   if (player notin tank)
      "You're not in a vehicle.";
   <<Go noun>>;
];

Verb 'drive'
   *                                -> VagueGo
   * noun=ADirection                -> Drive
   * 'to' scope=AnyOutdoorRoom      -> DriveTo;
```

The `##Drive` action simply redirects the command to `##Go`, because we already made that work.
The `##DriveTo` action is slightly different: we want to use the routine `PlayerTo` to move the player, because it takes care of working out the location and announcing it, printing the room description etc. So we move the tank to the destination, and then use `PlayerTo` to move the player to the tank (even though she is already in it): the `PlayerTo` routine will make sure everything's set right.
The verb grammar may be unfamiliar to you as well. There are two new kinds of verb grammar here: one is the `noun=Routine` token. This parses a noun for the action, calling the given routine to see which nouns are appropriate to the action. The routine `ADirection` is an existing routine that does exactly what we need. It is used in the I6 library for the `##Go` action.

The other new token is `scope=Routine`. This calls the given routine to decide which objects are in scope to be the noun, and then parses from that set. This is necessary in this case because if you are driving to a room other than the current location, that room isn't going to be visible to the player, so it would not ordinarily be available to be the noun of an action. `AnyOutdoorRoom` is a routine we're going to write now, which should be placed before the new grammar in the source.

```
[ AnyOutdoorRoom   x;
   switch (scope_stage)
   {
    1: rfalse;
    2:
      objectloop (x has outdoors)
      {
         PlaceInScope(x);
      }
      rtrue;
    3: "That's not somewhere you can drive.";
   }
];
```

This will be called at three stages during scoping. At stage 1, the routine is called to find out whether the action will accept multiple items (e.g. "Drive to Russia and France"). Returning false at this stage indicates it will *not* accept multiple items (because that would be madness). At stage 2, the routine is called to place in scope whatever items it thinks are appropriate. In this case we're using `PlaceInScope` to do that. Returning true at this point means "I have placed in scope everything that should be in scope for this action" so that the library will not add anything else into scope. Stage 3 is reached if the parser has been unable to match the player's input against any object in scope. The usual response "You can't see any such thing." would not be appropriate, so we print a custom one.

The things we want to place in scope are rooms that the player can drive to in her tank. In this demonstration game we will allow the player to drive to any outdoor room, which means we need to tell the machine which rooms are outdoors. I'm going to use an attribute for this.

Defining a new attribute is easy: put `Attribute outdoors;` near the beginning of your code (I'd suggest just after `Include "VerbLib";`). Now put the `outdoors` attribute on every room in the game except the house, which is our token indoor location.

Now the player should be able to drive to any outdoor location. Here's another room she might like to go in her tank:

```
Object russia "Russia"
 with  name 'russia',
       description "Dmitri always told you it was a beautiful
          country.",
 has   light outdoors;
```

Now, what else do we do with our tank? We drive our shopping home in it, shopping which might otherwise be too cumbersome to carry.

Add the following property to the yard object:

```
    n_to at_supermarket,
```

This will allow the player to walk north to the supermarket (once we've defined it), or indeed drive north in her tank.
Here's the supermarket, and the shopping.

```
Object at_supermarket "Supermarket"
 with  name 'supermarket' 'shop',
       description "You are outside your local supermarket.",
       s_to yard,
 has   light outdoors;
```

```
Object -> shopping "heavy shopping"
 with  name 'heavy' 'shopping',
       article "some",
       initial "Here's that shopping that you bought earlier and forgot to bring
home.",
       react_before
       [;
        Go:
           if (self in player && player in at_supermarket
                && noun==s_obj)
           "You can't walk all the way home carrying the heavy
           shopping. Or, at least, you're not going to.";
           if (self in player && player in tank)
             "You can't drive while you're carrying the shopping.";
        DriveTo:
           if (self in player && player in tank)
             "You can't drive while you're carrying the shopping.";
       ];
```

The react_before property prevents the player walking south from the supermarket if she is carrying the shopping. It also prevents the player from driving while holding the shopping, so she will have to put the shopping in the tank and then drive the tank home. When there, she'll have to get out of the tank to go into the house because the tank isn't allowed in the house. Dropping the shopping in the house wins the game. Or it will, once you add this property to the shopping object:

```
 after
 [;
  Drop: if (location==house) deadflag = 2;
 ],
```

That's all for now. Have fun with your tanks, everybody.

Inform 7 Segment by Dudeman

Hello everyone, welcome to another addition of coder's corner in Inform 7. Today we will be covering the topic of transportation. Just like last week, we are in luck because Inform 7 has a feature built in that makes this particularly easy to do. Inform 7 already has a predefined *kind* called a "vehicle" which is just like it is named, an object that can be entered and can be moved around from room to room with the player when he is inside.

Before we get into the finer details, let's first pause and create a little scenario along with a small world in which to test out our vehicle in.

```
When play begins:
say "When you woke up this morning and went out to get your daily paper, only to
be greeted by the sight of the most beautiful, drop dead gorgeous girl you have
ever seen unloading boxes outside of the newly sold house across the street, you
couldn't help but count your blessings and day dream at all the possible sexy
scenarios with your new hot neighbor. However, when you went over to introduce
yourself and offer to help move some stuff, before you knew it you got yourself
roped into picking up her TV from her ex-boyfriend all the way downtown. You
aren't exactly excited about running errands for her, but something tells you
that if you fetch her 'Love Mac Guffin' TV for her she might just be grateful
enough to give you some hot sex just like in a porno. After all, that is how
things really work right?".
```

```
Your Front Lawn is a room. "The front lawn of your house is neatly cut just
like always. You house is to the north and your driveway is to the west.".

Your Driveway is west of your front lawn. "A strip of cement connecting your
house with the street to the south of your house.".

Your Street is south of the driveway. "A normal asphalt street which your house
is on to the north. Your new neighbors house is also to the south while the
street continues on to the east.".

The Neighbors Driveway is south of your street. "The driveway leading to your
new neighbors house.".

A Rundown neighborhood street is east of your street. "A poorly maintained road
in a bad looking neighborhood where you new neighbor's ex-boyfriend lives. His
house is to the north while the street continues on behind you to the west.".

A dilapidated house is north of the rundown neighborhood. "Your neighbor's ex-
boyfriend's house is a old, broken down looking house, fitting with the general
look of the neighborhood. The street is to the south.".
```

Now it's time to create our vehicle. With the kind already predefined, it is as easy as typing;

```
a pickup truck is a vehicle in your driveway.
The description of the pickup truck is "A large, red pickup truck which has
served you well over the years you have driven it.".
The initial appearance of the pickup truck is "Here you can see your large, red
pickup truck waiting for you to drive it.".
```

Now the truck can be entered by the player and when the player moves a direction with commands like "go north", "north", or just "n" then the truck will move along with the player. While these a good, we also want to create a new way of moving when we are in the truck as the player will no doubt want to try to "drive" while in the truck. So what we need to do is create a new verb "drive" for the "going" command. We also want the player to be able to drive directly to a room with a similar action we will call "driving to" which will only work when inside a vehicle.

```
Understand "drive [direction]" as going when in the pickup truck.

Driving to is an action applying to one thing. Understand "drive to [any room]"
as driving to.

Check driving to:
if the player is not in a vehicle, say "You need to be in some kind of vehicle
before you can drive anywhere." instead.

Carry out driving to a room:
move the pickup truck to the noun;
move the player to the pickup truck.
```

While we now have a way for the player to move around while inside the vehicle, they are currently unrestricted in where they can move to. Being able to drive a pickup truck into your own living room just doesn't make much sense, so now we should probably place a limit on what type of rooms the truck can enter. This can be done using map "regions" which inform uses to categorize rooms into various types. Here we can create a region called the "road area" along with a few rules that tells inform to only allow the truck to enter rooms in this region.

```
The Road Area is a region.
Your driveway, your street, the neighbors driveway, and the rundown neighborhood
street is in the road area.
```

```
Instead of going to a room not in the road area while in the pickup truck, say
"That area is not exactly car accessible. Best head there on foot.".

Before driving to a room (called the place):
if the place is not in the road area, say "That area is not exactly car
accessible. Best head there on foot." instead.
```

And there we have it, we have created a vehicle which can move quickly on roads but not anywhere else which is exactly the kind of behavior we want for a truck. Now, all that is left is for us to finish our little game by giving the PC a task that can only be completed by using a vehicle. Since the girl's boyfriend lives all the way down town and with the TV probably being pretty heavy, we probably want to keep the player from being able to walk there and bring the TV back on foot and instead get him to use the car using a simple instead rule.

```
Instead of going east while in your street and not in the pickup truck:
say "If your going to go pick up that TV, you're going to need your truck to
get there and carry it back.".

Instead of going west while in the rundown neighborhood street and not in the
pickup truck:
say "It's a little far to be walking all the way home with a TV and leaving
your truck behind.".

A TV is a thing in the dilapidated house. "Here you can see a large TV sitting
on the front lawn which you presume is the one you are here to retrieve.".

Report going to the neighbors driveway while the player holds the TV:
say "Hearing your truck pull up and seeing the TV in the back, your new hot
neighbor comes out to great you and help you unload the TV. You gladly help her
move the TV into her living room where she begins to express her gratitude for
helping her and offers you a nice glass of lemonade for you hard work. After
you graciously accept her offer and watch her walk off waving her tight ass
behind her, you can't help but be disappointed that all you are going to get
for your efforts is a glass of lemonade. Just as you begin to doubt that pornos
are an accurate representation of real life, your neighbor walks back into the
living room holding two glasses of lemonade and to your surprise is no longer
wearing any clothes!

'What?' she asks noticing the surprise on your face. 'You didn't think all I
was going to do to thank you was give you some lemonade did you? Why don't you
take off those pants of yours so I can really show you how grateful I am?' she
says licking her lips and staring at your crotch. Now that is more like it...";
end the game in victory.
```

And there we have it. Now of course this mini-game is very incomplete and needs a lot of polishing, but it should give you a good idea of how to implement vehicles into a larger, more complete game. As you can now see, it's not really as hard as you may have thought and can be very useful in larger games. Anyways, I hope you found this article helpful and as always if you have any specific questions you are more than welcome to email me and ask. I am always willing to help those who seek it if I can. Otherwise, I will be back next month to tackle a new topic which I hope you will find just as helpful.

ADRIFT Segment by BBBen

There would be a number of ways to accomplish the tasks in this month's brief. I don't have one really clear idea of how to do it best, so I'll give you one approach. As in the last few tutorials, I won't explain every little detail, but will just give you the gist of things.

1. The player can walk between rooms like normal, but when in his vehicle, there should be a "drive" command that will take him to a specific room instantly (or rather in one turn as opposed to however many it would take to walk there).

Creating commands like "drive to park" that will transport you to a specific location can be easily accomplished with tasks. Simply use the actions tab to move the player to the relevant location, and give the task a description like "you get in the car and drive to the park". Anywhere you don't want the player to drive, you just don't create a task that allows passage there. Standard movement will still work.

The more complex part is trying to get the car to move around. Objects that are classed as "static" (objects that can't be picked up and carried) can only be moved with events; however, events aren't triggered the same way that tasks are, so that would make it a little fiddly to make the car a static object to move around after the player.

Instead, I would make an integer variable called "carlocation" and put a line at the end of every room description along the lines of "carinpark%carlocation%" or "carinstreet%carlocation%". Then in the ALR file create lines for every location to which the car can travel, like these:

```
carinpark0|

carinpark1|

carinpark2|The car is here.
```

Now the line will only show up when the variable "carlocation" is at number 2. Then in the "drive to park" task, add an action that changes the "carlocation" variable to 2. Repeat this process for all rooms to which the car will travel. Oh, and you'll also have to restrict any "drive to" tasks from working unless the variable is at the right number as well (make sure every location gets a different number, like, say, the park = 2, but the street = 1, and the bar = 3).

2. Make one room that the player cannot walk to (because it's too far away or whatever) and allow him to reach it by driving.

Easy peasy. Just create a room that isn't connected to the normal map and create a task that allows you to "drive to" the location; like "drive to moon".

3. One task involves picking something up from one location and taking it to another. The object in question is too awkward or heavy to be picked up and carried around, so the player will need to drive to that location, load it into the vehicle, and then drive it to the new location.

This is a bit complex again. There would probably be a few ways to do this, but it'll take a little bit of a workaround no matter what you do. First create the object, say, a fridge, and make it dynamic. Then create a task called "pick up fridge" (with "get fridge" and "take fridge" as other options, so it will override normal picking up commands, and just make it say "That is too heavy to carry, but you can load it into the car".

Next make a task called "load fridge into car". Give it the restriction that the "carlocation" variable has to be right for that location (so if the car is supposed to be at the park when you load up the fridge, make the task require that the "carlocation" variable be 2). Next make an action that moves the fridge to room Hidden. Then create another variable called "fridgeincar" and put an action in the "load fridge into car" task that changes this variable to 1.

Now make another task called "unload fridge from car" and make that move the fridge to "same room as player". It should require that the "fridgeincar" variable be 1, and it should have an action that sets that variable to 0.

That should cover it. There's some fiddling around you may have to do to get everything to look right. While these aren't the most challenging objectives in ADRIFT, they are objectives that will require a bit of a work-around to complete. You may need to do a little bit of tweaking beyond what I've already discussed here, but it won't be too hard to smooth over those issues. ◆

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.