THE BASIC ELITE
A Cemetech Production - Cemetech.tk

Newsletter :: Volume I, Number 2 :: June 13, 2005

In This Issue

Staff

Best Amateur TI Website Contest

By Jon Pezzino [jon_p@sbcglobal.net]

June 7, 2005 - The BASIC Elite is proud to sponsor the first "Best Amateur TI Website Contest." This will be a contest in which we will be judging TI sites submitted by applicants. Here are the rules:
1. Your site must be 100% free. We will not accept sites that have paid domains, hosting, maintenance, or otherwise.
2. Your site's focus must be TI-Calculators or anything directly related to them (i.e. programming ASM/Ti-BASIC).
3. Your site must be updated regularly - we will be checking submitted sites to make sure they are updated once a week.

We will be judging the following categories:
-**Best Overall Site** (The site we find to be the overall best)
-**Best Content** (The site with the most quality content to offer, such as programs)
-**Most Aesthetic** (The best-looking site)
-**Best Designed** (The best site in terms of web design, such as the use of DHTML or PHP)
-**Best Community** (The site with the most interesting, outgoing members)

Submit your site to kerm_martian@yahoo.com. The winners will be announced in the June 27th issue of The BASIC Elite Newsletter. Good luck to all entering the contest!"

Latest BASIC Elite Members

(alphabetical by last name)

Rivereye *(apprentice)*: Rivereye is the first member of The BASIC Elite to be appointed to the Apprentice position. This was created for those members interested in joining the Elite who show potential but are not quite at the level of full members yet. Rivereye has released one program so far, an RPG, and is working on some other programs that Cemetech members have suggested.
URL: http://endeavour.zapto.org/rivereye

Hart: The fourth full member to be admitted to the Elite, hart has shown great promise as a TI-BASIC programmer. Although he has not yet released a great deal, his work ethic is strong and he has a good knowledge of the language. He also has an excellent writing style that will make him a valuable member of the Newsletter staff.

Ultimate Pong Challenge

Win a BASIC Elite Membership!

By Jonathan Pezzino

This week, we at the BASIC Elite were presented with a very interesting challenge brought to us by one of our own members, Ti-Freak8x. He suggested creating a pong-like game with an interesting twist: the player must control four paddles to slowly corral the ball into a enclosed area created by the convergence of the four paddles somewhere in the center. Several Cemetech members (including myself) eagerly embraced this challenge, thinking it could be easily completed in a matter of days, if not hours. I quickly found to my dismay that while conceptually this program is simple, the code needed to create it in TI-BASIC quickly becomes bloated, rendering the game useless because it is so slow. Another Cemetech member and newly-minted BASIC Elite apprentice (see article below), Rivereye, seemed to think that said game could in fact by coded such that it would be playable. This prompted me to offer him the incentive of a full BASIC Elite membership should he successfully complete this challenge. Furthermore, I have decided to open up this challenge to the general public. Anyone who submits to me a pong game of the described nature of their own design shall be considered for automatic BASIC Elite Membership. Email your solutions to me at jon_p@sbcglobal.net.

Program Reviews

Filename: **Nibbles II**
Author: Tsukasa X
Email: tsukasah_x@hotmail.com
URL:
http://www.ticalc.org/archives/files/fileinfo/370/37092.html
Released: June 4, 2005
Reviewer: Jonathan Pezzino

Nibbles II claims to be a remake of the classic Nibbles game with a twist: you must avoid both your ever-growing body as well as "bad" food that appears every time you gulp down "good" food. The game plays exactly as it says. While the gameplay in and of itself isn't particularly original or fresh, what sets this program apart is the superb performance of the game. Rarely have I seen a TI-BASIC game run so smoothly and quickly, especially one that uses graphics. Were I given the game without knowing it is given in BASIC, I would have guessed it was written in ASM. This game has a very polished look to it. I don't think it is possible to write a faster snake-game in TI-BASIC, making this quite an accomplishment for the author. The graphics are clean and sensible and the interface is the epitome of simple functionality. Although the game played as the author said it would, I don't think that his "bad" food idea was necessarily the best or most exciting twist for a Nibbles game. Every time you gobble up good food, another bad food appears and the screen becomes quite cluttered in no time. I would prefer if the bad food appeared a little less often so that games could go a little longer without becoming impossibly difficult. Another problem with the strategy this author used was that bad food sometimes appears right in front of your snake, giving you no chance to dodge it. One final minor flaw was that the author eliminated the possibility of the snake going back into itself in a rather annoying way: you die when you accidentally try to go in the opposite direction you were previously going. Overall, this is a great game and a tribute to superior TI-BASIC programming; however, I would like to see the aforementioned issues addressed in the next release.

Graphics: 10/10
Gameplay: 7/10
User Interface: 9/10
Replayability: 6/10
Overall: 8.5/10

File Name: **Lemonade Stand V Ø.67**
Author: Eric Leong
E-Mail: mailto:watson783@optonline.net
URL:
http://www.ticalc.org/archives/files/fileinfo/371/37100.html
Released: June 5, 2005
Reviewer: Rivereye

Review: Judging by the screen shot the program looked simplistic, but when I actually tried it out, I discovered it was much more full-featured than I had initially thought. After playing this game for about ten minutes, I thought to myself, "This game is addicting!" Playing as an intrepid lemonade entrepreneur, you see the weather forecast for the day and choose how much ice, lemons, and sugar you want to stock, as well as the selling price for a cup of lemonade. You can modify your recipe by altering how many ice cubes you use per glass, and lemons and sugar per pitcher. You can also take out ads to promote business. Then you start your day, which lasts for a short time before you must prepare for the next. You must restock ice cubes every day, but sugar, lemons and cups last for a while. Although the author tried to introduce some variety in gameplay by adding random events such as your ice cubes melting, the game tended to be a little repetitive. All in all, not a bad game.

Ratings:
Size: 7/1Ø,
Speed: 8/1Ø
Originality: 5/1Ø
Execution: 7/1Ø
Graphics: 6/1Ø
Ease-of-Use: 8/1Ø
Replayability; 4/1Ø
Overall: 7/1Ø (Very good.)

Filename: **Timerize 2.6**
Author: OutDoneTI
Email: outdoneti@yahoo.com
URL:
http://www.ticalc.org/archives/files/fileinfo/341/34185.html
Released: June 9, 2005
Reviewer: Kerm Martian

While not necessarily the best or most well-developed program for the TI-84 and TI-84+, Timerize 2.6, featured at ticalc.org,

was the first TI-BASIC program to take advantage of the new time capabilities built into every TI-84 series graphing calculator. Some of its features depend on this functionality while others are use the regular programming commands; all, however, have something to do with manipulating or viewing times and dates. The first section of the program has, as any good time program should, a variety of handy ways to look at what time it is. We start with your basic digital clock, on a smart-looking if somewhat cluttered screen along with the date and the current holiday if applicable. Next is a useable analog clock, complete with three hands and the date. The final time-showing component, aimed more at those of us who program than the casual user, displays the time as a series of binary numbers. Interpreting the flashing dots is fun and easy once you get the hang of it

The second major part of Timerize v2.6 is a variety of conversion tools. Among these are options to convert any unit of time (for example seconds, minutes, or days) into any others. Unfortunately, this only goes as high as days; weeks, months, and years are not included in the conversions. It will also find the exact number of days between any two dates, although this too is limited to a century's worth of years.

The third and final section is a pair of programs that respectively count up from Ø as a stopwatch and count down as in a timer. These are fairly useful, but drain your batteries if left going for too long. Also, they are only accurate to the second, not fractions of seconds.

This program is relatively strong and small in size for what it offers, and I feel the programming group put some excellent effort into it. Aside from some aesthetic concerns and the limited conversion features, I would call this an innovative program done well.

Graphics: 6/1Ø
Controls: 8/1Ø
Gameplay: N/A
User Interface: 8/1Ø
Replayability: 5/1Ø
Overall: 6.75/1Ø

Latest Tips & Tricks

Every issue, we will be featuring tips, tricks, hacks, and programming strategies for TI-BASIC that will help you improve both your programs and your overall programming style. Some may be as simple as low-level optimization, others complicated conceptual strategies, but whatever the case, all of our tips are guaranteed to help you become a '1337' TI-BASIC programmer. The Tips and Tricks column is written by Jonathan Pezzino.

This issue, I will be discussing a trick to change pixel stroke size, a method to check for saved data and introduce you to a strategy for mimicking objects a la C++/Java into your program.

Change the Pixel Stroke. This is a documented feature to draw a thicker pixel with the Pt-On command:

```
:Pt-On(X,Y,n
```

When n is 1, the Pt-On command acts as usual. When n is 2, an open box with its center at (X,Y) is drawn. When n is 3, a 3x3 cross is drawn with its center at (X,Y).

Use 'Indicator Lists' to Check for Saved Data

One of the more difficult challenges in TI-BASIC is devising a method for loading saved data from previous instances of your program. The dilemma is this: implement auto-loading and assume that the user was savvy enough to run the installation program ahead of time so that they will not get an error when they run your program or add a Load option, which is awkward for the user because you will often your program to automatically load old data. I offer a third solution to the saved-data dilemma: so called 'Indicator Lists.' An Indicator List is a list of saved data whose existence the program can verify without causing an error if it does not. There are two ways to implement an Indicator List: the 'false initiation' method or the dim(check method. The false initiation method is quite simple. At the beginning of your program, store a dummy value (usually Ø) to the first element of the list you are using to save data then check the dim of the list. If the dim is just 1 (from storing the dummy value), then you know this is the first time the user has run the program and there is no saved data. Otherwise, you know that saved data exists in the list and you may feel free to proceed without fear of crashing your program through the dreaded ERR: UNDEFINED because if the list does not exist, storing to its first element will instantiate it. Here is an example (Note that Lbl 1 would be the point where you know that saved

data exists, otherwise the program continues on as
if no saved data exists):
```
:Ø→LDATA(1
:If 1<dim(LDATA
:Goto 1
:<assume no saved data exists>
```

The only true disadvantage of the false initiation method is that you will need to offset the list indexes of your data by one because the dummy value will otherwise clear the first value of saved data in the list, but this can easily be overcome with a little bit of planning beforehand. The dim(check method is equally easy to implement. Remember that another way to instantiate a list is to store a number to its dim. If the list already exists, doing so will not modify the data stored in it. Thus, you can safely store the desired dimension to your list in the beginning of your program and then check to see if the first value is Ø (there is no saved data) or another value (there is saved data). Here is an example:

```
:1Ø→dim(LDATA
:If Ø=/=LDATA(1
:Goto 1
:<assume no saved data exists>
```

There are a few more cons to the dim(check method, but you will want to use it if it is essential that you need to use the first value of your list data. The dim(check method will not work for lists of unknown size (unless you store 999 to the dim, the maximum size allowed) because you could potentially delete data if the saved list is longer than the dim you store. Another disadvantage of the dim (check method is if the first value of your list could potentially be zero. If such is the case, then the dim(check method will not work unless there is another value in the list which could never be zero were there saved data.

Objects in TI-BASIC

One of the main qualms programmers used to higher level languages (especially Java) will have with BASIC is lack of support for objects. However, you can simulate objects fairly well if you know what you are doing using complex numbers and lists. A complex number has the following properties (also listed are their equivalents in a higher level language):

Positivity/Negativity of real number (boolean)
Integer part of real number (unsigned int)
Decimal part of real number (unsigned int)

Positivity/negativity of imaginary number
(boolean)
Integer part of imaginary number (unsigned int)
Decimal part of imaginary number (unsigned int)

As you can see, one complex number can hold quite a bit of data (six instance fields, to be precise). Now, considering that you are programming on a calculator running at 6 MHz, that is quite a bit of flexibility and power if you know how to unlock it using the real(, imag(, iPart(, and fracPart (commands. Think of how many classes you can emulate with six instance fields; there are quite a few! For example, you could have an enemy spaceship that has an x and y coordinate (determined by Integer & Decimal parts of real number), a directional heading (Integer part of imaginary), a health amount (Decimal part of imaginary), a boss indicator (postivity/negativity of real), and some other indicator (positivity/negativity of imaginary). One could create quite a complex and memory-efficient game following this plan. Unfortunately, this approach is not very fast at all. It takes quite a bit of work to compress/decompress a complex number, making this implementation of objects impractical for games that are in real time. There is however, a solution that sacrifices memory-efficiency for more flexibility and speed. The other way you can implement objects is through parallel lists. In this approach, you create a series of lists, each one representing one instance field. For example, you might create LX, LY, LDIR, and LHEA if you were trying to create a game similar to the one described above. An object is represented by the values of the lists at a specific index. For example, the object with value one would have an x value of LX(1), a y value of LY (1), etc. As you can tell, this approach is much more memory-hogging, but it is worth it if you are more concerned about speed than memory-efficiency. Another advantage of this approach is that you are not limited to the number of instance fields you can have or their type. You simply create a new list for every variable you want to have.