RandomStream.bas

Britlion's Crazy Random Number generators! (Based completely on the stream generator from Patrik Rak, and much thanks for his work on this)

Boriel has included a better random function in the code; but this passes through floating point numbers, which is potentially fairly slow - and for games we usually require integer numbers anyway!

I've written a few functions that are a possible alternative.

This is the base function that does the hard work of generating a random number from 0-255 in the A register (or as a return value, conveniently enough). This is the same random number generator that Boriel is using, incidentally (based pretty much wholly on Patrik Rak's stream random generator, as posted on the World of Spectrum Forums).

Update: Tweaked for Einar Saukas' optimization, September 2012.

The following function, randomBase, returns a pseudorandom value between 0 and 255 - that is a one byte return. This is the base of Patrik Rak's random stream generator, and is the fastest function here. Other functions will call this one. **They will also call it FROM MACHINE CODE - so expect the ASM context label "random" to be there. If you change this label, you will have to also change the calling functions**

```
FUNCTION FASTCALL randomBase () AS UBYTE
ASM
random:
  ld  de,$A280   ; xz -> yw
  ld  hl,$C0DE   ; yw -> zt
  ld  (random+1),hl  ; x = y, z = w
  ld  a,l        ; w = w ^ ( w << 3 )
  add a,a
  add a,a
  add a,a
  xor l
  ld  l,a
  ld  a,d        ; t = x ^ (x << 1)
  add a,a
  xor d
  ld  h,a
  rra            ; t = t ^ (t >> 1) ^ w
  xor h
  xor l
  ld  h,e        ; y = z
  ld  l,a        ; w = t
  ld  (random+4),hl
END ASM
END FUNCTION
```

This function will update the seed value based on the current frames counter. To improve randomness, get the user to have a human interaction that can take a variable amount of time and then run this.

```
SUB FASTCALL updateSeed()
REM Updates the random generator seed from the FRAMES system variable.
time()
ASM
   LD A,E
   EX DE,HL
   LD HL,random+2
   XOR (HL)
   AND A
   JR NZ,updateSeedNotZero
   INC A
updateSeedNotZero:
   LD (HL),A
   LD HL,random+4
   LD A,E
   XOR (HL)
   LD (HL),A
   INC HL
   LD A,D
   XOR (HL)
   LD (HL),A
END ASM
END SUB
```

The above function requires the timer function, which simply grabs the time from the frames variable and returns is as a unsigned-long variable, in registers DEHL:

```
FUNCTION FASTCALL time() as uLong
asm
    DI
    LD DE,(23674)
    LD D,0
    LD HL,(23672)
    EI
end asm
end function
```

This function returns a value from zero to the specified limit number (limit <= 255). You can therefore, for example, roll a dice by calling `randomLimit(5) + 1` to get 1-6.

```
FUNCTION fastcall randomLimit(limit as uByte) as uByte
ASM
    AND A
    RET Z ; Input zero, output zero.
    LD B,A ; Save A

    LD C,255
    randomBinLoop:
    RLA
    JR C, randomBinLoopExit
    RR C
    JR randomBinLoop ; loop back until we find a bit.
    randomBinLoopExit:

    randomBinRedoCall:
    call random
    AND C
    CP B
    RET Z
    JR NC, randomBinRedoCall
END ASM
END FUNCTION
```

It's worth noting that the issue with the above is that it basically rolls a random, and if it's bigger than limit, it rolls another one. This could potentially take a while, and isn't guaranteed to be fast. Though the probability of failing to hit the zone is kept to 50% at worst, so on average it will roll 1.5 random numbers, I think, per call. This should usually be faster than a floating point multiply, I believe.

If you want, in a similar way to sinclair basic and ZX BASIC RND function, a number between 0 and 1, this function provides that, using a FIXED type return. This is usually good enough for most purposes, but is quite a lot faster to process than a full floating point number.

```
FUNCTION FASTCALL randomFixed() as FIXED
ASM
    call random
    push AF
    call random
    ld l,A
    POP AF
    ld h,a
    ld d,0
    ld e,d
END ASM
END FUNCTION
```