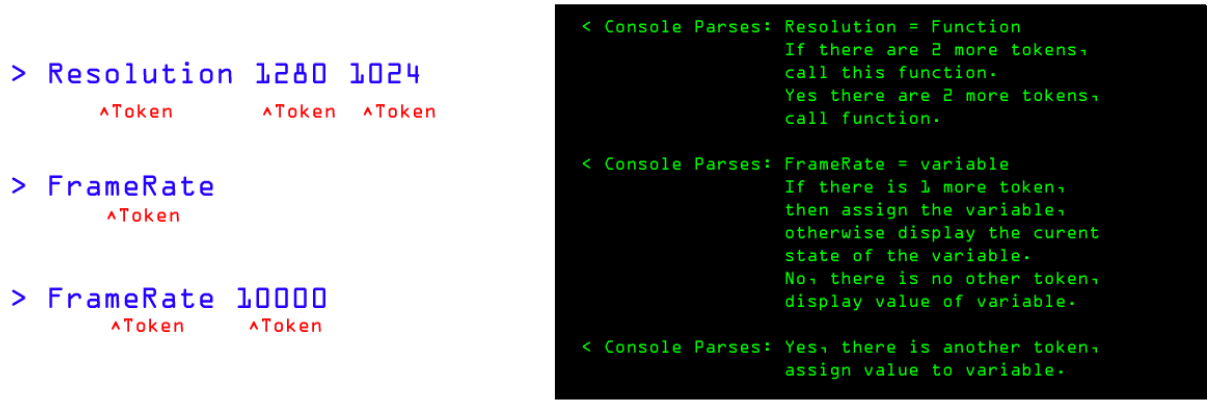
The console will be a multi-tasking part of the in-game CAS engine. After scrolling down, the console occupies half of the screen. It is used as an interface to any part of the game, allowing access to debugging variables and giving the player the freedom that he or she wants.

The technology behind the console is a symbol table. Association structures take a string and assign it to a wide variety of things, mostly variables and functions. Using a red-black tree to store the association table, we are guaranteed O (n*lg (n)) efficiency searching for a symbol (the cost of string comparisons is not part of the asymptotic run-time cost; since the maximum size of a string is known it has only a constant cost). Once we find a symbol match, we compare the number of following tokens required to complete that symbol. If the symbol can be completed, the action is executed, and the console command is successful.




The structure for the association has the knowledge required to make informed decisions about the type of the string. If a string token is stored as a variable in the table, then it could mean possibly three things:

1) The user wants to know the current value of that variable.
2) The variable is being assigned a value.
3) The variable is being passed as a parameter to a function.

Similarly, if a string token is stored as a function, it could mean:

1) The user is calling the function.
2) The function is a pointer, and is being assigned to point to a different function
3) The function is being passed as a parameter to a function.

The in-game fonts are 2D, but will be rendered as 3D objects. So it is the job of the console to parse the string tokens into flat polygons to be passed as billboarded sprites for the graphics engine. There will be two triangles used for every character, and the console will algorithmically texture map the characters to the triangles. The depth of the billboard will be equal to the front clip plane, and will be put on the top of the rendering stack, in order for the fonts to always be rendered on top of everything else.

The console will be very versatile with great functionality. Every variable, including game state statistics, can be accessed from the console, giving the programmers high level functionality for low-level routines. It is a common part of many first person shooters, and has proven to be a viable commodity time and time again.


Above: Bad quality mock up screen shot of the console.

The structures for the console:

```
typedef char scanline[100];

struct TokenInfo
{
   bool IsVariable;
   bool IsFunction;

   void *MemoryLocation;

   int NumberOfQualifyingTokens;

   char TokenSizes[10];
};

struct Console
{
   // Contains a history of previous commands.
   list<scanline> ConsoleScanlineList;

   // Contains all the possible console
   // commands and variables.
   map<string,token_info> ConsoleCommandRedBlackTree;

   // Contains the previous console commands,
   // parsed in a Red-Black Binary Tree,
   // allowing fast lookup when you want
   // auto-completion on a command.
   map<string,string> ConsoleLookup;
};
```

The console also allows auto completion, and pop-up tables of previously executed commands that start with the currently typed in scanline.