TI-83+ Z80 ASM for the Absolute Beginner

LESSON SIX:

- *ASM Gorillas, Part I: The Planning Stages*
- *ASM Gorillas, Part II: Variables and Constants*
- *Looking Ahead*

ASM GORILLAS, PART I: THE PLANNING STAGES

If you want to write a great game—whether Ti-Basic or ASM, whether C++ or Visual Basic—you should plan the game first. If you don't plan what you want to do, if you blindly start drawing pictures and coding without having any idea of what you're up against, you'll either give up a project or you'll spend longer on it than you intended.

For example, a good friend of mine started writing a program that would translate QBasic (Microsoft's special version of Basic) into ASM. While he had a great knowledge of QBasic and ASM, he failed to plan ahead: whether he could handle the project, what kind of time he would need, how to proceed, etc. After only two months of programming, he canceled the project permanently, much to the dismay of many Ti-83+ programmers.

Planning a game doesn't necessarily mean deciding what you're going to do every single day of your life, but it does mean having a general idea of how to proceed. My current project, S.A.D., required about six months of planning before the actual programming began.

Are you going to need six months to plan ASM Gorillas? Heaven forbid! I'd say only a few pages of planning are needed. However, it is needed. Even if you've done a major Ti-Basic program before, ASM programs require even more planning.

Be aware that some of the stuff we plan will not be programmed immediately. But you need to be ready with what you want to do for the time that you're ready to do it. Do you see what I mean? ☺ Oh, and be sure to play http://www.kongregate.com/games/Moly/gorillas-bas again if you need to remember the game "Gorillas."

Overview—Player the Game: The game consists of two players. Each player plays as a gorilla, armed with nothing but exploding bananas. The object of the game is to hit your opponent's gorilla with bananas more than his gorilla hits yours.

The game is played in several rounds. Every round, each of the two gorillas is placed on a random building on each side. Each player takes a turn to enter velocity—how hard his gorilla should throw an exploding banana—and the angle he wants his gorilla to throw the banana. The gorilla then throws the banana. If a banana goes to high or too far, or if a banana hits a building, the player is unsuccessful and has to wait while the other player has a turn. If a banana hits an opposing gorilla, the player who threw the banana gains a point, and the next round commences.

The game ends when all rounds have finished, and the player with the most points wins. In case of a tie, a tiebreaker round will occur to determine the winner of the game.

Main Menu: The game will have no splash screen. However, the main menu will have a background image. There will be four options: New Game, Load Game, Settings, and Quit.

Settings Sub-Menu: This menu will allow players to select wind speed ranges and set the gravity. Wind speed ranges will allow anything from strong winds to no winds to random winds, which will affect how fast the thrown banana is traveling. Gravity can be specified as a number.

New Game Sub-Menu: The player can either select a one player vs. AI game, a one player vs. one player game, or go back to the main menu

New Game Sub-Menu 2: Players can input their names, as well as the number of rounds.

Loading the Game: Every round, a “Please Wait…” screen will display in order to load the game. The game loads by drawing 12 buildings of different heights, placing the gorillas on the buildings, drawing the sun, and determining the wind speed.

Graphics: Most graphics are 8 x 8 images. Gorillas are 8 x 8 in size, and buildings are composed of selections of a collection of 8 x 8 parts. The sun will be bigger than 8 x 8.

Throwing a Banana: When the player enters the velocity and angle that his gorilla should throw the banana, the position for the banana is calculated one pixel at a time, taking into account velocity, angle, wind speed and gravity. If the player’s banana hits the sun, the sun will change its face reaction, but neither the sun nor the banana will be affected. If the banana hits a building, the building will be partially destroyed by the banana’s explosion.

ASM GORILLAS, PART II: VARIABLES AND CONSTANTS

Now that we have the general idea of the game, we need to proceed in an orderly fashion. Since you've only learned about variables and constants so far, we'll look at what variables and constants are needed for the game.

Now, note that planning is not 100% "I've got it all together." There will be some variables and constants we might, or will, add later that we won't be able to think of immediately. For sure, this includes variables we will add for sound and for drawing buildings.

From the planning, we know there are a lot of menus in the game. Menu_X and Menu_Y will be variables use to hold the position of the cursor at these menus.

We need variables to hold the range of wind speeds and the selected gravity from the menu. The variable for this "wind speed" will be a constant (called Wind_Speed_Multiplier) multiplied by a random number to determine the strength of the wind per round. If the player wants no wind, the constant will be equal to zero, which means that 0 multiplied by the random number is zero. If the player wants strong wind, the constant will equal six, which, when multiplied by a random number, creates a greater wind speed. Gravity ranges from 1 to 20, with 9.8 by default. Gravity can only be entered in precision of tenths, and is a variable simply called "Gravity." Our variable will hold this value multiplied by 10.

While we need variables for the number of human players and the names of players, we will add these later. Be aware, however, that these need to be added.

Since the positions of the gorillas and the bananas changes constantly, we need six variables for these positions: an X, Y for the banana, and an X, Y for each gorilla. For these variables (to practice future lessons), we will simply refer to them as Banana and Gorilla_1, Gorilla_2. Thus each variable will have TWO .dbs for two bytes of data, one byte for X and one byte for Y. We will also need a variable to hold the random wind speed selected for the particular round. Finally, we need variables to hold the velocity and angle of the tossed banana.

Each player needs a variable to hold their score. Of course, this means a variable is needed for the number of rounds.

We need one variable to hold data for animating the bananas and the gorillas. We'll talk later about what this variable does and why we need only one variable.

With all this in mind, let's put all these variables in your program. Type the code on the next page, and save your file as ASMGorillasMain.asm.

```
#include “ti83plus.inc”
.org  40339
.db    t2ByteTok, tAsmCmp

Menu_X:  ; Holds the X and Y positions for the cursor in menus
 .db 0
Menu_Y:
 .db 0
Wind_Speed_Multiplier:  ; A constant used in generating wind speeds
 .db 4   ; 4 is the default, medium winds
Gravity:
 .db 98  ; Gravity is 9.8 meters per second by default
Banana:  ; The X Position and the Y Position of the Banana that is thrown
 .db 0, 0
; X and Y positions of each of the player’s gorillas
Gorilla_1:
 .db 0, 0
Gorilla_2:
 .db 0, 0
Wind_Speed:
 .db 0
Velocity:
 .db 0
Angle:
 .db 0
Score:  ; Two bytes, one byte for each player’s score
 .db 0, 0
Number_Of_Round:
 .db 3  ; The default number of rounds is 3
Animation_Frames:
.db 0
```

Some of our values for the game ASM Gorillas will never change, so let's program some of the constants we'll need. By using constants, we will never have to remember numbers: we will only need to remember words. Which is easier to remember: 700 million miles an hour, or Speed_Of_Light?

Take out your calculator, and set the following coordinates: Xmin=0, Xmax=94, Xscl=1, Ymin=-62, Ymax=0, and Yscl=1. Move your cursor to X=11 and Y=-14, and type in "NEW GAME." This will allow you to visualize where the first part of the menu is going to be displayed. Our Main Menu, and our Sub-Menus will have their text displayed at X=11 and Y = 14 (With ASM, one doesn't need negative numbers for displaying text). So let's create some constants to hold these values.

Create a new text file called "ASMGorillasConstants.asm" and type in the following:

```
MainMenuItem1X .equ 11

MainMenuItem1Y .equ 14
```

Also, go to the main file, ASMGorillasMain, and after #include "ti83plus.inc", type #include "ASMGorillasConstants.asm"

Let's work on some more constants! We want the calculator to know how many items each menu has. The Main Menu has 4 items, and the Settings, Players and Names Sub-Menus have three items each.

We also want the calculator to know what menu it is working with. We identify each menu with a number. Add the following to ASMGorillasConstants:

```
Main_Menu .equ 0
Settings_Menu .equ1
Players_Menu .equ 2
Names_Menu .equ 3

Main_Menu_Items .equ 4
Settings_Menu_Items .equ 3
Players_Menu_Items .equ Settings_Menu_Items
Names_Menu_Items .equ Player_Menu_Items
```

Notice that you can assign a constant to a constant? That's because a constant is just a permanent number.

We'll need four X,Y groups of constants for displaying text in-game. Two groups are used for the location of the Player's Names/Angle/Velocity, and two are used for displaying score if the player wants to see the score.

```
Player1NameX .equ 0
Player1NameY .equ 0
Player2NameX .equ 51
Player2NameY .equ 0
Player1ScoreX .equ 0
Player1ScoreY .equ 58
Player2ScoreX .equ 83
Player2ScoreY .equ 58
```

Finally, we need an X and Y for displaying the text for “Loading,” which will be big text rather than small text. Since we’re using big text to display the “Loading” text, we use X = 0 to 15 and Y = 0 to 7.

```
LoadingTextX .equ 3
```

```
LoadingTextY .equ 3
```

LOOKING AHEAD

Remember the three registers that we looked at in Lesson 4? We talked about register A, which is one byte and is responsible for much of the CPU's math. We also talked about H and L. H and L are also one byte, and therefore cannot be less than 0 or greater than 255.

However, the calculator has more registers than A, H and L. We'll look at some of these registers next week, as well as some more instructions for writing programs.