**Program 6**

**Due Fri, Dec 10th at class time (no late work allowed)**

**40 pts**

**You may pair up with *one* person for this project.**

Write an alien-shooter game as follows:

* The shooter will have a cannon positioned in the lower-left corner of the screen. The location of the cannon cannot change, but the user may change the *angle* of the cannon barrel by pushing the **up** or **down** arrows. (hint: see your cannonball lab for the math/logic of aiming the cannon). The cannon can be represented by a simple straight line, with one end anchored at the lower left corner of the screen, and the other end positioned as directed by the user.
* The user may only fire one cannon-ball at a time. The user fires by hitting the space bar. The cannon ball travels in a ballistic path (exactly like it did for your cannonball lab). You may hard code the speed of the cannon ball (experiment to find a speed that works well with the game play). (Examine your code for the cannonball lab to get the ball to travel in a proper path).
* On the right hand side of the screen, the game should position 10 aliens in a row at the top of the screen. Every two seconds, one alien from the top row should be picked at random to begin moving down the screen. The position of moving aliens should be updated every second (you can decide the distance that the aliens move each second). Note that although the aliens begin their life in a row, they don't all move down together since only one begins moving every two seconds. It will take 20 seconds before all 10 aliens are moving.
* If the cannonball hits an alien, the alien should disappear from the screen (it is dead). One cannonball may kill multiple aliens.
* After all 10 aliens are off the screen (either because they have been killed or reached the bottom of the screen), your game should automatically start level 2. Level 2 restarts the 10 aliens at the top of the screen (i.e., it looks just like level 1). *However, level 2 aliens move faster*. At all times, the upper left corner of the screen should show the current game level.
* Also, in the upper left corner of the screen, you should maintain a lives counter. The lives counter decrements every time an alien reaches the bottom of the screen without being hit by a cannonball. When the lives counter reaches zero, the game is over (an appropriate message should be displayed on the screen). You can decide how many lives the player gets, but it should be small enough so that the aliens can win the game.
* In the upper left corner of the screen, you should maintain a kill counter. If all 20 aliens are hit before the lives counter reaches zero, then the human wins the game.
* When the game is over, the winner should be displayed on the screen.

**Hints:**

1. Since these DarkGDK programs require extra project settings, I have provided you with a pre-configured project (Program6) on bboard. Download this project and use it for your program.
2. In addition to the dbXXX functions illustrated in your lab10 projects, you will need the following additional functions:

**dbHideSprite(sprite number)**

This function hides the specified sprite. This is useful following a collision.

**dbShowSprite(sprite number)**

This function shows the specified sprite (undoes the effect of dbHideSprite).

**dbCloneSprite(sprite-to-be-cloned, number to be assigned to the new sprite);**

This function creates a new sprite from an existing sprite. Once cloned, the sprites have no relationship to one another.

**bool dbSpriteCollission(first sprite to be tested, second sprite to be tested);**

This function returns true if the indicated sprites have collided.

**itoa(integer to put into a string, char [], base);**

This stands for "int-to-ascii". This is not a graphics function; it is a C++ library function that converts an integer into a string. You must pass it a character array into which the string will be placed. This is useful so that you can display an int value (i.e., first convert it to a string, then use the dbText function to display the string).

**dbText( x coord, y coord, char string-to-be-displayed[ ] )**

This function displays the specified string at the given screen coordinates.

1. Create your own graphics to use for the spites. You may use Paint or any other painting program you want. Sprites should be small (say, not over 30x30). If your program supports it, make the background pixels transparent. Alternatively, making the background pixels black will be fine. DO NOT SPEND ALL YOUR TIME MAKING COOL PICTURES; SIMPLE FIGURES ARE FINE. Put the *graphics files in the same folder with your source code*.

**Grading:**

Your grade depends on the functionality of your application and the design of the application. Break your program up into manageable, understandable functions with appropriate parameter lists, using the proper style guidelines that we have emphasized all semester.

**Submit Instructions**

PAY ATTENTION: THIS IS DIFFERENT THAN THE NORMAL PROCEDURE. IMPROPER SUBMITS CANNOT BE GRADED.

Step 1: In Windows Explorer, locate your Program6 solution folder for this program.

Step 2: If there are known problems with your program (e.g., it crashes when I do this…) then these problems must be documented in a Word file named PROBLEMS. Put this Word file at the top level of your Solution folder.

Step 3: Click RIGHT on the solution folder, then select **Send To** from the pop-up menu, then select **Compressed (zipped) Folder**. This will create a new zip file in the same folder that contains the Program6 folder.

Step 4: Rename this zip file using your lastname.zip (or, if you worked in a team, the file should be named lastname1\_lastname2.zip).

Step 5: Submit the zip file to the appropriate bboard assignment link.